Lesson 2: Basic Database Design

## INTRODUCTION

In this lesson, you will learn about basic database design. This lesson is designed to provide you with foundational information needed to understand the basic structure of a database including an eight step process of how to design a database to store geospatial data.

## LESSON OBJECTIVES

By the end of this lesson, you will be able to:

1. Define basic database design terms.

2. Create a database schema.

3. Import and export a database schema.

2. Design a relational database.

## LEARNING SEQUENCE

|  |  |
| --- | --- |
|  | |
| Required Reading | Read the following:  Basic Database Design   * Database Design and Construction |
| Assignments | Complete the following assignments:   * Lab 2: Basic Geodatabase Design * Quiz 2: Basic Database Design |

## INSTRUCTION

## Database Design and Construction

### Schema

A database schema is the structure of a database described by integrity rules. We can think of a schema as a blueprint to a database. It does not contain any data, however it does contain all the structure and rules required to hold all the data. A database must adhere to a set of rules defined by the administrator or database manager.

* It is important that the database administrator or manager design an appropriate schema with rigid rules to control what can be stored as a data value in the database. When a schema is constructed, they can be imported and exported from databases for easy replication.
* It is important to understand how records are uniquely identified in a database. Since the database is composed of tables and relationships, it is important that each table, row, or record in a table be able to be referenced uniquely. The way a record is uniquely identified is through use of a key. A key is a unique identifier for a record in a database. There are three kinds of keys: primary, foreign, and composite.

### Primary Keys

A primary key is a unique identifier for each record in a table. The purpose of the primary key is to allow the user and database to uniquely identify specific records in the table. A primary key can also be used to organize the records within the table by sorting the entire table by the primary key column in ascending, descending, or another sorting method. In database design, the primary key is denoted by the letters ‘PK’, or a key icon.

In this example patient ID is the common attribute that contains the primary key for the patients table. In this table, each patient the hospital is provided a unique identifier and provides an unambiguous way to uniquely identify each patient in this and other tables.

### 

### Foreign Keys

A foreign key is a value used to establish a relationship of one table to another table that uses that key as the primary key. In other words, a foreign key does not uniquely identify the records in a table that it is located in. It allows its table to be linked to another table that uses that key as the primary key. In database design, the foreign key is often denoted as ‘FK’.

### 

### Composite Keys

A composite key is a key that is composed of two or more columns in a table. Sometimes a column does not provide enough information to uniquely identify its row. In this case, multiple columns can be combined in a certain way so that when using all the combined columns it creates a composite key allowing for each row to be uniquely identified.

### Integrity Rules

### Now that we are aware of the basic idea of keys we can learn about some integrity rules that can be enacted in a database. These integrity rules can be assigned to a single column or multiple columns in a database and provide a mechanism to restrict what values can be stored in these columns in the database. It is important to restrict values in a database so that you have strong data integrity.

#### Null Value

### The null value integrity rule can be defined on a single column and can be set to either allow or disallow null values in that column.

#### Unique Column Values

### A unique column value is another integrity rule. With this rule a column will only allow unique values to be entered and not a value that is been entered on any other record on that table in that column.

#### Primary Key Values

### Most tables in the database will have a primary key, requiring that each record in the table be uniquely identified. If the table should have a primary key, the primary key integrity rule can be placed in effect on the table.

#### Complex Integrity Checking

### There can be complex integrity checking which allows the user, or database administrator to define rules for columns that do or do not allow for changing of the records depending on the integrity rules that are set.

### Domain Integrity

Related to integrity rules are domain integrity. For each column a domain of values can be specified which means that only values within the domain list can be entered in that column. An example would be if the column records the age of a person. The domain of values would be any positive integer since it would not make sense for an age to be a negative number.

You can also specify an upper limit on the age to guard against the possibility of someone entering in an age of, say, 300 years. In addition to values a domain can also specify specific formats such as requiring seven digits for Social Security numbers with hyphens after the third and fifth number.

### Entity Integrity

Switching from columns to rows in the database table, we come to the concept of entity integrity. Entity integrity deals with maintaining data integrity for each record in a table. In a properly designed database, each entity should be uniquely identifiable, which is done through a primary key.

In order to make the primary key have integrity the database should not allow the primary key to be null and it must also force the primary key to be unique within a table. Without this entity integrity we would have duplicate primary keys which would not allow for a user or database to uniquely identify a record.

### Referential Integrity

Since a foreign key is used to relate one table to another, we must enforce referential integrity rules to maintain this relationship. To enforce referential integrity on foreign keys, if a foreign key should exist, it must exist as a primary key in a record of another table. This ensures that the relationships amongst tables remains constant and that a foreign key cannot be orphaned in the database.

### Referential Rules

As tables are related to each other in a database, referential rules can come into effect. Let’s review a few referential rules commonly used in database design.

#### Restrict

The first reference rule is the restrict rule which does not allow for the change of specific data in a database.

#### Set to Null

Set to null means that when the reference data is updated or deleted, all of the associated dependent data is set to null that way it is not orphaned in the database.

#### Default

The default rule means that when the reference data is updated or deleted, all of the associated dependent data is set to a default value of the database.

#### Cascade

The cascade rule is in effect when the reference data is updated and all associated dependent data in another table is correspondingly updated. In other words, when referenced rows are deleted from a table all other rows pointing to that data with their foreign keys are also deleted from the database.

#### No Action

There is the no action referential rule which does not allow for the change of reference data, meaning that if the record it references is deleted, it itself is not deleted.

### 8 Steps to Design a Database

To design a database you need to follow these eight steps:

Step 1: Determine the purpose of the database.

Step 2: Find and organize the required data for your database’s subject.

Step 3: Divide the data into tables.

Step 4: Turn the data items into cell columns.

Step 5: Specify the unique identifiers of your data.

Step 6: Set the table relationships.

Step 7: Evaluate your design.

Step 8: Apply normalization rules.

The next several slides will explain each of the eight steps in detail.

### Step 1: Determining the Purpose

The first step in designing a database is to determine the purpose of the database. At this point you should answer these important questions. First, what is the ideal mission statement the database? In other words, how could its use be summarized?

Next, you need to determine what you wish to accomplish with this database, what the end result is, and who the users of the database will be. For example, a mission statement for our sample database is that the database will keep track of all of the doctors working in the hospital and the department they belong to along with their patients for scheduling purposes.

### Step 2: Organize

The second step in designing a database is to organize all of the data pertaining to our subject. The data could exist in the form of ledgers, handwritten notes, shape files, other databases, pictures, video files, and so on. You must imagine all the information you will need to collect from your users, such as patient information, doctor information, department information, and so on.

You also need to determine what product you want to return to the user. Is it a map, a spreadsheet, or a video? You should also ask which questions you want the database to answer such as “where are the patients that a particular doctor is taking care of?”

### Step 3: Dividing into Tables

When you have determined the purpose of your database and collected all of the required information, the next step is to determine how the database will be divided into separate logical tables. Each table in a database should contain information with respect to a single theme. As an example we could have a patients table, a doctors table, and the departments table. Once the subjects or main topics are chosen, we need to create columns within these tables. The purpose of the columns is to store the descriptive information for each record in that table.

### Step 4: Cell Columns

Now that we have the database broken in to tables, we should focus our attention on each individual column and what information we want to store in each cell in that column. At this point you should also determine whether complex information should be stored in one column, or broken up into multiple columns. A home address can have five separate entities such as address, city, state, zip, and country. This could be stored as five separate columns, or a single column. Depending on the purpose of the database, you would need to choose how granular you would like to have the information in each column and table.

### Step 5: Unique Identifications

The fifth step in designing a database is to determine which columns will be the unique identifiers for the records. Each table will have what is considered a primary key or some sort of unique identifier. Remember that the primary key only works if all entries in the primary key field are different.

If you are in a situation where you do not have a good column to act as a primary key it is common for databases to have an auto number format or some other way to automatically assign unique identifiers to each record. It is typically recommended to avoid the auto number format and instead try to choose a meaningful primary key such as a case number for crime, employee number, a patient number, and so on.

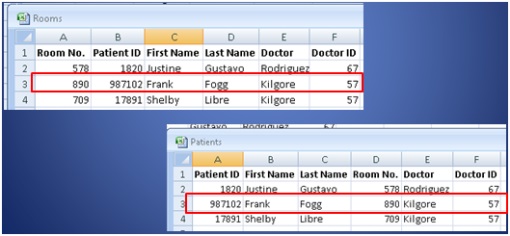
### Step 6: Table Relationships

With the details of each table determined, it is now time to turn our attention to table relationships. The power of relational databases is the ability for tables to reference each other which provides for powerful querying analysis. When determining how you wish to have tables relate to each other there are many things to consider.

The first thing to consider is the cardinality of the relationship. The relationship cardinality refers to how many records will be included from each table participating in the relationship. There are three possible cardinalities: one-to-one relationship, one-to-many relationship, and many-to-many relationship. These relationships are often denoted as 1:1, 1: n, m: n respectively. Each one of these cardinalities is covered in some more detail in this lesson.

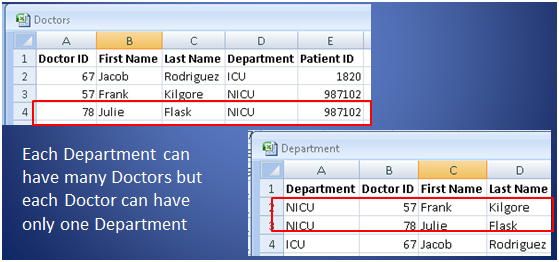
#### One-to-One

In the one-to-one relationship each record in one table can appear only once in another table. For instance in the rooms table you can only have one patient per room, and in the patients table, the patient can only be assigned to one room.



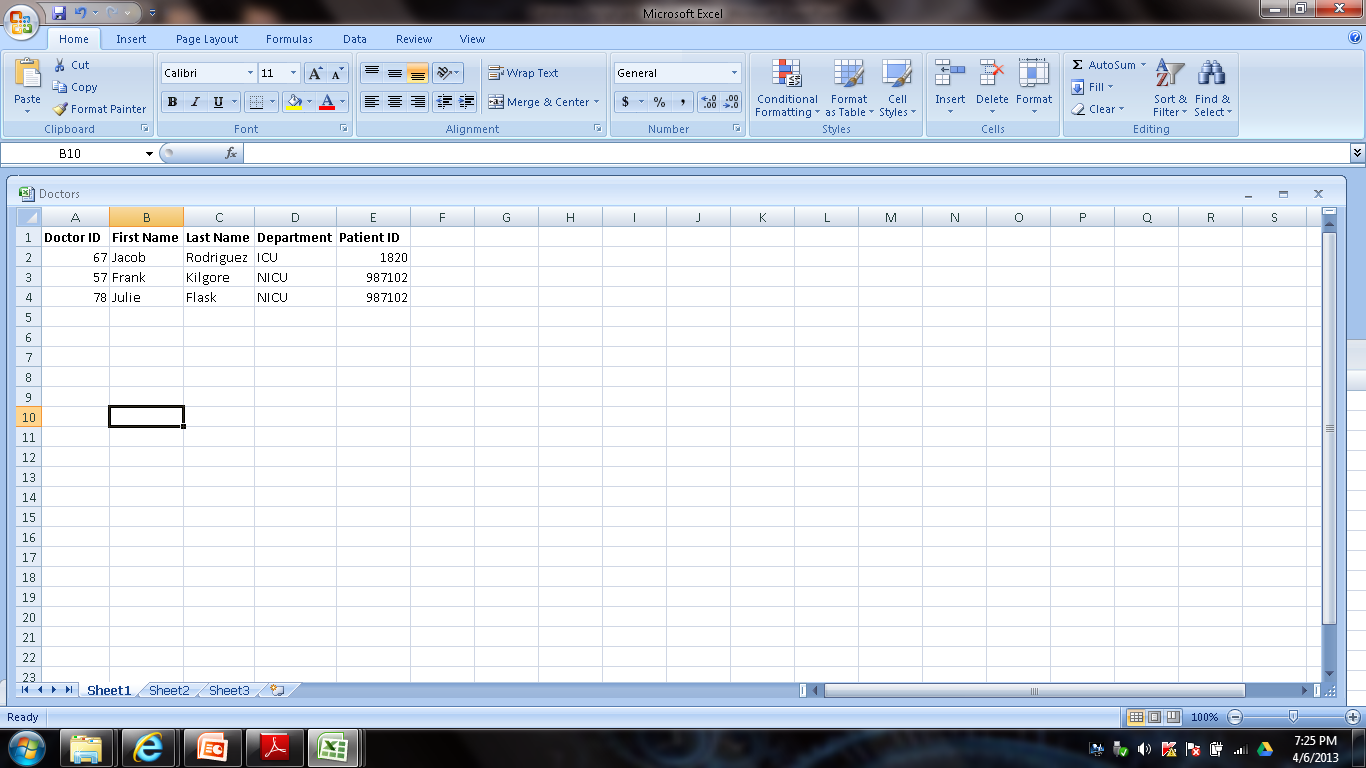
#### One-to-Many

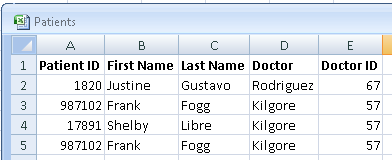
In the one-to-many relationship each record in one table can be related to many rows in another table. For instance each department can have many doctors, but each doctor can only have one department.



#### Many-to-Many

### In the many-to-many relationship, many records in one table can be related to many records in another table. For example, each doctor can have many patients, and each patient can have many doctors.





### Step 7: Refining Design

The seventh step in designing databases is to refine the design. With the tables organized just right, they should be populated with the proper records. Then revisit each table to ensure that nothing was forgotten, unnecessary information is not be included, and that all cells are being populated and all integrity rules are being followed. Basically, you do a complete check to ensure that everything is in order, and if it is not, then you would need to change the database schema.

### Step 8: Normalization Rules

### The final step in a database is normalization. Relational databases are considered properly designed when the data has been divided into tables to reduce redundancy of information, among other reasons. During normalization, you should review tables, and consider splitting them into multiple tables to reduce the amount of redundancy in the database. There are three common stages of normalization, and we will discuss these three in more detail. We should note that there are other forms of normalization, but these three should typically be achieved at a minimum.

### Normalization: First Normal Form

To achieve the first normal form in your database, you should ensure that each cell only contains one value and not multiple values. For example, if we look at the items column in this table, we can see that there are three value stored in a single column. Because there are multiple values in a single column, this table is not in the first normal form.

|  |  |  |
| --- | --- | --- |
| **Order** | **Date** | **Items** |
| 254 | 10/24/2011 | Corn, Apple, Bread |

#### No Repeated Columns

### The second rule to making tables conform to the first normal form is to not have any repeated columns. This is a modified version of the orders table in which we resolved the issue of having multiple values in a single cell. The problem now though, is that we have repeated columns: one for each item purchased. Besides the issue of potentially needing hundreds of column to handle very large orders, thereby wasting many columns for very small orders, it is difficult to do any sort of summarizing of what the customer purchased as we would have to check every single item column which might take a long time.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order | Date | Item 1 | Item 2 | Item 3 |
| 254 | 10/24/2011 | Corn | Apple | Bread |

#### Normalized Table

### This is the order table normalized to the first normal form. The order is broken into three different records, one record for each item purchased on that order number. This meets the requirements of the first normal form since there is a single value in each cell and there are no repeated columns.

|  |  |  |
| --- | --- | --- |
| Order | Date | Item |
| 254 | 10/24/2011 | Corn |
| 254 | 10/24/2011 | Apple |
| 254 | 10/24/2011 | Bread |

### Normalization: Second Normal Form

### In order for table to be in the second normal form, it must first be in the first normal form and there should be no partial dependencies, meaning that all attributes in a table must be fully dependent upon the primary key in its row. In other words, a table should only store data relating to a single “thing” and that “thing” should be entirely described by its primary key.

#### Composite Key

### This is a more complicated example of the order table. In this table the primary key is actually a composite key of the OrderID and Item# columns. That means that in order to be in the second normal form each non-key column (every column other than OrderID and Item#) must be fully dependent on the primary key.

### In other words, does the value of OrderID and Item# imply the value of every other column on that row? The answer in this case is no, they do not. As you may have noticed, if we only know the OrderID, then we know the customer and the date of the order without having to know the Item#. Therefore, the date and customer columns do not depend on both columns composing the composite key. Because of this, this table is not in the second form.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OrderID | Date | Item# | CustomerID | Quantity | Item |
| 254 | 10/24/2011 | 1 | 45 | 1 | Milk |
| 254 | 10/24/2011 | 2 | 45 | 2 | Cookie |
| 254 | 10/24/2011 | 3 | 45 | 1 | Cola |
| 125 | 10/22/2011 | 1 | 13 | 4 | Orange |
| 264 | 10/12/2011 | 1 | 98 | 3 | Deli Meat |
| 264 | 10/12/2011 | 2 | 98 | 1 | Bread |

#### Decomposition

In order to achieve the second normal form, we must break this table into two tables which is known as decomposition. To properly decompose the tables to achieve the second normal form, we put everything that applies to each order in one table, and everything that applies to each order item in the second table thereby achieving the second normal form.

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Item#** | **Quantity** | **Item** |
| 254 | 1 | 1 | Milk |
| 254 | 2 | 2 | Cookie |
| 254 | 3 | 1 | Cola |
| 125 | 1 | 4 | Orange |
| 264 | 1 | 3 | Deli Meat |
| 264 | 2 | 1 | Bread |

|  |  |  |
| --- | --- | --- |
| **OrderID** | **Date** | **CustomerID** |
| 254 | 10/24/2011 | 45 |
| 125 | 10/22/2011 | 13 |
| 264 | 10/12/2011 | 98 |

### Normalization: Third Normal Form

To achieve the third normal form, the table must be in the second normal form and every field is either part of the key, or provides a fact about the entire key and nothing else. In other words, non-key fields that are facts about other non-key fields in a row must be removed through decomposition.

Revisiting our order database, note that we have created a new field name description. The description field is a fact about the product ID, and is not dependent on the primary key nor does it describe the primary key. Additionally, the description column provides redundant information. For example, product ID 23 refers to a cookie. Imagine what would happen if you need to change the description of product ID from cookie to cracker.

In order to do this, you need to change every single record in this table that refers to product ID 23. Imagine what would happen if you missed one, there would be an inconsistency in your data. Additionally, what if none of the customers ever purchased item number three on order 254? Would product ID 15 and product description ‘Cola’ still exist in the database? In fact, it would not, so the next time a customer purchases a cola, the clerk would need to enter the ProductID and description back into the database instead of referencing it somewhere else in the database. In order to reach the third normal form, we decompose this top table.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **OrderID** | **Item#** | **Quantity** | **ProductID** | **Description** |
| 254 | 1 | 1 | 3 | Milk |
| 254 | 2 | 2 | 23 | Cookie |
| 254 | 3 | 1 | 15 | Cola |
| 125 | 1 | 4 | 23 | Cookie |

|  |  |  |
| --- | --- | --- |
| **OrderID** | **Date** | **CustomerID** |
| 254 | 10/24/2011 | 45 |
| 125 | 10/22/2011 | 13 |

#### Third Normal Form Example

This is the third normal form. We decomposed the top table by removing the description field and adding it to a new table along with its primary key of ProductID. Now, for each table, every field is either part of the primary key, or provides a fact about the entire key. We have now achieved the third normal form for this database.

|  |  |  |  |
| --- | --- | --- | --- |
| **OrderID** | **Item#** | **Quantity** | **ProductID** |
| 254 | 1 | 1 | 3 |
| 254 | 2 | 2 | 23 |
| 254 | 3 | 1 | 15 |
| 125 | 1 | 4 | 23 |

|  |  |  |
| --- | --- | --- |
| **OrderID** | **Date** | **CustomerID** |
| 254 | 10/24/2011 | 45 |
| 125 | 10/22/2011 | 13 |

|  |  |
| --- | --- |
| **ProductID** | **Description** |
| 3 | Milk |
| 23 | Cookie |
| 15 | Cola |

## SUMMARY

In this lesson you had the opportunity to learn about basic database design and construction. You learned that database schemas provide the structure which governs how a database functions. An eight step process was reviewed in this lesson to provide you with the information needed to know how to build a basic database.

## ASSIGNMENTS

1. Lab 2: Basic Geodatabase Design
2. Quiz 2: Basic Database Design