Table of Contents

In the first unit, we had a brief intro to some database terms. In this unit and the next few units we will explain these types of terms in a bit more depth. This document will seem rather dry, but go though it and come back next week and read it again. After a reading or two (or three) you should be able to understand these terms in context.

1. Terms

Application
Attribute
Base table
Client
Column
Data independence
Data integrity
Data type
Database
DBMS
Foreign key
Normalization
Primary key
Relation
Relational database
Relationship
Row
Schema
Virtual table

2. Relational databases

A **database** is a collection of related data organized for some purpose. Some people will use the term database for any type of storage including paper files. Most people will assume that the database is a computer system managed by a collection of programs called a database management system (dbms).

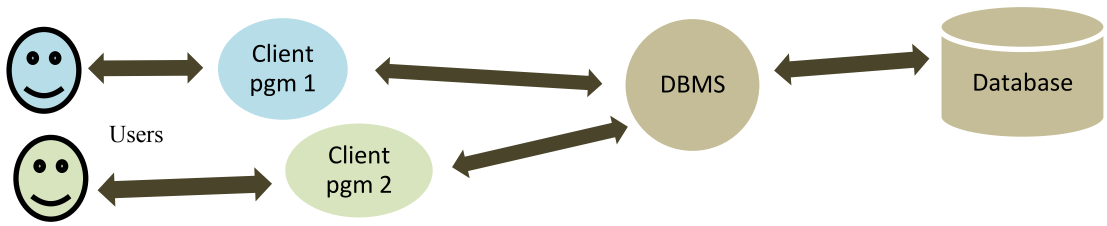
A **relational database** stores data in structures called **relations** which can be represented by two-dimensional tables. A relation is a mathematical concept- but we can think of it as a table. We will discuss relations in a bit more detail later in this document.

3. Client, application

These are terms that can have several different definitions. The way I will use them is that the client is a purpose that lets us work with the database. The client lets us enter SQL statements and see the results. The client lets us build database objects such as tables. We can also run scripts via the client. But as you will see over the semester, coding SQL statements is not the appropriate way for most end-users to get results from a database. We do not expect students enrolling in classes to do this by coding Insert statements. For an end-user we want to provide a more specialized interface- probably using menus and text boxes and buttons. The way an end-user interacts with a database is generally through a specialized application that provides an interface appropriate to the end-user. We can have an application that handles student enrolling in classes and a different application that allows staff to build the semester schedule of classes. The application program might use SQL statements to work with the database. In this class, we do not build the application level software. We work at the SQL level.

4. DBMS features

The **DBMS** is a collection of programs that manage the database. When we issue an SQL select statement, we enter it in a client program which sends the SQL statement to a translator program which changes our text into a query that can be processed. Then another part of the dbms takes that request and gets the data from the physical data in storage. Now the DBMS send the data back to the **client** program which may need to do some additional steps to format it for our display.



4.1. Physical data independence

One of the advantages of using a dbms is that the end-user and, to a large extent, the developer does not need to know how the data is stored physically on the persistent storage devices - often hard drives. The storage details are more in the realm of the database administrator (dba). For example, the data stored on the disk might be organized in a particular physical order to make retrieving data more efficient. The SQL to retrieve the data does not logically depend on that ordering. If the physical order of the data in persistent storage is changed, the SQL application does not need to be changed. If a developer wants to store a date value, she should not be concerned about how the data value is stored in terms of the bit patterns. In reality, a developer often has to know something about the storage of data to write the most efficient queries and to decide on the proper data types to use.

4.2. Logical data independence

With logical data independence we are talking about the ability of the dbms to access data even if the organization of the underlying data is altered. For example, suppose you define a table that stores the student's street address and their zip code as two attributes. It should not make any difference to your code whether the data is stored with the address first in the persistent storage or the zip code first. In a relational database the order of the attributes in the table can have no logical significance. Suppose I decide to add a new column to the zoo_2016 table and allow that column to be null; your SQL code for A01 should still work.

4.3. Data integrity

The dbms is responsible for protecting the integrity of the data. This has several aspects. If the developer defines an enrollment date as a date value, the dbms should reject an attempt to store a value such as Feb 31, 2009 since that is not a valid date. If the developer defines an attribute as an integer number between 1 and 15, then the dbms should reject an attempt to store a value such as 51 since it is out of range.

Systems vary in the proper response in some situations. Suppose the developer defines a last name attribute as a string field with a maximum of 15 characters. What should the dbms do with an attempt to enter a value that is longer than 15 characters? One approach is to reject that value and another approach is to store only the first 15 characters of that value.

4.4. Query optimization

It might be that our query is rewritten in some way by the translator program to make it more efficient. This is one of the jobs of the dbms (optimization) and the dbms should not change the logic of your request. The dbms generates a plan of execution as to how it will get the data from the tables. For our first sets of queries this will be a pretty simple plan- but as we work with queries that get data from several different tables and as our tables get bigger, this execution plan will become more important. We do not focus on query efficiency in this class as our job to create a query to return a correct result set; but in a database with tables with thousands of rows of data, efficiency is much more important.

4.5. Concurrency control

In our class assignments you will be the only user accessing your tables; in more realistic situations a dbms will have to handle multiple users working with the data at the same time. For example, we might have two students trying to enroll in the last open seat for a class. The dbms handles this but the way that a developer writes code can affect this situation. This feature, concurrency control, is only briefly discussed in this class. We discuss this topic more in the database programming classes.

4.6. Database security

Database security- in terms of user access, is handled by authentication and authorization. Authentication means identifying a user- generally via a login and password. Authorization deals with a set of privileges that the user has been granted.

When we log into the database, the dbms checks our user name and password to see if we have the right to login. Then it checks what types of work we are allowed to do- can we create tables? Or just read tables that already exist? Can we change the data in the tables?

4.7. Backup and recovery

This is another topic that is more dba oriented. Our tables for class are not storing mission critical data (except for your turning in assignments on time). We have a set of SQL statements to build the database and load the tables with preset data. So you can rebuild the database if needed. Hopefully you will save your scripts for the queries you create.

But for an online real time system, backup and recovery is critical; CCSF would be in trouble if the enrollment system failed and we had to ask students to enroll in their classes again. You would not be happy if your bank's database failed and you had no way to determine the status of your bank accounts. A major dbms will include features to do backups of data while the system is running and help with recovery of the data. For a mission critical system this has to be a lossless recovery- not a backup from the last day of the previous month.

5. Business entities and database components

When we use a database in a company there is always some purpose for which the database was constructed. A **business entity** is something for which we want to keep data. Examples of entities are: students, customers, books in a library.

One of the early steps in designing a database is to decide on

- the entities that are important to our company
- the attributes we need to save for these entities
- what type of data the attributes store
- how the data values for one entity are related to other entities
- what rules we have about the data

The database that we are using for the next few weeks deals with a veterinary clinic and the exams the clinic does for animals. The entities we will consider are:

- the animals the clinic sees
- the clients who are responsible for those animals
- the clinic staff
- the exams the vets perform

These entities will be handled in the database by creating tables.

For an entity, the pieces of data we stored can be called **attributes**. Examples of attributes for an animal seen at the clinic include:

- the animal's name
- the type of animal
- the date the animal was born
- the client who is responsible for that animal
- the date of any exam
- the details of the exams for that animal

We will need to make some decisions about the attributes to be sorted. For this database I am simplifying some of these decisions. For example, I think we can agree that each animal has one value for its animal type- the animal might be a dog or a bird. For the clinic we can say that each animal has exactly one value for its animal type and it must have that value.

But we might not insist that the animal has a name. Maybe the vet is treating a new litter of puppies and the puppies do not yet have a name. But we might decide that we will only store one name for each animal.

We will expect that most animals will have multiple exams- so we will need a way to store multiple sets of exam data for each animal.

Suppose this is a representation of part of the table for animals.

an_id	an_type	an_name	an_dob	cl_id
10002	cat	Gutsy	2010/04/15	3560
11015	snake	Kenny	2012/02/23	4534
11025	bird		2012/02/01	4534
12035	bird	Mr Peanut	1995/02/28	3560
21004	snake	Gutsy	2011/05/12	5699

The attributes are stored in **columns** and each **row** in the table includes the attribute values for a specific entity (an **entity instance**). A column in a table has a name and a **data type**. For the animal table example, we could have the following
an_id(integer), an_type(character), an_name (character), an_dob(date) , cl_id (integer).

Each row in the table should say one true thing about an animal. The first row says "We have an animal named Gutsy which is a cat. The animal was born April 15, 2010 and is associated with the client with client id 3560. We are using the value 10002 as an id for this animal."

In this section of the table we have 5 rows- so we are saying 5 true things about animals. Notice that the following represents the same 5 true things.

an_id	cl_id	an_name	an_dob	an_type
11025	4534		2012/02/01	bird
12035	3560	Mr Peanut	1995/02/28	bird
10002	3560	Gutsy	2010/04/15	cat
21004	5699	Gutsy	2011/05/12	snake
11015	4534	Kenny	2012/02/23	snake

We can also define rules for the data, called **constraints** which limit the data values that can be entered. We might want a rule that the an_type values are limited to a certain set of values. We might want a rule that the values for an_id has to be a positive number.

The collection of all of the rows in the current table is called the **entity set.** An individual row can be considered an **entity instance**. We need to be able to distinguish the individual rows in our table, so each table should have a **primary key-** an attribute or collection of attributes that uniquely identifies that entity instance. No two rows in the same table can have the same value for the primary key and no row can be missing a primary key value. Here we will define the primary key for this table as the an_id attribute.

Traditional **normalization** rules say that each attribute should store a single value; this is often interpreted as storing a single simple scalar value. Normalization rules say that all attributes in a row should be determined by the primary key of that entity instance. We will come back to the concept of normalization later in the semester.

Part of database design is deciding how many tables we have and what data is stored in each table. When we store data in tables, we need to organize it efficiently. Often this means that we split the data into several tables and then build **relationships** between the tables.

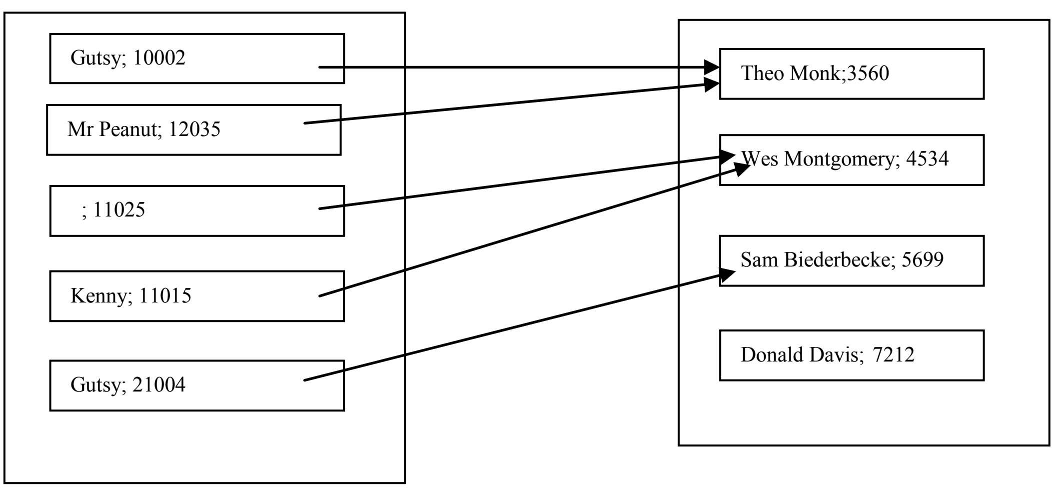
The term **base table** refers to a table that we define with the Create Table statement. Base tables hold data values. The term **virtual table** refers an in-memory data structure that appears to hold the data.

6. Relational databases and relations and relationships

The relational database model has been used for more than 40 years (a very long time for a computer model). The relational database model was developed by Dr Codd in the late 1960's and is the underlying model for the majority of database systems in use.

In the relational model, we think of the data as being stored in **tables** (relations) - with data being stored as a series of rows and columns. A relation stores a set of values that are related to the same entity. The data in one table will usually be associated with data in other tables in the database. The association between two tables is called a **relationship**. Relationships are implemented by the two related tables having data values in common.

In our vets database we have a table with client information. Each client gets a cl_id value which is the primary key for that table. We also store the client's name and address data in the clients table. The value for the cl_id attribute in the animals table has to match a value for cl_id in the clients table. In the animals table, the cl_id attribute is called a foreign key. Then we can navigate from a particular animal in the animal table to the related client in the client table. We can also navigate from a particular client in the client table to the animal in the animal table for that client



When we talk about a database as a logical thing, we think of a collection of tables that are logically related to each other. We may include other types of objects with the database- such as views and processing routines.

In summary: data values are stored in columns. Columns are grouped together into a row that represents an entity instance. Rows are grouped together into a table which represents a business entity that we care about. Tables are grouped together into a schema- a collection of tables. Schemas can be grouped together into a database.