Overview of Database Systems

by Terry Sergeant

Contents

1 Introduction

1.1 Definitions

At the most basic level a **database** is a collection of data. A database typically has a well-defined structure that accompanies and organizes the data to make it accessible. "A **database management system (DBMS)** is a collection of interrelated data and a set of programs to access those data." (Silberschatz, et al)

1.2 Rationale

Databases are widely use to organize data in virtually every sector of business and research. Examples: banking, airlines, universities, credit cards, etc.

Imagine being a programmer for a bank that wants to keep track of customer contact information and account information. If you were a programmer implementing programs for the bank you would have to not only worry about the user interface, but how to represent information and store it in files. Consider some of the issues you would have to address:

- **Selecting file type and organization** is a significant undertaking. Any time there needs to be a change in file structure all programs that use the file will need to be rewritten. In addition, writing robust routines to handle reading from files in the case of missing or incomplete information is difficult.

- **Data redundancy** can occur if you store a contact phone number in two separate locations. Redundancy opens the door for inconsistency and update issues.

- **Changing user needs** would require involving a programmer to create or modify a program to produce the desire list.

- **Enforcing rules for valid data values** would be needed in order to prevent bogus entries. For example, an account number would need to follow a certain format and the programmer would have to perform significant error checking to validate user entries.

- **Handling concurrent access** is a surprisingly difficult problem to solve correctly. Many applications, including banking, would need to allow multiple tellers to access the same account concurrently. **Give an example of what can happen if two tellers are modifying the same account concurrently.**

- **Making transactions atomic** is needed to ensure a consistent state. **What does "atomic" mean in this context. Consider a transaction to transfer funds . . .**

- **Maintaining security** in a large system with many levels of users is an important and necessary task. For example, you don't want every user to be able to view or modify every piece of information in a bank's database. **Can you think of examples of this in university databases? HR databases?**

Many modern DBMSs provide built-in mechanisms to handle these issues for the programmer so that their focus can shift from implementing solutions to these difficult problems to making the user-interface accessible.

2 Topics in Database Systems

2.1 Database Design and Use

We will learn to use three DBMSs this semester:

MS Access is a standalone (one-tier) DBMS designed to be simple to use and optimized for small databases. It provides a rich set of tools for implementing user-interfaces.

PostgreSQL is client-server (two-tier) DBMS that provides advanced features and is designed to handle large databases.

MongoDB is a widely deployed non-relational database management system that is often used as a backend to web applications.

There are several models used by various DBMSs to organize data. Some common models are:

Relational Model This is (currently) the most commonly used model and will be the focus of this course. Databases based on this model organize information into two-dimensional tables (with various restrictions). The process of *normalizing* a relational databases during the design phase is important. MS Access and PostgreSQL are relational DBMSs.

NoSQL This is a loosely defined category of models that have risen from databases developed for high-traffic web sites. Databases based on a NoSQL model do not provide the same level of consistency guaranteed by the relational model, but in exchange provide better performance.

XML This method of organizing data is more relaxed than the relational model and is primarily used as a backend data exchange format by (typically) legacy applications. Data is stored as plain text.

2.2 Database Programming

Relational DBMSs almost universally allow programming of the database in the form of a non-procedural language called **Structured Query Language (SQL)**. In addition, many DBMSs allow access via general purpose programming languages (e.g., C++, Java, Perl, Python, VB, etc.) by way of standard interfaces such as ODBC or JDBC.

2.2.1 SQL

SQL is implemented (to some degree) in virtually every modern relational DBMS. There are some variations among different DBMSs, but in general, if you learn SQL you can get around in any DBMS. SQL statements are categorized as follows:

Data Manipulation Language (DML) These are commands that allow you to view contents of the database (i.e., perform queries) and to insert, remove, and modify database entries.

Data Definition Language (DDL) This is the part of SQL that allows you to create or delete tables and modify the structure of the database.

Data Control Language (DCL) These commands are used by a DBA to deal with database users and to assign access permissions, etc.

2.2.2 General-Purpose Programming

SQL is good for manipulating the database itself. Providing user-interfaces (such as through a web page) is typically done by using other tools or languages. Thus, there is a need for procedural, high-level programming languages to be able to converse with a DBMS. Two widely used standards for bridging that gap are ODBC (Open Database Connectivity) or JDBC (Java Database Connectivity).

2.3 DBMS Implementation

There are three levels of data abstraction relavent to the study of database sytems:

- The *view level* is the level at which typical database end-users interact with the database. They are concerned with the lists and other information produced by the database and are not concerned with underlying structure. There can be many views depending on user needs.
- The *logical level* deals with the way in which tables have been created and relationships established. DBAs and applications programmers view the database at this level. The organization of the database which has been defined at this level is sometimes called the *schema*.
- The *physical level* has to do with the way in which files are arranged and stored on the underlying secondary storage device so as to allow efficient retrieval. Typically, only a DBMS programmer is concerned with this level.

Storage management (physical level) is just one concern with regards to the implementation of a DBMS. Other issues include details related to how queries are processed, query optimization, and transaction management. **What do we mean by "transaction" and why does it need to be managed?**

2.4 Database Administration

A **database administrator (DBA)** is a person who is charged with managing a database. This can include the design and programming of the database. In addition a DBA would be responsible managing users, establishing security settings, making backups, and meeting the needs of the database users. Virtually every topic discussed this semester will be relavent to being a database administrator.

3 A Quick Example

The example database given here is used to introduce some terminology and to help "grease the skids" for some the more thorough treatment of these topics that will follow.

Suppose you want to store information about your favorite music albums in a database. Suppose that you will store the following information for each album: album title, album year, genre, artist name, artist hometown. We'll discuss methods for database design later, but for now suppose you intend to use three tables to represent this information: `Artist`, `Genre`, and `Album`.

3.1 An ER Diagram

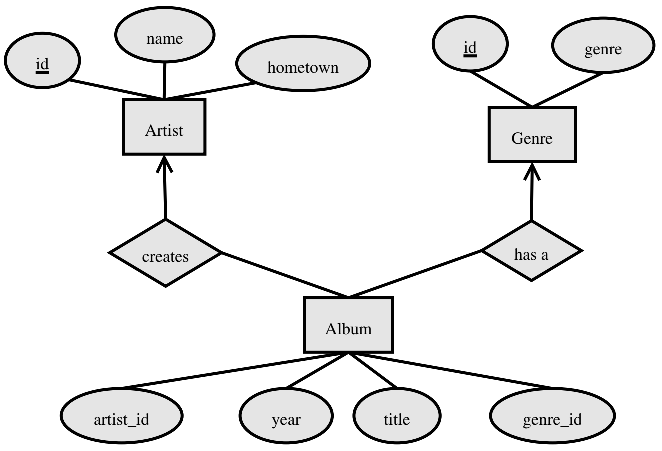
One way to represent our proposed database is using an Entity-Relationship (ER) diagram. The conventions for this type of diagram are:

rectangles represent **entities** (i.e, tables)

ovals represent **attributes** (i.e, fields); an underlined name signifies a primary key (more on keys in a minute)

diamonds represent **relationships**

lines show connections or membership; an arrow represents the cardinality of a relationship (one-to-one, one-to-many, etc.)



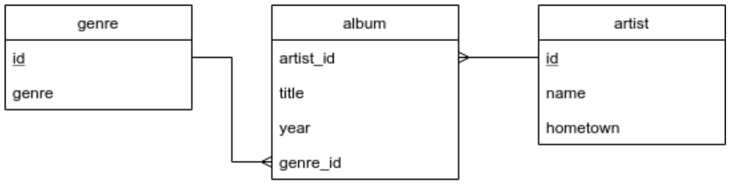
Spend some time examining this.

The ER diagram simply gives a pictoral view of the proposed (or actual) design of a database. The database design/structure is sometimes called the database **schema**. The schema refers to the design of the database, *not* to the contents of the database.

3.2 A UML Diagram

UML is used in many disciplines including database design. The UML conventions for representing databases are:

- Each table is represented as a rectangle with the name of the table at the top.
- The fields of each table are listed in the rectangle below the name.
- Relationships between tables are designated with lines connected a field in one table to a field in another. The ends of the line are plain if it represents a "one" and three-pronged if it represents "many".
- Primary key fields are underlined.



Spend some time examining this.

UML diagrams are another way to provide a pictoral view of a database design.

3.3 A Tabular View

Suppose that we have inserted the following records into the tables in our database:

Artist

id	name	hometown
1	Dog Man	Coyote, Wyoming
2	The Computing Whiners	Seattle, Washington
3	Soul Screech	Boston, Alabama

Genre

id	genre
1	Rock
2	Pop
3	Jazz

Album

artist_id	title	year	genre_id
1	Howling Knights	2003	3
1	Woof, Woof to You Too!	2004	2
1	Puppy Love	2005	3
2	Are You My Motherboard?	2003	1
2	CPU Burnin'	2005	1
3	Stop the Pounding in My Brain	2003	1

Spend some time looking at the arrangement of these tables. Can you make sense of the `Album` table? Spend a moment to compare the tabular representation of the database with the ER diagram.

A **key** is a field (or collection of fields) that is sufficient to uniquely identify a record. In the ER diagram the `id` field of the `Artist` table and the `id` field of the `Genre` table we designated to be key fields. **What field(s) might be used as a key in the Album table?**

3.4 Some Things We Might Do

The arrangement of data into three tables is helpful in that the tables conform to the definition of a **relation** (which in turn allows us to apply a variety of mathematically defined operations in order to extract information). This arrangement is, by itself, not that helpful for provide usable information to a typical user. One of the primary reasons for using a database is for the purpose of allowing users to extract information easily. This is typically done by creating **views** (or **queries**). Here are some examples of information we might want from our database:

- List all albums along with the artist name and genre name (and order the list by album year, then by album title).
- List all albums in 2003 (or prior to 2003).
- Count how many "Rock" albums there are.
- List all artists from Coyote, Wyoming.

We will learn how to construct databases and queries in more than one DBMS.