Databases vs. Flat Files

by Sophia

WHAT'S COVERED

In this lesson, you will explore the differences in flat files and databases regarding flexibility, data consistency, security, and data redundancy, in three parts. Specifically, this lesson will cover:

1. Databases

Databases have evolved from flat files to relational databases over time to meet the needs of more complex applications.

Flat files are simple databases that store data in a single table. A more recent example of a flat file is a spreadsheet. Flat files that stored data in plain text in a table format were originally developed by IBM in the 1970s. There were no indexes, keys, or relations between the information between rows, but each row was its own dataset. For example, in a flat file that stored customer orders, each order would be in a separate row. The fields (such as Name, Address, City, and so on) would be separated by a **delimiter** character, a sequence of one or more characters for specifying the boundary between separate, independent regions in plain text, mathematical expressions, or other data streams, such as a comma or a semicolon. As the name implies, a delimiter character defines the boundary of a field.

A flat file can be efficient for small datasets, but it becomes difficult to manage and query as the data grows.

DID YOU KNOW

Flat files also do not support relationships between different pieces of data, which can limit their usefulness for complex applications.

A flat file database can potentially contain a lot of duplicated information, and back in the 1970s, data storage space was limited. All that duplicated information took up valuable processing time and storage space, so a new way needed to be developed that would not just be more robust for searching but also economize data storage space.

Spreadsheet applications such as Visicalc (1979) and Lotus 1-2-3 (1983) were developed to more easily manage data in structured rows and columns, and such applications are still very popular today, such as Microsoft Excel. Although spreadsheets make the data much easier to view and query, they do not solve the fundamental problems of redundancy and connectivity. A better way to manage complex, related datasets was needed.

BIG IDEA

Relational databases are a more advanced type of database that stores data in multiple tables. Each table has a unique set of fields (columns), and the tables are linked together by common values in certain columns. This allows for more complex queries and relationships between data, making relational databases more suitable for large and complex applications.

The evolution from flat files to relational databases was driven by the need for more efficient and scalable data storage and retrieval. A British computer scientist named Edgar Codd wrote a paper in 1970 that introduced the fundamental concepts of what we now know today as the relational database model, and shortly after that, IBM developed the SQL language as a means of interacting with such a database. Oracle was one of the most influential early commercial relational database products. Introduced in 1979, it helped popularize the relational database model. The relational database solved processing and storage issues, allowing applications to become more complex. This solved many of the limitations of flat files that became more apparent over time and as data use and needs became more complex. Relational databases offered a solution to these limitations, and they have become the standard for database management in most organizations.

Flat files are still popular today, mostly via spreadsheet apps such as Microsoft Excel and Google Sheets. Often you will see flat and relational databases used in conjunction with each other, as each has its own strengths and weaknesses depending on the business problem that needs to be solved. Since 1970, both types of databases have grown, become common, and are widely used today, but there are core benefits that relational databases have over flat file databases.

KEY CONCEPT

Relational databases are more:

- Efficient at storing and retrieving data than flat files because they use a more compact data representation, and they can take advantage of indexes to speed up queries.
- Scalable than flat files because they can be easily partitioned to store large amounts of data on multiple servers.
- Flexible than flat files because they can support complex relationships between data, which makes them more suitable for a wider range of applications.
- Efficient at storing data than flat file databases; relations between tables allow for data to be stored once but used in any number of complex relationships with other data.

Relational databases have become the standard for database management in most organizations. They are used by a wide variety of applications, including enterprise resource planning (ERP) systems, customer relationship management (CRM) systems, and data warehouses.

Flat file databases are still used and have certain advantages over relational databases in some situations. For example, NoSQL flat file databases can store data in a simple, unstructured, or semi-structured format, typically

in the form of key-value pairs. **MongoDB** is a popular example of a NoSQL flat file database that stores data in an object-oriented way.

Over time, both relational and flat file databases have shown that they can be used in conjunction with each other; it is really just a matter of what is the most efficient way of storing and retrieving data. A NoSQL database is a good choice where rapid read and write operations are important or when storing unstructured or semi-structured data. However, relational databases remain the most popular type of database for general purpose use.

TERMS TO KNOW

Relational Database
A database that has multiple tables, with shared common columns between tables.

Flat File
A data file that is not related to or does not contain any linkages to another file.

Spreadsheet
A row-and-column grid in which structured data tables can be created.

Delimiter
A sequence of one or more characters for specifying the boundary between separate, independent regions in plain text, mathematical expressions, or other data streams.

MongoDB
A popular example of a NoSQL flat file database that stores data in an object-oriented way.

2. Flat File History

For thousands of years, people kept track of data on clay tablets, papyrus, vellum, and then finally, paper. Keeping track of information by identity and then a checkmark is nearly as old as civilization itself. Physical file systems are where the idea of databases began; as computer systems were being designed in the 1950s and 1960s, people began trying to work out the best way to store and process data.

A mathematician named John Backus, working for IBM in 1959, proposed the creation of a specialized database and programming language that would help people find data better. Before this, the “ledger,” or book, was the way to do accounting, taxes, census, and other information. All of this was collected and handled with paper and pencil.

From a computer viewpoint, the ledger style of recordkeeping was very hard to process and took up more hard drive space than was available. These types of paper systems were organized through a system of file folders and filing cabinets. Typically, there wasn’t a significant amount of data collection required, and reporting was quite limited. This type of system worked well as a data repository. However, as companies became larger and needed to report on their data, managing the data in these physical file systems became too difficult and too costly.

Generating reports from physical file systems was difficult and error prone.

REFLECT

Can you imagine being in an aircraft hangar-sized warehouse like the picture above?
Because people were used to the physical processing of data, files in early computerized file systems were stored similarly to physical files. There were “boxes” of “cards,” with each one of those “boxes” being a flat file and each “card” being a row in the flat file database, much like in spreadsheets today. If a business user needed information, they turned to data retrieval specialists who handled all of the data processing requests for the company. These early computerized flat file systems became just as much of a burden to manage due to related and overlapping data, with no means of controlling or managing the data across each of the files.

The shortcomings of the physical recordkeeping and the difficulty in duplicating it on early computerized file systems were the impetus for developing the first computerized database systems. Those first database systems took time to develop. It was not until the 1970s when more advanced programs were created that allowed the use of the new math. Even then it was still difficult to create databases. The early databases, both relational and flat, had challenges in returning quick answers to questions and providing answers that humans could make sense of. Those old systems also had very complex systems administration and lacked security.

3. Weaknesses of Flat Files

You are just learning how to use a database, so let's set up a link for you to meet your first database in this class.

TRY IT

Click on this link to launch your first database:

You'll learn a lot of commands in this class, but for right now, you can just follow along and learn to move around a database. You will learn a lot more about these commands as you progress through the class.

Let's look at two tables, the invoice table and the customer table. To get all the information in each table, enter this one table at a time. Type the text shown below and then click the Run button (the right pointing triangle) in the upper right corner of the screen.

```
Select * from invoice;
```

This will show you all the records in the invoice table. Then repeat the process using the following command:

```
Select * from customer;
```

This will show you all the records in the customer table.

Notice what these two tables have in common: They both have a customer_id field. In this database, invoices are stored in the invoice table and related to the customer through the customer_id. If you had to use a flat file database, you could only have a single table in which to store both the customer and invoice data. All of the information would be stored in a single record. The customer's information, like the name, address, phone, and email, would be stored in the same row as the invoice information. If the customer makes more than one purchase, all of that customer information would have to be reentered for each item they order.

REFLECT

Even today, that would waste storage space and cost in processing power trying to process a huge, ever-growing file. Could you imagine Amazon or eBay working from one table only for all their customers and third-party sellers? Eventually, the table would become too big to process, so having a shared column between two tables makes processing and storage much easier and quicker to work with.

Imagine if you had to enter all the address and user information every time you made an online purchase from a company. So, rather than storing your address, or your family's address, you had to enter that every time. If you ran a query to pull up a list of your orders, you would get a lot of records back if you ordered from that company a lot. That might make it very hard to find out what your current order is and if it has been processed. The whole order process would become more difficult each time you placed an order. What other problems can you see as the invoice table keeps on getting bigger and bigger, or the store becomes very popular?

Keeping customer data in flat files creates a lot of data redundancy, as a company would have the same data stored multiple times, not only in the same flat file but perhaps in other flat files for other purposes.

If you have data redundancy due to repeated data, then you have poor data security. Having multiple copies of the same data increases the chance of unauthorized access to the data. The operating system would have to handle the security for flat files rather than the database. This could open the database or the operating system to misuse if someone gets the wrong permissions for the system.

Potential data inconsistency would also exist, like the address scenario above. You could have different and conflicting versions of data appearing in various locations in your records. Data inconsistency can further increase if you have data entry issues that go undetected.

Finally, since you would have repeated information, the data file would become much larger and more difficult to manage and maintain. You would also lack the flexibility to manage the data in a flat file, as you could only enter data that fits the data structure. In the example, you would only have one customer and one invoice inserted together in the same flat file table. If you later wanted to record other information about customer transactions, you might be unable to change the available columns to include those options.

SUMMARY

In this lesson, you learned that the overall style and structure of **relational databases** have evolved from physical file systems and computerized **flat file systems**. Compared to those earlier methods of storing data, relational databases offer improved flexibility, data consistency, security, and reduced data redundancy. Finally, you learned about key **weaknesses of flat files**. Next time, you will learn more about various parts of a database system.

Source: THIS TUTORIAL WAS AUTHORED BY DR. VINCENT TRAN, PHD (2020) AND FAITHE WEMPEN (2024) FOR SOPHIA LEARNING. PLEASE SEE OUR TERMS OF USE.

TERMS TO KNOW

Delimiter
A sequence of one or more characters for specifying the boundary between separate, independent regions in plain text, mathematical expressions, or other data streams.

Flat File
A data file that is not related to or does not contain any linkages to another file.

MongoDB
A popular example of a NoSQL flat file database that stores data in an object-oriented way.

Relational Database
A database that has many tables, with shared common columns between tables.

Spreadsheet

A row-and-column grid in which structured data tables can be created.