Understanding Databases

When I work on a jigsaw puzzle, it's to enjoy working on it with my family. It generally takes days over the Christmas holiday to solve one puzzle each year.

When I study something, it's generally because I want to, need to, or get paid to. My assumption is that you got this book because you need to understand how databases work; and this is where you're starting or restarting your learning process.

This chapter explains what a database is, why it's important, and how it works.

What's a database?

A database is an organized set of data. You create and maintain a database by using a software application. The simplest database is a two-dimensional table, like a worksheet.

Worksheets have columns and rows. Each column describes a piece of data. The collection of columns defines the subject of the data, and the rows contain instances of the subject.

Modern spreadsheet programs support collections of worksheets in a single file. They call that single file a workbook. Likewise, databases typically consist of more than one table because we seldom manage data in isolation.

You create workbooks, or collections of worksheets, with application software. There are three major vendors in that application software market; and, they are Microsoft Excel, Apple Numbers, and Open Office Calc. You can create powerful workbook solutions with any of these spreadsheet products.

After you create a workbook solution, you maintain the various spreadsheets by using the application software. The application software lets you add, modify, or delete data from any part of the workbook. Data inside spreadsheets can be simple – numbers and strings, or data can be complex, like formulas. You also can link worksheets to external database sources, which makes the simple data dynamic. At the more advanced level, you can create solutions that link workbook files together. This type of solution combines multiple workbook files into an analytical appliance. This type of multi-document workbook acts like a database.

Application software is a set of programs that let you create and maintain workbooks or databases. We call the application software of a database a database management system (DBMS).

DBMS is the most generic acronym for databases. DBMS can describe application software that manages files, relational tables, or name-value pairs. A DBMS that works with files typically works with flat files. Flat files are fields of descriptive information. The fields in flat files are delimited by commas or tabs, or organized by character position. DBMS also can describe hierarchical, networked, relational, object-oriented, or JSON structures databases.

- Hierarchical structures act like inverted trees, where you have a root node that descends through nodes to leaf nodes. Leaf nodes are nodes without dependent nodes.
- Networked structures are like inverted trees but they also have pointers at each ordinary node. Pointers act like traffic cops, and they point requests for matching data to data that may exist on a different branch of the same inverted tree.
- Relational structures create relations between tables. They do this by designating some set of columns in a table as a unique key. The unique keys let you connect rows from one table to rows from another table.
- Object-oriented structures group data into structures with *accessor* and *mutator* methods. An accessor method gets data from an object and is more commonly known as a getter. A mutator method assigns values to an attribute (or field) in an object, and is more commonly known as a setter.
- JSON is an acronym that stands for JavaScript Object Notation. JSON structures are name and value pairs. The name can point to simple data, like a number or string; or a name can point to a complex data, like a collection or embedded object.

Oracle, MySQL, DB2, PostgreSQL, and SQL Server are examples of relational database management systems (RDBMS) software. While Oracle and PostgreSQL act as relational databases, they also support objects. Because of their added behavior, Oracle and PostgreSQL are also object relational database management systems (ORDBMS).

While most databases can support JSON objects in large text columns, there are specific databases designed to manage name-value pairs. Cassandra and MongoDB are designed for high concurrency and name-value pair management. They also support direct access to JSON objects.

Most relational database management systems provide a common query language to work with data. The common query language is known as an application-programming interface (API). In RDBMS systems, it is SQL (Structured Query Language), which is a shortened acronym for SEQUEL (Structured English Query Language). SEQUEL was selected by IBM but

dropped because it infringed on a trademark. Name-value pair databases don't use SQL, and are NoSQL databases because they have a different API.

Other query languages exist for non-relational database systems. Object-oriented database management systems (OODBMS), like Versant, implement object-oriented programming languages as their primary interface. XML databases, like the MarkLogic database, implement XQuery as their query language.

Regardless of the database type, databases organize data in a structured format. The structured format requires software to manage its organization and both maintain and query the data.

Why is a database important?

Databases are important when they hold data that you need. The more data you need the more important the database. They supplement your ability to remember facts. Just as calculators let you perform accurate and quick math operations on numbers, databases let you find, add, and remove data.

A great example of a simple database is a contact application in a smart phone. It lets you add new, remove old, and find current contacts. Some contacts will have only their names and telephone numbers, and both street and email addresses.

As the data you need increases, your ability to recall it eventually diminishes. Recall is swamped by quantity. Databases record the information in an organized set of tables that contain facts. Application software knows how those facts are stored, how you can add new facts. Application software knows how those facts are stored, how you can add new facts, and how you can combine facts into information.

The cost of building such an engine is expensive. The cost is prohibitive for most software projects. Most application software incorporates a database management system (DBMS) for this reason, like MySQL, Oracle, SQL Server, or SQLite. They adopt the DBMS software's standard application programming interface (API) for organizing and accessing the data stored in the DBMS. This allows application software developers to focus on how the data is meaningful.

While the API is critical to the success or failure of application software, its maintainability and flexibility is conjoined to the successful organization of data in tables within

the database. The choices made in organizing data in a database directly govern how and what may happen in application software; a great database design provides the application software with flexibility and avoids the pitfalls of corrupting data.

Another reason for the existence of DBMS software is the need to access different data simultaneously by different users. We often discuss simultaneous user access as concurrent access because we rarely have interest in the same datum (or piece of data). If two people have interest in the same fact, then the first one that reaches it locks it until they're done using it. Any other party who wants access to the same fact must wait their turn to access it. This type of access method acts like a first-in and first-out queue.

DBMS software can support the concurrent interaction of many users. Facts would be lost or corrupted without that ability. For example, you may be able to focus on elements of two conversations at the same time, but you'll miss some details from each. Three concurrent conversations stretch your ability further, and so forth. DBMS software disallows the loss of concurrent details by managing them in separate channels. The channels are implemented by DBMS software differently, but more or less it's like having many concurrent processes or threads of operation (see the processes and threads sidebar for more on this).

DBMS software implements these concurrent operations to gain processing benefits unobtainable any other way. Databases achieve this through MultiVersion Concurrency Control, which is abbreviated as MCC or MVCC. MCC ensures that one user won't inadvertently destroy another user's change before it's permanent. MCC does this by guaranteeing all changes to data are ACID-compliant. ACID is an acronym for properties that guarantee the integrity of concurrent operations on data. The phrase *"concurrent operations on data"* is wordy. More often than not they're referred to as database transactions.

ACID stands for atomicity, consistency, isolation, and durability. Atomicity means everything or nothing will happen in a transaction. Consistent means that the data will change from one state to another. Isolation means that no other user can see your changes until they're made permanent. Durable means that you've written them to disk.

Virtually all businesses need concurrent access to data, and while a few may think it's unnecessary, they agree it would be helpful. After all, not everyone will wait in a long line (queue) for service. A queue may occur when two users want to change the same piece of data at

the same time. DBMS software locks the data when the first person attempts the change and holds that lock until they've completed their interaction with the database.

Processes and Threads

A process is a program running in a discrete or shared memory segment on a computer. You or a program start (or launch) a process from the operating system. Once launched, a process has complete control of its execution within the scope of privileges granted by the operating system. Processes read and write information to memory in their Process Control Block (PCB). When they complete their purpose, they relinquish their memory space to the operating system. Processes may return values to the operating system or write data to local or virtual disk.

A thread is a lightweight process. This is computer jargon or a fancy way of saying that a process starts a thread within its memory space. This makes a thread part of a process. Threads may read and write their own information to their Thread Control Block (TCB). They may also read or write to the owning PCB. Like processes, they relinquish their memory space at completion. Unlike processes, they release that memory back to the owning process not the operating system. Threads typically return values to the process management thread, which is also known as the thread of control.

Threads are most useful when processes want to divide and conquer the work they perform. Threads allow them to spread the load because they can run in isolation at the same time.

How does a database work?

A database works by using several components together. The components are: the client-server model, listener processes, optimistic and pessimistic communication paradigms, session management, locking, shared versus discrete memory, and repository management.

Client-Server Model

All databases provide an interface mechanism. That mechanism is generally called the client or client software. The database server that maintains the data is the server.

Many relational databases provide client interfaces that support interactive and batch modes of operation. Interactive mode is self-evident: the user types and see the result of the

typing and execution of commands on the user's console. Batch processing is a bit different. Users submit a job in batch processing and typically see a log file that documents its execution.

Client software can communicate directly with the database across the network or through operating system pipes. The communication can be through a proprietary access channel or industry standards like the Open Database Connectivity (ODBC) or Java Database Connectivity (JDBC). Major database products provide command-line interfaces that work in both interactive and batch modes. Client tools, like Quest's Toad, MySQL Workbench, Oracle SQL Developer, and others, provide graphic user interfaces (GUIs) that interact with the server like the command-line interface.

Client software generally requires you to install some software on your local computer. The client software provides the tools that let you speak to the server software across the network or through a local operating system pipe. While the ODBC model requires a set of dynamic link libraries (a `.dll` on Windows) or shared object files (a `.so` on Linux or Unix). The JDBC requires an implementation of a vendor provided Java ARchive file (JAR).

Aside from managing the data, server software implements an SQL statement-processing engine. It also controls a listener process. Listeners are programs running in the background. They listen for incoming requests on some operating system port or pipe. Requests are typically messages sent from the client software. Server software also manages the base files where the data is stored and log, redo, and archive log files.

Listener Processes

Listener processes run as part of the server software. They may run independently as separate processes or as spawned subprocesses.

Once started, listeners actively listen on a server port. A server port is a virtual or logical address through with one process can communicate with another process. The server program process that runs on the port address is the listener process. Client program processes send messages to the server at the port address and the listening process then creates a communication socket between the client and server software.

Sockets are created on ephemeral (or transitory) ports. The communication between the client and server software may be optimistic or pessimistic, which depends on how the request is made.

Optimistic and Pessimistic Communication Paradigms

There are two ways for the client software to communicate with the server software. One approach is optimistic and the other is pessimistic.

A pessimistic connection requires a permanent channel between the client and server software. While connected the client software can (a) instruct the server software to do many things, (b) confirm whether the server software has performed those things, and (c) prevents others from seeing the changes until the client software determines they're complete. You implement a pessimistic connection across a server side pipe or a network socket.

A server-side pipe is an Interprocess Communication (IPC) process. The server-side operating system manages the pipe and how messages are sent and received through the pipe. This type of communication requires that the client software be installed on the local machine along with the server software. This is the typical configuration for all DBMS software.

Network sockets are two-way channels across a network. The network can be as simple as the loopback on a server machine. Network sockets rely on TCP/IP (Transport Control Protocol/Internet Protocol) communication. Moreover, these connections send data across one channel and keep the connection alive through the other channel. The channel that keeps the connection alive guarantees the state between the client and server processes.

Optimistic connections are like an instant message. The client simply sends the message. Whether the message is received or not may require another message or two, and eventually a reply from the other party. The model would place the client software in the role of texting the first message and the server software in the role of responding to the text message. This interchange pattern can rely on the User Datagram Protocol (UDP) or a timeout value on a TCP/IP connection, and it is known as a stateless communication method.

You see this implementation when the client software communicates across the network using the JDBC or ODBC communication frameworks. However, both of these frameworks may support pessimistic communication when they implement a server-side module that maintains state between interactions (or, self awareness and control across a series of instructions). An example of this type of software is a JServlet.

Session Management

Session management occurs when the client software connects to the server through its listener process. The duration of optimistic connections may be short and

limited to a single SQL statement while pessimistic connections may be long and involve multiple SQL statements.

The database server software manages the number, resource access, and duration of sessions. Sessions are the working environment for users who connect to the database as authorized users in the database.

Users connecting interactively work as developers. Applications connect as users with security credentials or as anonymous users, and interact through an API.

Locking

Locking prevents another user from changing data before the first user completes their work with it. Locking is a major part of the process that guarantees ACID-compliance in a concurrent processing environment. A DBMS puts a lock on a row, set of rows, or table when one user transacts against a row. Dependent on how it is implemented, locking may prevent another user from querying a result that's in the midst of change, but it always prevents another user from changing data before the first user completes their work with it.

Whether locking impacts a row, set of rows, or table is an implementation detail of the DBMS. The more robust DBMS software locks the smallest amount of data necessary.

Locks are released when the original user completes their work and signals the database they've finished their changes. The signal is made with a `COMMIT` statement when they want to make their change(s) permanent, and a `ROLLBACK` statement when they want to undo their change(s).

Shared versus Discrete Memory

Memory in a computer is where information is stored to run programs. Random Access Memory (RAM) is limited and dedicated to running programs without reading and writing to disk. Whereas, memory is also disk storage. When programs use all available RAM, they cache segments of memory (known as pages) to disks, and this type of cached memory is virtual memory.

Starting programs on a computer launches a process. A process may be single threaded or multithreaded. A process requests memory when it starts. The request may

ask for a dedicated (discrete) or shared memory segment. The operating system kernel replies to the request and allocates memory to the program.

Shared memory describes the process of creating a memory segment that you want to share with other processes. Discrete memory describes the process of creating a memory segment that you want to use exclusively by a single program.

Some DBMS software uses a shared memory segment for all interactions with the database. In those cases, the DBMS allocates a portion of the memory to each database connection.

Using a single shared memory segment is a form of static marshaling. Implementers prefer static marshaling because it eliminates the dynamic marshaling cost. Dynamically marshaling allocates and deallocates memory for each database connection.

Static marshaling makes the footprint of the memory more consistent, contiguous, and manageable. There are various ways to describe memory management, and each vendor has their own vocabulary to describe the process.

Discrete memory is seldom implemented for concurrent DBMS software. The cost of dynamic marshaling is why it's avoided.

Repository Management

The server-side component of the DBMS consists of two principal parts. One is a data repository and the other is a set of programs that manage the repository. Most modern databases actually manage more than one repository (or database instance) with the same set of programs.

The data repository is synonymous with the term database or the more technical and accurate database instance. The set of programs make sure that all activities in the database comply with the MVCC rules and ACID-compliance of operations. These programs also ensure the integrity of memory segments, sessions, and resource locking. They write log files that let the database step-backward in time to undo operations and provide the ability to recover operations when something fails.

Databases created by DBMS systems often involve more than a single physical file. DBMS software manages the process of reading, writing, and controlling these files. As a rule, you never have the ability to write to the data files of a database instance.

Vendors implement various mechanics to support these data files. They also create various files that support different activities, like control files, redo log files, and archive log files. Control files keep inventory of all files involved in a single database instance. Redo log files keep track of changes before they're permanent and after they're committed or rolled back. They only contain very recent information. The DBMS may discard redo log files through normal processing or transfer their content to archive log files. Archive log files keep track of changes made over a long period of time and facilitate backup and recovery when a major error occurs.

Summary

This paper explained what a database is, why it's important, and how it works. It discussed the following key elements: the client-server model, the databases listener, the difference between optimistic and pessimistic connections, session management, locking, and the differences between shared and discrete memory, and concept of repository management.

