Relational Database System Architecture- A Discussion of RDBMS Components

CSC597 Database and Security - Summer 2022

Matthew Dunaway

HISTORY OF THE DATABASE

The origins of a database can be first seen in the collection and organization of literature in various forms in a library. The earliest known library was The Library of Ashurbanipal in Nineveh around the 7th century B.C. It housed a collection of 30,000 clay tablets organized by subject and included the “Epic of Gilgamesh”. (Andrews, 2016) Other libraries would be collected and organized in similar fashion, but this Assyrian library is the first recorded physical collection of related information that, in its own fashion, was indexed by related subjects, managed by an Assyrian DBMS/DBA (a librarian), and had the ability to be updated through plundering other civilizations and bringing the collections back to the library. This **physical database** is a textbook example of the definition, which is a device or construct housing the information and the method to access the information between object sources. ("Physical and Logical Databases", 2010)

DATABASE DESCRIPTION

A **database** is a collection of logically related information that can be accessed, managed, and updated. The logically related information, data, is any information related to an object of consideration. This data can be physical characteristics, scientific measurements, interactions, or images of the object. The basic **logical structure of the database** can be simplified on five building blocks. The **fields** of a database are the most specific attributes of an object. A **record** contains a collection of these fields of object attributes. A **table** contains the records organized by an identifying key and a unique table name. The **company** is a sub-collection of a database that can specify private tables and public tables or any other grouping. Finally, the **database** organizes the related tables with a schema describing the relationships. ("Database- Overview, Roles and Components, DBMS", 2022)

Diagram

Description automatically generated

Logical Structures of a Database ("Physical and Logical Databases", 2010)

FUNDAMENTAL COMPONENTS OF THE DATABASE

The purpose of the report is to discuss the main components that make up most of the database architecture designs in use today. We will list the main components of the database, as exemplified in open source and commercial designs, and how database form translates to common functionality in nearly all database management systems (DBMS). While there has been momentum in the architecture type of non-normalized database architectures such as a data warehouse, the product of data warehouses, data marts, is relational, object-oriented data. For this reason, the architecture discussed in this report will be relational database system architecture and the components common to them. The five components common to most DBMS’s are *client communication manager, process manager, relational query processor, transactional storage manager, and shared components and utilities*. These components will be outlined in a step-by-step user query format because most of the readers have a high-level understanding and experience using a web-based query on a smartphone or personal computer.

Graphical user interface, diagram

Description automatically generated

Five Components of a DBMS (Hellerstein, Stonebraker & Hamilton, 2007)

ARCHITECTURAL COMPONENTS OF A RDBMS

The first component discussed is the **client communication manager**. If a person wants to check their account balance in their personal bank account at a gas station before purchasing fuel, they will open their bank’s application to query the bank’s database. The bank application is different from a software or program in that it has a distinct function but communicates similarly. In this case, the customer’s smart phone is deemed to be the **client** in the upcoming communication relationship. The client application requests a connection with the banks API, or application programming interface. The API approves the communication request with encryption keys, usernames, and passwords and sends the query to the banks database manager. Here the query is specified by the database manager to the actual database in the form of SQL, COBOL, or other language specific to the communication request. The bank database **server** application returns the balance of the customer’s account to the database manager, which provides that information to the bank API. The bank API completes the communication to the bank application on the client’s phone and displays the account balance. (Karnes, 2021) This client-server communication manager is on the server side of this communication and maintains the connections to and from the database through the open database connectivity API or the Java database connectivity API. It is also responsible for responding to SQL commands from the caller and return the data and any return messages to the requester. This is called a two-tier, or client-server, type of connection. If the bank application required a connection to the bank web server to connect to a database server, that would be a three-tier connection. If the banking application must connect to its own or the banks application server before making a connection with the web server, API, and the bank’s database management system is called a four-tier connection. (Hellerstein, Stonebraker & Hamilton, 2007)

The next component of the DBMS is the **process manager**. Within the request from the API to the DBMS, the process manager assigns a thread of computation to the query or command and connects the data and control output of the threads back to the communication manager. This takes advantage of a multithreaded server assigning an operating system thread to each DBMS query. Assigning the thread of computation requires a decision from the DBMS to perform admission control. Here, system resources of the database server are quantified, and the admission control task of the process manager can instruct the system to perform the query immediately or defer execution until system resources are adequate. (Hellerstein, Stonebraker & Hamilton, 2007) In our example of checking an account balance, this is when your balance request is weighed in priority to all other requests for data in the banks database. If there are thousands of customers receiving electronic deposits on payday at that exact moment, the process manager could defer your query until some, or all the balances are updated. This is the second task the process manager performs called dispatch and scheduling. If your query was performed early in the morning when a bank doesn’t perform many transactions, your query would be processed immediately.



Thread per DBMS Query (Hellerstein, Stonebraker & Hamilton, 2007)

RELATIONAL QUERY PROCESSOR

A **relational query processo**r converts a SQL statement into a more optimized dataflow execution plan and executes that plan in the DBMS worker. The results of this program are sent back to the DBMS client one record at a time, over multiple trips or in batches. (Hellerstein, Stonebraker & Hamilton, 2007) There are three steps to SQL (*relational*) query processing with multiple embedded subtasks contained in these steps. Step one is the *query parsing procedure*. To parse means to analyze something into its constituent parts. The first parse is a syntax parse. For relational queries, this will commonly be the SQL syntactic validity. Syntax is a set of rules for language about spelling and placement of phrases. If, for example, you queried a SELECT ALL using FORM instead of FROM, that syntax would be invalid, and a control message would be sent to the client instead of the requested data. (Gujral, 2018) Next would be a semantic parse, which would check the query’s validity in relation to database objects and commands associated with that object. An example of a semantic error would be if the query searched for a table name that did not exist. The next task in parsing requires hash values for SQL statements to be compared with previously executed SQL statements for hash value matches. If a different query leads to the same hash value, the parse step would perform a soft parse, and move to the execution step. If the database cannot use existing hash values in the shared pool to execute, the query processor will move to the *optimization step*, called a hard parse. This is also called a library cache miss.

Diagram

Description automatically generated

Query Processing Overview (Gujral, 2018)

The optimization of the SQL query picks up after the hard parse. It is an iterative process in which multiple execution plans are examined and resource cost is determined (time/memory/CPU) software called a *row source generator*. The lowest cost plan is finally moved to the *execution phase*. The SQL engine receives the iterative execution plan called a parse tree and reads the data from the database disk into memory.

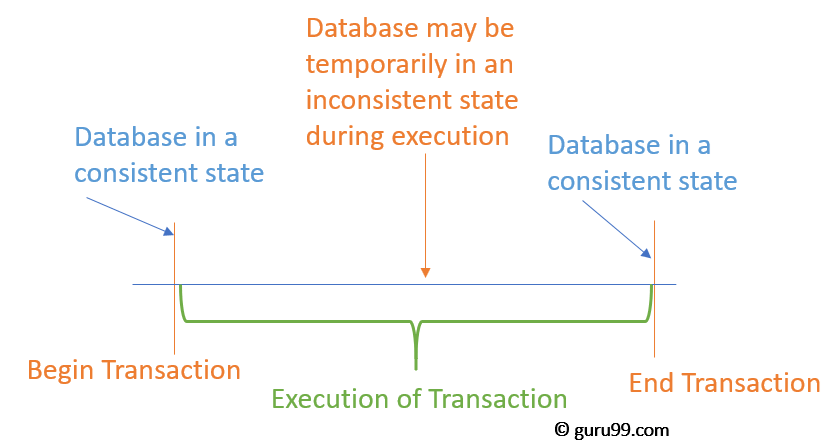
Table

Description automatically generated

Execution Plan showing Cost and Row Source Generation ("SQL Processing", 2017)

TRANSACTIONAL STORAGE MANAGER

The transactional storage manager is another task of the DBMS that controls where and when logical processes for the physical database are performed. In simple terms, a transactional storage manager controls the ability to data storage in space and time. A transaction is everything that occurs because of the execution operation of the relational query from the previous step. During execution the database is in an inconsistent state logically. This is because the data being accessed is in a queue for read/write operations and cannot be accessed by other operations until the execution is finalized. (Peterson, 2022)

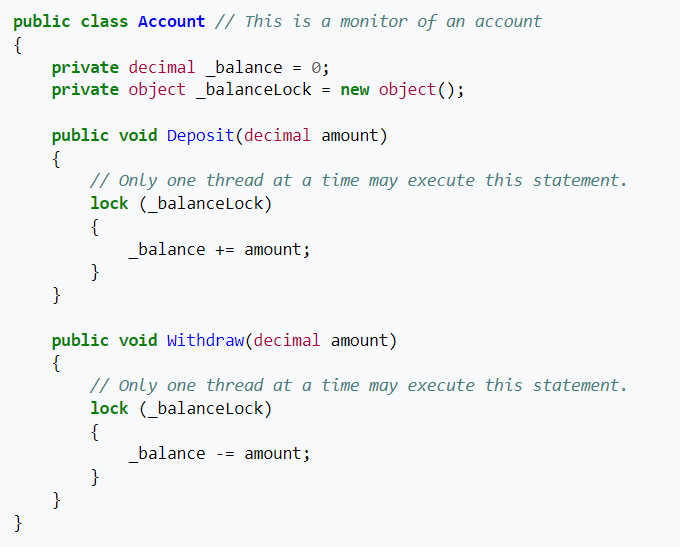


Data Transaction (Peterson, 2022)

The acronym describing the properties of a high-integrity database transaction is **ACID**. *Atomicity* describes a single unit of transaction. There is no partiality of the execution. The transaction fully succeeds or fully fails. This prevents the occurrence of partial writes that can be picked up by another process before the end-state is achieved. An example would be a bank transfer. The execution involves two steps: withdrawing $10 from X account and depositing $10 into Y account. If either step is executed independently, money can be either lost from X account or created in Y account. To achieve atomicity, DBMS implement a log system of the database (*metadata*) to capture consistent states on both ends of the execution. This ensures that an error in isolation of a database object or consistency of the object’ relationships can be reverted to a consistent state. ("Atomicity-Wikipedia", 2022)

*Consistency* is the next ACID property describing database integrity. This property ensures that a database could only be brought from one valid state to another by a transaction. This valid state is defined by maintaining database invariants, preventing database corruption with an illegal transaction, and guaranteeing the relationship of the primary key and foreign key. The example would be withdrawing $10 from an account with $5 available. This would be an illegal transaction. The database would need to revert to $5 available while sending a control message informing of an error in the transaction. This would maintain a consistent state in the database object by limiting the violation of any defined database constraints. (Peterson, 2022)

*Isolation* is another ACID property that defines how or when one client’s transaction can become visible to another client or clients. Isolation can be described as low isolation or high isolation. Low isolation allows the transaction to be seen by more clients with the tradeoff being that concurrency issues arise for data integrity. More clients seeing an object may allow clients to request a transaction with that object. Too many requests to change an object concurrently can lead to invalid or “lost” updates to the object. A high isolation of a database object will also have tradeoffs. It will reduce the concurrency issues of lost updates but is system resource intensive and may block client’s transaction requests. Isolation is implemented at the database level through various systems. On single tier systems temporary tables implement isolation. On 2, 3, and 4 tier systems, a combination of stored procedures and transaction processing managers will implement isolation. ("Isolation - Wikipedia", 2022) Isolation is guaranteed by locking algorithms that prevent two transactions at the same time. There are read locks and write locks The two-phase locking is the most popular mechanism currently and is made up of the expanding phase and the shrinking phase. Locks are acquired in the expanding phase by transactions and released from transactions by the shrinking phase. ("Two-phase locking - Wikipedia", 2022)



Lock Statement C# Example ("lock Statement (C# Reference)", 2013)

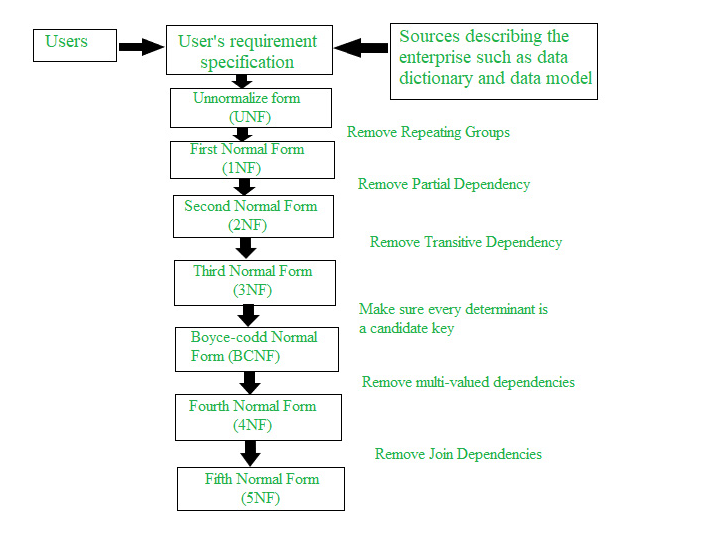
*Durability* is the final ACID property that describes the persistence of transaction changes even during a database system failure. This is achieved by a DBMS recording a transaction into the database log prior to making the changes on the permanent record or confirming the completed transaction to the client. This ensures that if the system crashes, the log manager can perform the unfinished transaction before allowing a read or write process to occur on that object. The transaction is only considered complete when the log has been “flushed” from the temporary log storage to the permanent disk storage. ("Durability - Wikipedia", 2022) This has become especially important in the era of portable electronics. With intermittent connectivity and temporary power supplies, system shutdowns and incomplete transactions have become much more common.

SHARED COMPONENTS AND UTILITIES

There are shared components of a commercial DBMS that are ubiquitous in most architecture but are seen as minor. These include the *catalog manager, memory manager, replication and loading services, administration, monitoring, and batch utilities*. The *catalog manager* is a form of metadata that records the users, schemas, relationships, and indexes of the database. The catalog manager is stored as tables in the memory as well so the client can inspect these at the same time as reviewing the data tables. The *memory allocator* is part of the memory management that portions out database memory for DBMS tasks. Buffer pools tend to be a large part of the allocation, but query operators and optimization can also require a large portion of memory. *Replication and loading services* can copy a database across a network as a warm standby for reliability and backup issues. It is also used for distributed database systems for enterprise systems. These replication services are performed by a log-based sniffer program that copies all the changes made to the main database and copies them to geographically dispersed network or server copies. The *administration, monitoring and batch utilities* are a set of tools used to perform maintenance on the DBMS. These include physical and virtual reorganizations of disk partitions and inserts/deletes on the actual data tables to optimize space and performance. Also included are backup processes and bulk load(*batch*) updates such as data warehouse updates performed once a day. Resource governors are also included and can monitor query performance and provide locks and limits to prevent inefficient queries from monopolizing to much of the database system resources. (Hellerstein, Stonebraker & Hamilton, 2007)

ERRORS ENCOUNTERED IN DATABASE ARCHITECTURE

Normalization is a technique consisting of three or more rules that are applied to data so that redundancy can be reduced in data storage and relations can be established between rows, columns, and tables. It also addresses errors caused by insertion, deletion, or modification of data in the database lifecycle. The first normal form ensures there is a primary key a user can utilize to identify a tuple, or row of related information. (Li, 2019) It also ensures each column is atomic, or unique. If there wasn’t a primary key, users would have to look up instances according to non-unique criteria and could lead to the addition and deletion of information not intended to be modified. Additionally, unique objects in columns makes updating or searching by object characteristics are targeted and don’t change non-related information. (Barick, 2020) The second normal form ensures that all columns relate to the tables primary key. If the table relates to a customer and there are columns describing salesperson attributes, these columns must be moved to a separate table. This step also ensures that modifications meant to affect the customer do not affect the information of the salesperson, or vice versa. The third normal form ensures that there are no in-table dependencies between columns. If a city depends on a zip code or a plate number depends on a car type, there is a transitive dependency. These columns aren’t completely dependent on the primary key and could lead to updating a customer’s city but not the zip code and errors are introduced. Again, a separate column must be created to isolate this transitive dependency. These three normal forms applied to every table in a database will remove the possibility of introducing partial update errors, one of the most common errors on a new database. (Li, 2019)



The Normalization Process (Barick, 2020)

There are many issues a developer could run into that may cause a failure or error in a database. The previous section went over normalization errors, namely **redundancy**, and the solutions to prevent update errors. Another common problem is the **n+1 error**. When a child table is called every time a parent table is, more than one query is needed to accomplish this. Too many queries can lead to a query flood and a reduction in performance. To solve this issue, the database developer initiates eager loading. Eager loading returns the parent object without automatically providing all the child objects. This object deference improves operation efficiency while still providing baseline functionality. ("5 Common Database Errors", 2021) When a developer or architect creates a database, one of the most important criteria needed is the type of data being stored. This simple parameter determines the storage size, server setup, database manager, and many other aspects of the build. **Ignoring data requirements** can lead to storage inadequacies, buffer errors, and management policies. There are many other errors that can occur but eliminating these will ensure that database architecture is built on solid foundation principles. ("5 Common Database Errors", 2021)

SECURITY SOLUTIONS TO DATABASE VULNERABILITIES

Security should be a part of any database architecture plan. The goal is ultimately to protect the data itself under the security principles of **confidentiality**, **integrity**, and **availability**. Most security threats can be addressed by ensuring these three principles. (Kidd, 2020) Confidentiality is the ability to keep unauthorized users from accessing secured data. Data breaches compromise the secret information you are tasked with protecting. Breaches can come from human error, unprotected passwords, or SQL injections. Integrity is just the data’s trustworthiness. A great example of this type of attack was the Stuxnet worm, which compromised the integrity of the nuclear centrifuges rpm settings and sabotaged a nuclear plant. Availability threats come as denial-of-service attacks or ransomware. These attacks prevent you as an administrator or the user from accessing your own information. Confidentiality, integrity, and availability are protected by limited administrator rights, running updated firewalls, maintaining complex passwords, working on a VPN, limiting user devices, and password expiration time limits.

CONCLUSION

The history and architecture of an RDBMS is complex. From starting as a simple library with index cards to maturing into distributed network database system, the RDBMS has a rich background from which its architecture is built. The architectural components of the RDBMS are broken down into five systems here for simplicity, but other authors and other systems can and will add to or take away from these components to suit their uses and users. These should just be referred to as a framework for the functions that need to be addressed when planning a RDBMS build.

REFERENCES

1. Hellerstein, J., Stonebraker, M., & Hamilton, J. (2007). Architecture of a Database System. Foundations And Trends® In Databases, 1(2), 141-259. DOI: 10.1561/1900000002
2. Database- Overview, Roles and Components, DBMS. (2022). Retrieved 9 June 2022, from <https://corporatefinanceinstitute.com/resources/knowledge/data-analysis/database/>
3. Andrews, E. (2016). 8 Legendary Ancient Libraries. Retrieved 9 June 2022, from <https://www.history.com/news/8-impressive-ancient-libraries#:~:text=The%20Library%20of%20Ashurbanipal&text=The%20world's%20oldest%20known%20library,organized%20according%20to%20subject%20matter>.
4. Physical and Logical Databases. (2010). Retrieved 9 June 2022, from <https://docs.microsoft.com/en-us/previous-versions/dynamicsnav-2009/dd355372(v=nav.60)?redirectedfrom=MSDN>
5. Karnes, C. (2021). APIs in Banking: Four Use Cases (And Why You Should Care). Retrieved 21 June 2022, from <https://www.csiweb.com/what-to-know/content-hub/blog/apis-in-banking-four-use-cases-and-why-you-should-care/>
6. Gujral, P. (2018). SQL | Query Processing - GeeksforGeeks. Retrieved 28 June 2022, from <https://www.geeksforgeeks.org/sql-query-processing/>
7. What is Semantics? - Definition from Techopedia. (2018). Retrieved 28 June 2022, from <https://www.techopedia.com/definition/687/semantics-computing#:~:text=Semantics%20is%20a%20linguistic%20concept,to%20both%20humans%20and%20machines>.
8. SQL Processing. (2017). Retrieved 28 June 2022, from <https://docs.oracle.com/database/121/TGSQL/tgsql_sqlproc.htm#TGSQL178>
9. Peterson, R. (2022). Transaction Management in DBMS: What are ACID Properties?. Retrieved 29 June 2022, from <https://www.guru99.com/dbms-transaction-management.html>
10. Atomicity (database systems) - Wikipedia. (2022). Retrieved 29 June 2022, from <https://en.wikipedia.org/wiki/Atomicity_(database_systems)#:~:text=An%20example%20of%20an%20atomic,saving%20it%20to%20account%20B>.
11. Isolation (database systems) - Wikipedia. (2022). Retrieved 20 July 2022, from <https://en.wikipedia.org/wiki/Isolation_(database_systems)>
12. Two-phase locking - Wikipedia. (2022). Retrieved 20 July 2022, from <https://en.wikipedia.org/wiki/Two-phase_locking>
13. Durability (database systems) - Wikipedia. (2022). Retrieved 20 July 2022, from <https://en.wikipedia.org/wiki/Durability_(database_systems)>
14. Li, L. (2019). Database Normalization Explained. Retrieved 27 July 2022, from <https://towardsdatascience.com/database-normalization-explained-53e60a494495>
15. 5 Common Database Errors (and What to do About Them). (2021). Retrieved 27 July 2022, from <https://www.apollotechnical.com/common-database-errors/>
16. Barick, A. (2020). Normalization Process in DBMS - GeeksforGeeks. Retrieved 28 July 2022, from <https://www.geeksforgeeks.org/normalization-process-in-dbms/>
17. Kidd, C. (2020). What Is the CIA Security Triad? Confidentiality, Integrity, Availability Explained. Retrieved 28 July 2022, from <https://www.bmc.com/blogs/cia-security-triad/>
18. Data Integrity Attacks: Welcome to the next level in Cyber Security arena. (2016). Retrieved 28 July 2022, from <https://cyware.com/news/data-integrity-attacks-welcome-to-the-next-level-in-cyber-security-arena-0136466d>