International Journal of Engineering Research & Management Technology

ISSN: 2348-4039

Volume 9, Issue-3 May-June 2022

Email:editor@ijermt.org www.ijermt.org

DATABASE ARCHITECTURE AND MANAGEMENT

Kamalnayan Panda

GD Goenka Public School,Delhi

Abstract:

The modern day IT department of the government has a big challenge in the form of the management of enormous and ever-increasing volumes of data, as well as the delivery of a quality-led software product that maximises the use of capital while minimising associated expenses. Users are able to create, construct, alter, and exchange their databases with one another through the use of the query language, which is a cryptographic protocol, also known as a series of programmers. This protocol offers its customers procedures. People are able to access records, change data, and report or construct data thanks to a collection of software known as a database management system (DBMS). This enables direct access to the database, which may also be monitored. The fact that content management systems were initially presented in the 1960s as such is not a novel occurrence by any stretch of the imagination. The term "database" is typically used to refer to a collection of data and the structure of that data. Database Management System (DBMS) generally requires access to this data, requires an automated software development package, enables users to communicate with one or more repositories, and provides accessibility to all of the data stored in the domain. DBMS also generally requires an automated software development package (although limitations that deny access to certain data that serve a purpose). The three schema architecture for data base management system and the three level object oriented database management system architecture serve as the foundation for this particular design.

keywords: *DBMS, architecture , management*

Introduction

Database Management Systems, sometimes known as DBMSs, are intricate and crucially important software systems. The modern database management systems are the product of decades of study conducted in academia and industry, as well as intensive software development in corporations. Database systems were some of the early online server systems to be extensively implemented, and as a result, they were the pioneers in the development of design solutions that included not only data management but also applications, operating systems, and networked services as well. The earliest database management systems are among the most significant software systems in the history of computer science.

The concepts and implementation challenges that were pioneered for DBMSs are widely replicated and continually reinvented. The lessons that may be learned from database systems architecture are not as widely recognised as they ought to be for a variety of different reasons. To begin, the community of applied database systems is not particularly large. Because market forces can only sustain a few number of rivals at the high end, there are only a few successful implementations of database management systems. The group of people who are active in building and implementing database systems is a close-knit one.

Many of these individuals went to the same schools, worked on the same significant academic projects, and cooperated on the same commercial products. Second, the discussion of database systems in academic circles frequently overlooks architectural concerns. Database systems are generally presented in textbooks with an emphasis on algorithmic and theoretical difficulties, which are natural to teach, study, and test. However, these presentations typically exclude a comprehensive description of the architecture of the system when it is fully implemented. In conclusion, there is a significant amount of received knowledge concerning the construction of database systems, but very little of it has been documented or disseminated to a wide audience.

In this work, we make an attempt to summarise the most important architectural features of contemporary database management systems, including a discussion of more complex subjects. There are mentions of some of them in published works, and we offer references for them when necessary. Other problems are buried deep inside product manuals, while others are simply passed down via the community's oral history. When it is appropriate to do so, we utilise commercial and open-source systems as examples of the many architectural styles that are covered in this article. However, because to the limited amount of space available, it is not possible to enumerate all of the exceptions and subtleties that have been introduced into these multi-million line code bases, the most of which are well over ten years old. In this paper, we want to concentrate on general system design and stress concerns, which are not often covered in textbooks.

We hope that this will provide a helpful context for techniques and ideas that are more commonly recognised. We are going to proceed on the assumption that the reader has prior knowledge of the subject covered in database systems textbooks as well as the fundamental capabilities of contemporary operating systems such as UNIX, Linux, or Windows. Following the discussion of the overarching architecture of a DBMS, which will take place in the following section, we will then present a number of references to background reading on each of the individual components.

Relational Systems: The Life of a Query

Relational database management systems are the most developed and commonly utilised of all the databases that are now in production (RDBMSs). These systems may be found at the heart of a significant portion of the world's application infrastructure, including e-commerce, medical records, billing, human resources, payroll, customer relationship management, and supply chain management, to name a few of the aforementioned fields. The proliferation of web-based commerce and community-oriented websites has done nothing except expand the amount of traffic they receive and the scope of their use. Relational systems serves as the record registries that are used by practically all online businesses and the majority of online content management systems (blogs, wikis, social networks, and the like). Relational database systems, in addition to playing a significant role as infrastructure for software, also act as.

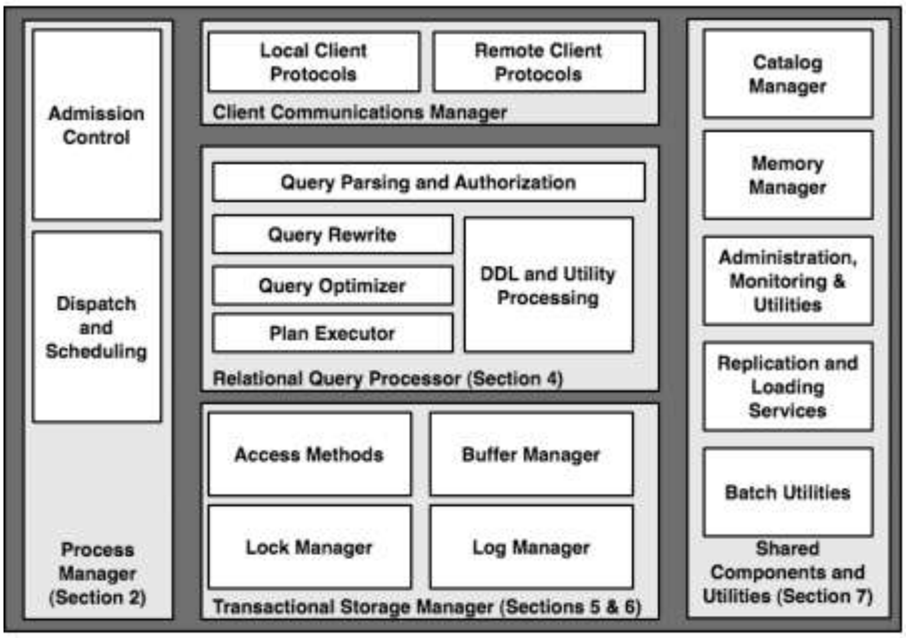
Admission Control
Dispatch and Scheduling
Process Manager (Section 2)
Local Client Protocols
Remote Client Protocols
Client Communications Manager
Query Parsing and Authorization
Query Rewrite
Query Optimizer
Plan Executor
DDL and Utility Processing
Relational Query Processor (Section 4)
Access Methods
Buffer Manager
Lock Manager
Log Manager
Transactional Storage Manager (Sections 5 & 6)
Catalog Manager
Memory Manager
Administration, Monitoring & Utilities
Replication and Loading Services
Batch Utilities
Shared Components and Utilities (Section 7)

Fig. 1.1 Main components of a DBMS.

a point of reference that is widely recognised that may be used for new expansions and revolutions in database systems that may come forth in the future. As a direct consequence of this, the entirety of this study will concentrate on relational database management systems. As shown in Figure 1.1, a typical relational database management system is composed of five primary parts at its core. We are going to walk through the life of a query in a database system as a means of providing an introduction to each of these components and the way in which they fit together. In addition to that, this serves the purpose of providing an outline of the remaining portions of the article. Consider a straightforward yet common database interaction that takes place at an airport, in which a gate agent clicks on a form in order to get the passenger list for a particular aircraft. When you click on this button, a single-query transaction is initiated, which operates more or less as follows:

1. The client, which is the personal computer located at the airport gate, makes a call to an application programming interface (API), which then connects with the Client Communications Manager of a database management system (DBMS) across a network (top of Figure 1.1). Sometimes, this connection is made between the client and the database server directly, such as using the ODBC or JDBC connectivity protocol. Other times, the connection is formed through an intermediary. A "two-tier" or "client-server" system is the name given to this particular configuration. In other circumstances, the client might communicate with a "middle-tier server," which could be anything from a web server to a transaction processing monitor or something similar. This "middle-tier server" would then make use of a protocol to proxy the communication between the client and the DBMS. The common name for this kind of structure is a "three-tier" system. Between the web server and the database management system (DBMS), there is typically a fourth layer known as the "application server" that exists in many web-based situations. Due to the many available choices, a typical database management system (DBMS) has to be compatible with a wide variety of communication protocols. These protocols are utilised by a variety of client drivers and middleware systems. However, at its core, the responsibility of the DBMS' client communications manager in each of these protocols is roughly the same: to establish and remember the connection state for the caller (whether it be a client or a middleware server), to respond to SQL commands from the caller, and to return both data and control messages (result codes, errors, etc.) as appropriate. This is true for all of the

protocols. In the context of our straightforward illustration, the communications manager would first validate the client's identity, then set up a state to remember the specifics of the newly established connection and the current SQL command across calls, and then send the client's initial request deeper into the DBMS so that it can be processed.

2. The "thread of computation" must be assigned to the SQL command as soon as the database management system (DBMS) receives the first SQL command from the client. In addition to this, it is responsible for ensuring that the data and control outputs of the thread are connected to the client via the communications manager. The DBMS Process Manager is responsible for carrying out these responsibilities (left side of Figure 1.1). At this point in the query, the most important choice that the DBMS needs to make concerns admission control. Specifically, the DBMS needs to decide whether the system should begin processing the query immediately or postpone execution until a time when there are sufficient system resources available to devote to this query. We explore Process Management in detail.

3. The gate agent's query will be able to start executing after it has been accepted and given a thread of control to handle. It accomplishes this by calling upon the programme that is stored in the Relational Query Processor (center, Figure 1.1). This collection of modules determines whether or not the user has permission to execute the query and then compiles the SQL query text provided by the user into an internal query plan. After being constructed, the plan executor is responsible for managing the resultant query plan. For the purpose of carrying out any query, the plan executor is made up of a collection of "operators," which are relational algorithm implementations. Typical operators are responsible for implementing relational query processing activities like as joins, selection, projection, aggregation, sorting, and so on. They also make calls to lower levels of the system to retrieve data records. In order to fulfil the requirements of the gate agent's inquiry, our sample query makes use of a limited subset of these operators, which has been compiled through the query optimization procedure. Within Section 4, we will talk about the query processor.

4. There may be one or more operators included in the gate agent's query plan, and these operators are responsible for making data requests to the database. These operators make calls to the Transactional Storage Manager of the DBMS (shown at the bottom of Figure 1.1), which is responsible for managing all data access (read) and modification (create, update, delete) operations. These calls retrieve data from the DBMS. The storage system incorporates algorithms and data structures, known as "access methods," for the purpose of organising and gaining access to the data stored on the disc. These access methods include fundamental structures such as tables and indexes. A buffer management module that decides when and what data should be transferred between memory buffers and disc buffers is also included in this component. To continue with our illustration, while accessing data using the access methods, the gate agent's query has to run the transaction management code in order to guarantee that the well-known "ACID" attributes of transactions [30] are maintained (discussed in more detail in Section 5.1). Locks are requested from a lock management in advance of accessing data in order to guarantee that the execution will proceed correctly despite the presence of other concurrent requests. If the gate agent's inquiry included making changes to the database, the system would communicate with the log manager to make certain that the transaction would be permanent if it was committed and that it would be completely reverted if it was cancelled. In Section 5, we go deeper into the topic of storage and buffer management. The topic of transactional consistency architecture is covered.

5. At this stage in the life of the sample query, it has started to access data records, and it is prepared to use these records to calculate results for the customer. In order to accomplish this, we need to "unwind the stack" of actions that we have discussed up to this point. The access methods hand control back to the operators of the query executor, who are responsible for coordinating the computation of result tuples based on the data in the database. As the result tuples are generated, they are placed in a buffer for the client communications manager, who is responsible for sending the results back to the person who made the original call. The client will often make extra calls to retrieve more data progressively from the query when dealing with huge result sets. This will result in many iterations via the communications manager, query executor, and storage manager. At the conclusion of the query in our straightforward illustration, the transaction is finished and the connection is closed. As a consequence of this, the transaction manager cleans up state for the transaction, the process manager frees any control structures related to the query, and the communications manager cleans up communication state related to the connection.

Process per DBMS Worker

The process per DBMS worker paradigm, shown in Figure 2.1, was utilised by early DBMS implementations and continues to be utilised by a great deal of commercially available software today. Because DBMS employees are mapped directly onto OS processes, this paradigm may be put into action with only a moderate amount of effort. The timesharing of DBMS workers is managed by the OS scheduler, and the DBMS programmer may rely on OS protection capabilities to isolate common issues such as memory overruns. In addition, a wide variety of programming tools, such as memory checks and debuggers, work very well with this process paradigm. This paradigm is made more difficult by the in-memory data structures that are shared across all DBMS connections. Some examples of these structures are the lock table and the buffer pool (discussed in more detail in Sections 6.3 and 5.3, respectively). These shared data structures have to be explicitly allotted in shared memory that is supported by the operating system and is available to all DBMS processes. This needs OS support, which is readily accessible, in addition to some specialised DBMS programming. Given that a significant portion of the memory that is considered "interesting" is shared between processes, it is necessary for this model to make heavy use of shared memory, which, in fact, nullifies some of the benefits that are associated with the separation of address spaces. Process per DBMS worker is not the most desirable process architecture in terms of scalability to extremely large numbers of concurrent connections. Scalability problems emerge due to the fact that a process maintains more state than a thread does and, as a consequence, uses more memory. To switch processes, you have to switch the process context, which includes the security context, memory management state, file and network handle tables, and any other relevant process context. A thread switch eliminates the requirement for this. Despite this, the process per DBMS worker paradigm is still widely used and is supported by a variety of databases, including Oracle, PostgreSQL, and IBM DB2.

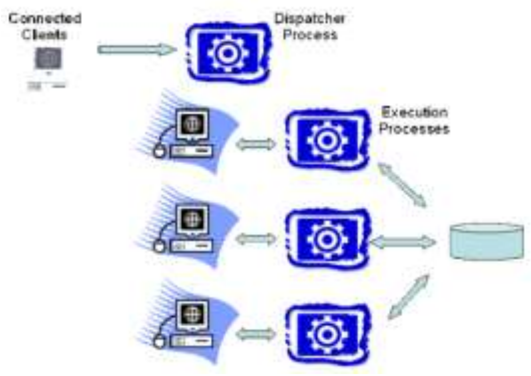

Fig. 2.1 Process per DBMS worker model: each DBMS worker is implemented as an OS process.

2.1.2 Thread per DBMS Worker

All of the DBMS worker activity is hosted within a single multithreaded process according to the thread per DBMS worker architecture (Figure 2.2). A dispatcher

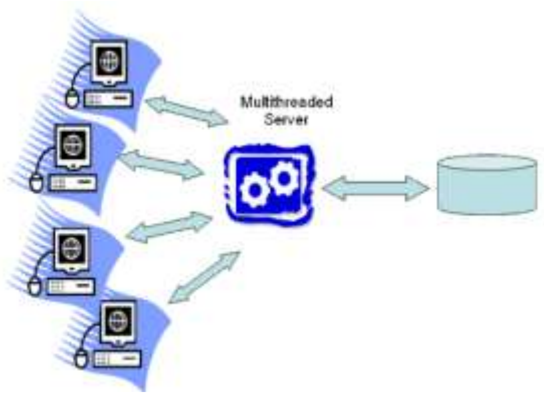

Fig. 2.2 Thread per DBMS worker model: each DBMS worker is implemented as an OS thread.

thread or a small number of similar threads acts as a listener for new DBMS client connections. A new thread is created for each connection that is made. Following the submission of a SQL request by a client, the request is carried out in its entirety by the associated thread operating a DBMS worker. This thread operates within the DBMS process, and after it has finished processing, the result is sent back to the client. The thread then waits on the connection for the subsequent request to come from the same client. This architecture presents the typical difficulties associated with multi-threaded programming: the operating system does not protect threads from each other's memory overruns and stray pointers; debugging is difficult, particularly when dealing with race conditions; and the software may be difficult to port from one operating system to another due to differences in threading interfaces and the scaling of multi-threaded processes. Because of the significant usage of shared memory, many of the multi-programming issues that are present in the thread per DBMS worker model are also present in the process per DBMS worker model. This is owing to the fact that both models make considerable use of shared memory. In recent years, the thread API variations that exist between operating systems have been diminished; yet, tiny discrepancies between platforms continue to present complications when debugging and tuning. Ignoring these implementation challenges, the thread per DBMS worker paradigm scales effectively to huge numbers of concurrent connections. As a result, it is utilised in various production DBMS systems of the current generation, such as Sybase, MySQL, Informix, IBM DB2, and Microsoft SQL Server.

Database Management System

A repository is a collection of organised data that is often kept in an electronic format, is registered, and can be accessed via a software system. Structured design and modelling methodologies are frequently utilised in contexts where computer systems are of a more complex nature. Database management systems are pieces of software that allow for interaction with end users, implementations, and the database itself for the purposes of data collecting and analysis (DBMS). In point of fact, the DBMS package offers fundamental features that are necessary for the application's ongoing maintenance. In its most basic sense, the term "database" refers to a collection of data that is organised in a specific manner. Transport to such data is typically provided by a DBMS, also known as a Database Management System, which is comprised of an incorporated software package. This system enables customers to communicate in one or more repositories and enables direct exposure to all of the information that is collected in the database. The database management system (DBMS) possesses a variety of features that, among other things, enable huge volumes of data to be input, saved, and retrieved, as well as techniques for controlling how this data is arranged. Because of the close connection that exists between them at the present moment, the term "database" is occasionally used in a way that is not explicit to refer to a repository or DBMS for the purpose of its management. Outside of the realm of corporate information technology, the word "database" is frequently used with the intention of implementing a database management programme in any group of connected data (for example, a database or a card index) as scale and usage criteria. The primary features of existing database management systems (DBMSs) are data interpretation, adjustment, and the elimination of interpretations that define the institution of the data. This is something that enables the administrators of a repository and its data, which can also be divided into four primary practical communities. The process of providing information in a format that is either instantly retrievable or capable of being retrieved by further programmes is referred to as "retrieval." Inclusion, modification, and omission of authentic data. The information that was acquired can be made available as a consequence of either updating or combining existing information from the registry in a form that is substantially equivalent to the type that is stored in the computer database or in a form that is more recent.

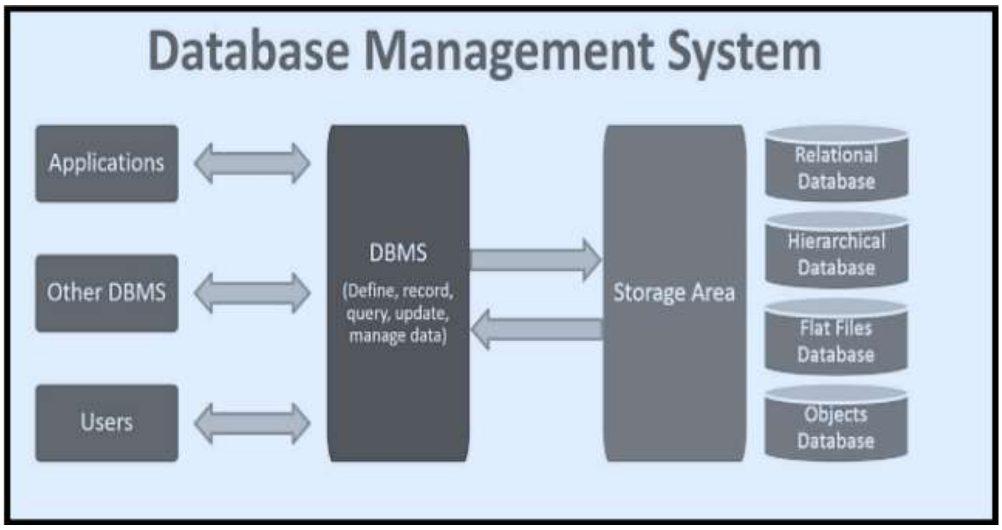

Figure 1 Database Management System

The following is a list of the modules and components that make up the standard database management system. A database module that serves as the primary layer of the system and is responsible for dealing with applications, users/clients, and other sub-databases. The database module communicates with the programme and the users in order to define, update, query, and otherwise manage the data that has been saved. The database module has an associated storage area, and this area contains certain pre-defined parts, including relational, hierarchical, flat file, and object sections.

A. Three Schema Architecture

One example of a representational design for database systems is known as the Three Schema architecture. In order to achieve the goal of separating the user applications from the real database, it offers support for numerous user views and data independence for the programmes. Fig. shows that this architecture is divided into three layers. 1 Internal level: this level is formed of the internal schema, which explains the actual physical storage of data. This level is the lowest one in the hierarchy. 2) The conceptual level: this level is formed of the conceptual schema and explains the organisation of the whole database for a community of users. It focuses on identifying entities, connections, user activities, and other limitations rather than the specifics of data storage at the physical level, thus it may conceal such features.

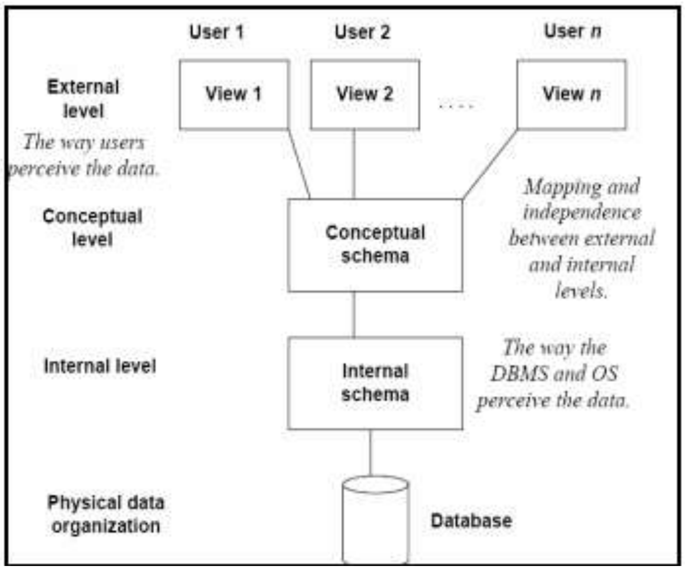
User 1
User 2
User n
External level
The way users perceive the data.
View 1
View 2
View n
Conceptual level
Conceptual schema
Mapping and independence between external and internal levels.
Internal level
Internal schema
The way the DBMS and OS perceive the data.
Physical data organization
Database

Figure 1. Three Schema architecture

3) External level: This level provides an external view to end users i.e. it provides the end users with that part of database in which they are interested and hides other low level details

The above three schema architecture is well suited to the needs of a relational database environment, but it is not suitable for a cloud environment. This is because a cloud environment requires access by many different types of users, each of whom has different service requirements. As a result, a cloud environment requires a higher level of customization. Additionally, cloud systems have severe requirements for both privacy and security, neither of which can be satisfied by this design.

Three-Level Object-Oriented Database Architecture

The three-level object-oriented database architecture, which is used for object-oriented database access and administration, is based on updatable views and offers a mapping of stored items onto virtual objects. This design was developed by Microsoft. The middle layer of this design is what's known as a database management system (DBMS) governed middle layer. This architecture defines the following user roles and responsibilities: Fig.2 Database programmer is responsible for creating internal and conceptual schema of the data based upon previously created design, as per the business requirements. Its features include that it is transparent, provides users with the ease of management and modifications. This architecture defines the following user roles and responsibilities: Fig.1 Database administrator is responsible for creating internal and conceptual schema of the data based upon previously created design, as per the business requirements. Fig The external schema for specific users is something that the database administrator is responsible for establishing. It is up to him or her to generate viewable updates that are based on the data store. A database user is an application programmer because an application programmer utilises a database and is familiar with the interfaces and views that are supplied by the database administrator.

As a result, the aforementioned three players collaborate with one another to offer administration of user rights in addition to guiding the development of database applications. Consequently, the Three-level Object-Oriented Database Architecture is an excellent choice for an object-oriented database.

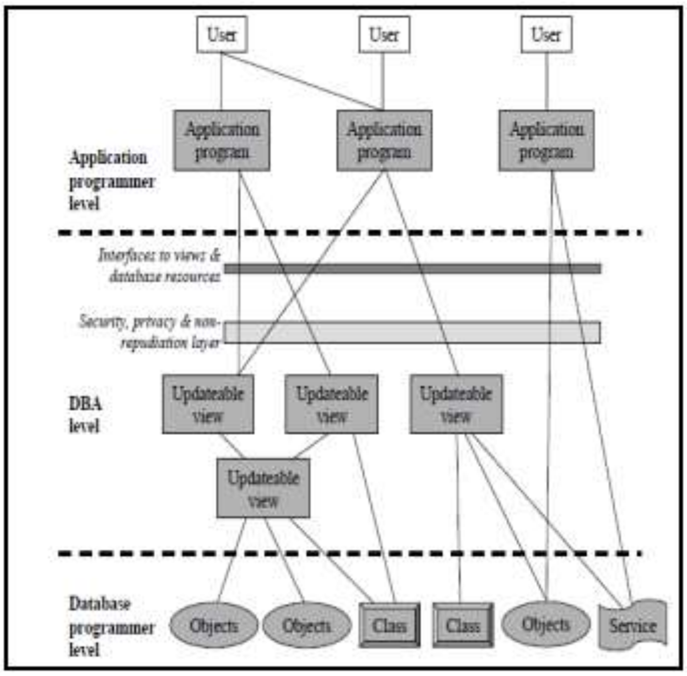

Figure 2.A conceptual view of three levels Object Oriented Architecture

Since we are aware that the computing environment for a cloud-based system is very different from that of traditional database environments, as well as environments that are object-oriented and XML-oriented, with an emphasis on user data privacy, security, scalability, elasticity, availability of resources, and so on, we will begin by defining what we mean by "computing environment." According to the specifics of their use cases and circumstances, many cloud customers may have varying requirements for the kinds of specialised services and access restrictions that they require. As a result, in order to overcome the limitations of applicability of three schema architecture for database and Three-Level Object Oriented Database Architecture, we have proposed an architecture for cloud data access. This architecture has advantages over the existing architecture for database access in the areas of physical access, logical access, and technical access. The following is a list of some of the most significant concerns that our suggested solution addresses:

- Customization: given that the data stored in the cloud will be used by a wide range of customers, each of whom will have unique resource requirements. Every user of a cloud will have resources made accessible to them through the cloud that are tailored to their own needs.
- Conceptual modelling: Clients will utilise the data stored in the cloud in a variety of different ways to fulfil their specific requirements. Therefore, each user will have their own unique perspective on the data stored in the cloud.
- Hiding information: A cloud user should be restricted to using just the portion of the service that corresponds to the needs that they have expressed.
- Security, privacy, and access to cloud data: administrators and servers of cloud data bases should have flexible facilities to give and access resources to users according to the access permissions allowed to those users.

- Orchestration and choreography of services: because cloud computing is built on scaling and pay per use, linking and automating of work flows, orchestration and choreography of cloud services are vitally significant. Cloud services must be coordinated at every tier.

Conclusion

This study investigates databases as well as the many kinds of databases. In the course of writing this paper, I came across a number of different database definitions. In conclusion, the database is often an accumulation of data that is managed and kept up to date on a consistent basis. This is done in order to access the material that has been recorded. The typical databases will typically contain accumulated stores of data records or file folders. This study examines a database that is stored in the cloud as well as a database that is based on semantics, and it comes to the conclusion that different kinds of databases each have their own advantages and disadvantages depending on the situation. For example, in order to manage meta-data, also known as data about data, a large storage capacity is required; therefore, in this scenario, a cloud-based database works efficiently. On the other hand, if the customer wants a specific amount of data and the data is of a specific type, then a semantic-based database works efficiently. Well-defined architectures for relational dbms and object-oriented data are the Three Schema Architecture and the Three-Level Object-Oriented.

Reference:

1. A. Boicea, F. Radulescu, and L. I. Agapin, “MongoDB vs Oracle - Database comparison,” in Proceedings - 3rd International Conference on Emerging Intelligent Data and Web Technologies, EIDWT 2012, 2012.
2. F. Färber et al., “The SAP HANA Database -- An Architecture Overview.,” IEEE Data Eng. Bull., 2012.
3. A. Pavlo et al., “Self-driving database management systems,” in CIDR 2017 - 8th Biennial Conference on Innovative Data Systems Research, 2017.
4. J. DeBrabant, A. Pavlo, S. Tu, M. Stonebraker, and S. Zdonik, “Anti-caching: A new approach to database management system architecture,” Proc. VLDB Endow., 2013.
5. A. Chatr-Aryamontri et al., “The BioGRID interaction database: 2015 update,” Nucleic Acids Res., 2015.
6. B. Alam, M. N. Doja, M. Alam, and S. Mongia, “5-Layered Architecture of Cloud Database Management System,” AASRI Procedia, 2013.
7. Gerard Conway and Edward Curry, “Managing Cloud Computing: A life cycle approach”, 2nd International Conference on Cloud Computing and Services Science (CLOSER 2012), 2012, pp. 198-207.
8. Ashraf Aboulnaga, Kenneth Salem, Ahmed A. Soror, Umar Farooq Minhas,Peter Kokosielis, Sunil Kamath “Deploying Database Appliances in the Cloud”,Bulletin of the IEEE Computer Society Technical Committee on Data Engineering”, 2009.
9. Donald Kossmann, Tim Kraska, Simon Loesing, “An Evaluation of Alternative Architectures for Transaction Processing in the Cloud”, SIGMOD’10, 2010.
10. Mladen A. Vouk, “Cloud Computing – Issues ,Research and Implementations", Journal of Computing and Information Technology - CIT , 2008,pp 235–246.
11. Sunguk Lee, “Shared-Nothing vs. Shared-Disk Cloud Database Architecture”, International Journal of Energy, Information and Communications Vol. 2, Issue 4, 2011.

12. Piotr Habela1, Krzysztof Stencel, Kazimierz Subieta, “Three-Level Object- Oriented Database Architecture Based on Virtual Updateable Views1” ADVIS 2006,Fourth Biennial International Conference on Advances in Information Systems, Volume 4243, 2006, 2006,pp 80-89.

13. D.C. Tsichritzis, A. Klug (eds.): The ANSI/X3/SPARC DBMS Framework: Report of the Study Group on Data Base Management Systems, Information Systems , 1978.