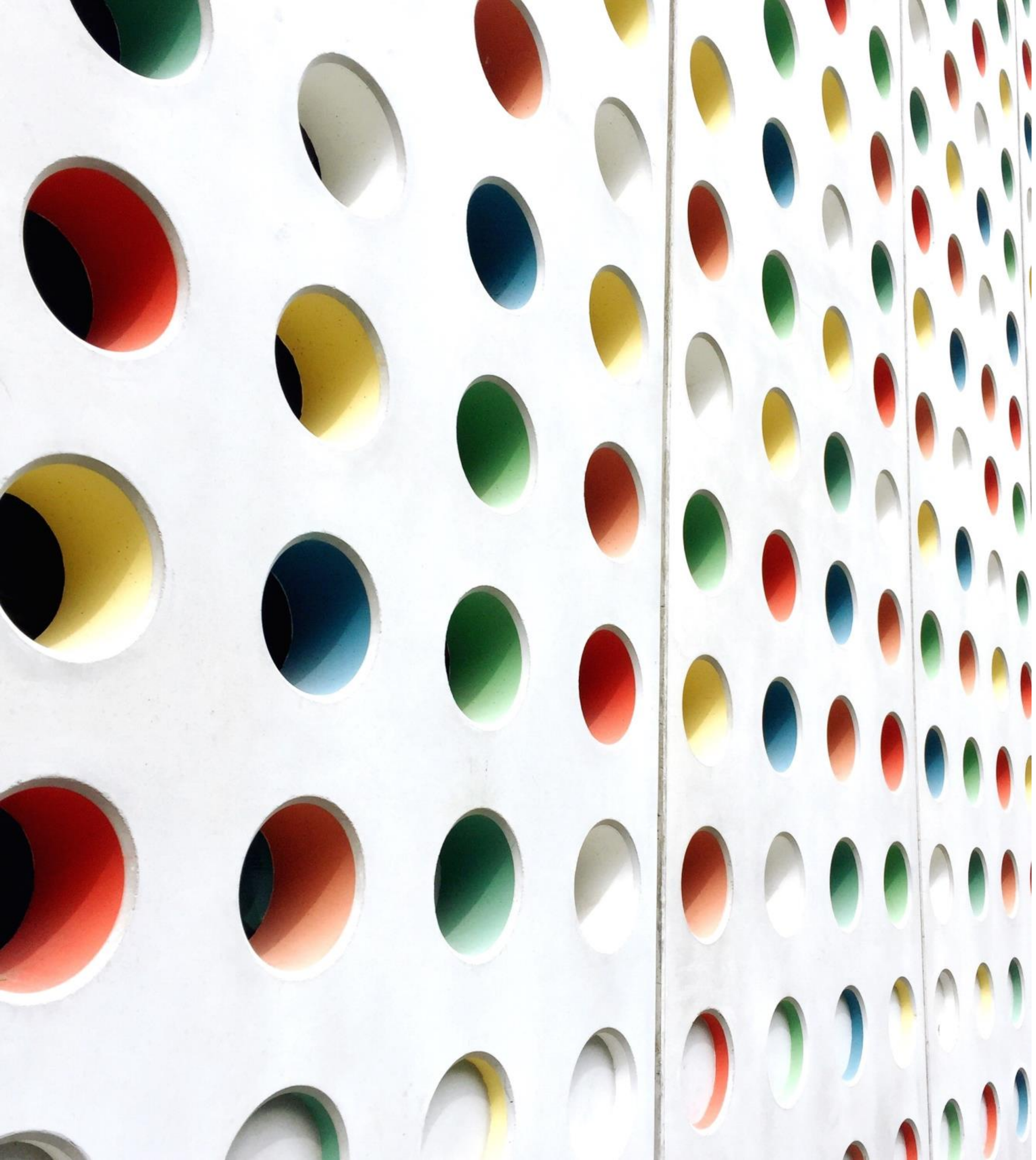
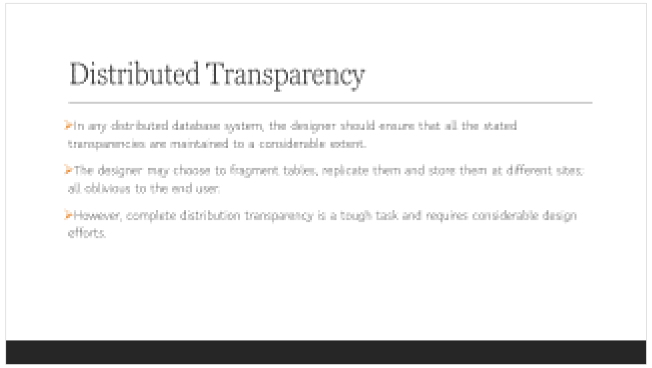


CS685 – DISTRIBUTED DATABASES

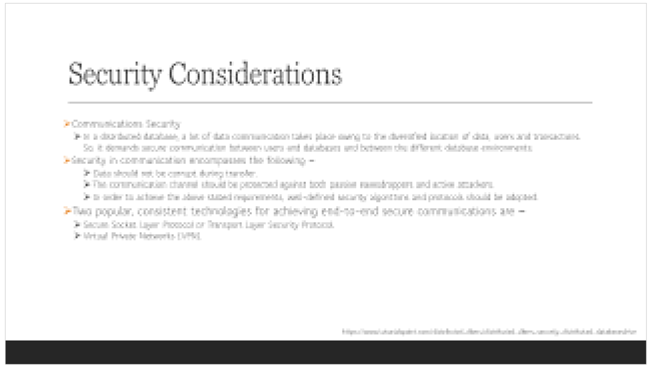
WEEK 2 – DISTRIBUTED DATABASES

Topics



Distributed Transparency

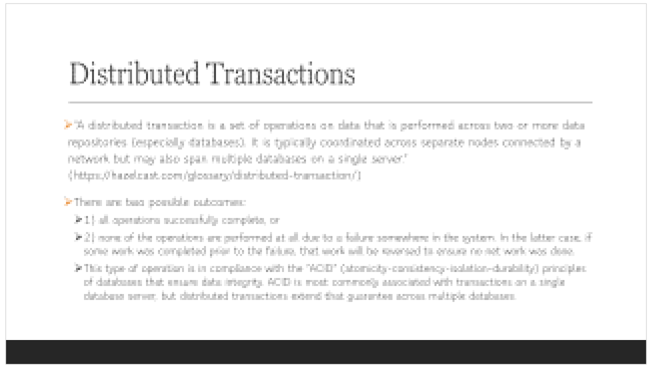
- In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent.
- The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user.
- However, complete distribution transparency is a tough task and requires considerable design efforts.



Security Considerations

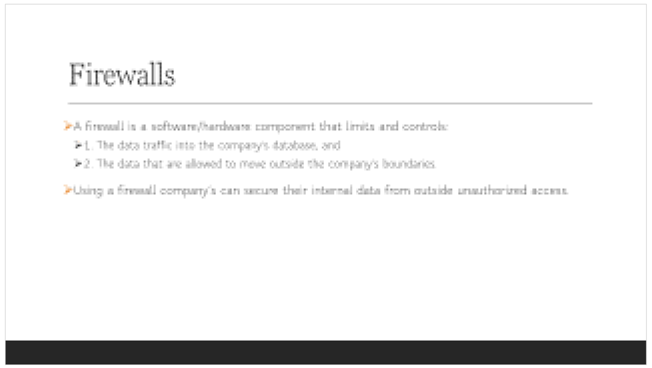
- Communications Security
  - In a distributed database, a lot of data communication takes place owing to the diversified location of data, users and transactions. So, it demands secure communication between users and databases and between the different database environments.
- Security in communication encompasses the following −
  - Data should not be corrupt during transfer.
  - The communication channel should be protected against both passive eavesdroppers and active attackers.
  - In order to achieve the above stated requirements, well-defined security algorithms and protocols should be adopted.
- Two popular, consistent technologies for achieving end-to-end secure communications are −
  - Secure Socket Layer Protocol or Transport Layer Security Protocol.
  - Virtual Private Networks (VPN).

[illegible]



Distributed Transactions

- "A distributed transaction is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network but may also span multiple databases on a single server." (https://hazelcast.com/glossary/distributed-transaction/)
- There are two possible outcomes:
  - 1) all operations successfully complete, or
  - 2) none of the operations are performed at all due to a failure somewhere in the system. In the latter case, if some work was completed prior to the failure, that work will be reversed to ensure no net work was done.
  - This type of operation is in compliance with the "ACID" (atomicity-consistency-isolation-durability) principles of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions extend that guarantee across multiple databases.



Firewalls

- A firewall is a software/hardware component that limits and controls:
  - 1. The data traffic into the company's database, and
  - 2. The data that are allowed to move outside the company's boundaries.
- Using a firewall company's can secure their internal data from outside unauthorized access.

Distributed Transparency

- In any distributed database system, the designer should ensure that all the stated transparencies are maintained to a considerable extent.
- The designer may choose to fragment tables, replicate them and store them at different sites; all oblivious to the end user.
- However, complete distribution transparency is a tough task and requires considerable design efforts.

Distributed Transparency

- Allows a distributed databases to be treated as a single logical database.
- If the DDBMS displays transparency the users does not need to know:
  - That the data are partitioned.
  - That the data can be replicated at several sites.
  - The data location.
- There are three levels of distribution transparency.
  - Fragmentation transparency
  - Location transparency
  - Local mapping transparency

We will cover each three on the next slides.

Fragmentation Transparency

- This is the highest level of transparency.
- The end user or programmer does not need to know the database is partitioned.
- Database accesses are based on the global schema,. so, the user does not need to specify fragment names or data locations..
- Fragmentation transparency enables users to query upon any table as if it were unfragmented.

Location Transparency

- Location is the middle level of distribution transparency.
- Exists when the end user or programmer must specify the database fragment names but not where those fragments are located.
- The actual location where the file is stored doesn't matter to the user.

Local Mapping Transparency

- Exists when the end user or programmer must specify both the fragment names and their locations.
- To illustrate the use of various transparency levels, suppose you have an EMPLOYEE table containing the attributes EMP_NAME, EMP_DOB, EMP_ADDRESS, EMP_DEPARTMENT, and EMP_SALARY. The EMPLOYEE data are distributed over three different locations: New York, Atlanta, and Miami.
- The table is divided by location; that is, New York employee data are stored in fragment E1, Atlanta employee data are stored in fragment E2, and Miami employee data are stored in fragment E3. Consider the following figure.

Transparency Examples

Now suppose that the end user wants to list all employees with a date of birth prior to January 1, 1960. To focus on the transparency issues, also suppose that the EMPLOYEE table is fragmented, and each fragment is unique.

The unique fragment condition indicates that each row is unique, regardless of the fragment in which it is located. Finally, assume that no portion of the database is replicated at any other site on the network.

Case 1: The Database Supports Fragmentation Transparency

Case 2: The Database Supports Location Transparency

Case 3: The Database Supports Local Mapping Transparency

The query conforms to a non-distributed database query format; that is, it does not specify fragment names or locations.
The query reads:

```
SELECT *
FROM EMPLOYEE
WHERE EMP_DOB < '01-JAN-
1963';
```

Fragment names must be specified in the query, but the fragment's location is not specified. The query reads:

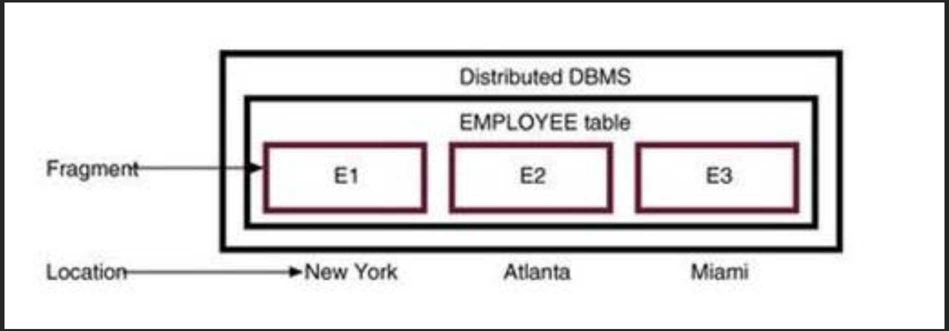
```
SELECT *
FROM E1
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E2
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E 3
WHERE EMP_DOB < '01-JAN-1960';
```

Both the fragment name and its location must be specified in the query. Using pseudo-SQL:

```
SELECT *
FROM El NODE NY
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT *
FROM E2 NODE ATL
WHERE EMP_DOB < '01-JAN-1960';
UNION
SELECT * FROM E3 NODE MIA
WHERE EMP_DOB < '01-JAN-1960';
```

http://www.myreadingroom.co.in/notes-and-studymaterial/65-dbms/553-different-types-of-distribution-transparency.html#:~:text=Local%20mapping%20transparency%20exists%20when,EMP_ADDRESS%2C%20EMP_DEPARTMENT%2C%20and%20EMP_SALARY.

Transparency Examples

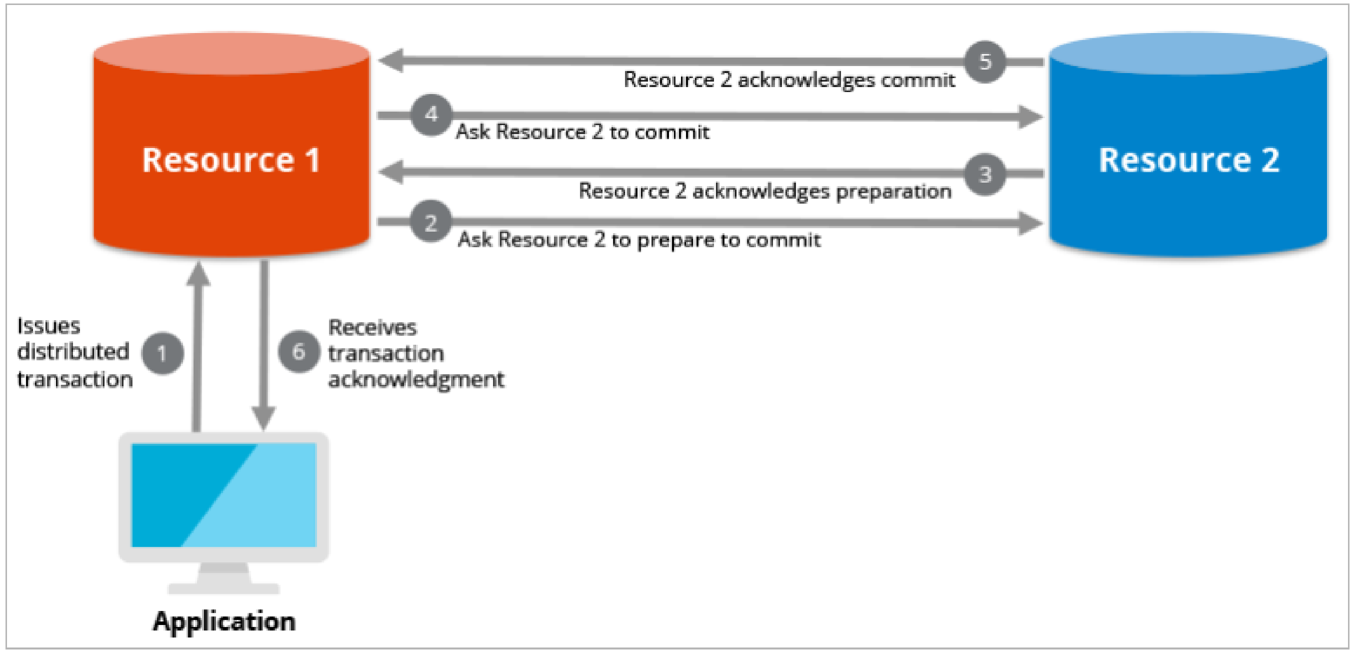


Distributed Transactions

- "A distributed transaction is a set of operations on data that is performed across two or more data repositories (especially databases). It is typically coordinated across separate nodes connected by a network but may also span multiple databases on a single server." (https://hazelcast.com/glossary/distributed-transaction/)
- There are two possible outcomes:
  - 1) all operations successfully complete, or
  - 2) none of the operations are performed at all due to a failure somewhere in the system. In the latter case, if some work was completed prior to the failure, that work will be reversed to ensure no net work was done.
  - This type of operation is in compliance with the "ACID" (atomicity-consistency-isolation-durability) principles of databases that ensure data integrity. ACID is most commonly associated with transactions on a single database server, but distributed transactions extend that guarantee across multiple databases.

Distributed Transactions

- ➢The operation known as a “two-phase commit” (2PC) is a form of a distributed transaction. “XA transactions” are transactions using the XA protocol, which is one implementation of a two-phase commit operation.


Resource 1
Resource 2
5 Resource 2 acknowledges commit
4 Ask Resource 2 to commit
3 Resource 2 acknowledges preparation
2 Ask Resource 2 to prepare to commit
Issues distributed transaction 1
6 Receives transaction acknowledgment
Application

A distributed transaction spans multiple databases and guarantees data integrity.

How Do Distributed Transactions Work?

- Distributed transactions have the same processing completion requirements as regular database transactions, but they must be managed across multiple resources, making them more challenging to implement for database developers.
- The multiple resources add more points of failure, such as the separate software systems that run the resources (e.g., the database software), the extra hardware servers, and network failures.
- This makes distributed transactions susceptible to failures, which is why safeguards must be put in place to retain data integrity.

How Do Distributed Transactions Work?

- For a distributed transaction to occur, transaction managers coordinate the resources (either multiple databases or multiple nodes of a single database).
- The transaction manager can be one of the data repositories that will be updated as part of the transaction, or it can be a completely independent separate resource that is only responsible for coordination.
- The transaction manager decides whether to commit a successful transaction or rollback an unsuccessful transaction, the latter of which leaves the database unchanged.

How Do Distributed Transactions Work?

- First, an application requests the distributed transaction to the transaction manager. The transaction manager then branches to each resource, which will have its own “resource manager” to help it participate in distributed transactions.
- Distributed transactions are often done in two phases to safeguard against partial updates that might occur when a failure is encountered.
- The first phase involves acknowledging an intent to commit, or a “prepare-to-commit” phase. After all resources acknowledge, they are then asked to run a final commit, and then the transaction is completed

How Do Distributed Transactions Work?

- We can examine a basic example of what happens when a failure occurs during a distributed transaction. Let's say one or more of the resources become unavailable during the prepare-to-commit phase.
- When the request times out, the transaction manager tells each resource to delete the prepare-to-commit status, and all data will be reset to its original state.
- If instead, any of the resources become unavailable during the commit phase, then the transaction manager will tell the other resources that successfully committed their portion of the transaction to undo or "rollback" that transaction, and once again, the data is back to its original state. It is then up to the application to retry the transaction to make sure it gets completed.

Why Do You Need Distributed Transactions?

- Distributed transactions are necessary when you need to quickly update related data that is spread across multiple databases.
- For example, if you have multiple systems that track customer information and you need to make a universal update (like updating the mailing address) across all records, a distributed transaction will ensure that all records get updated.
- And if a failure occurs, the data is reset to its original state, and it is up to the originating application to resubmit the transaction.

When Distributed Transactions Are Not Needed

- In some environments, distributed transactions are not necessary, and instead, extra auditing activities are put in place to ensure data integrity when the speed of the transaction is not an issue.
- The transfer of money across banks is a good example. Each bank that participates in the money transfer tracks the status of the transaction, and when a failure is detected, the partial state is corrected.
- This process works well without distributed transactions because the transfer does not have to happen in (near) real time. Distributed transactions are typically critical in situations where the complete update must be done immediately.

Security Considerations

- Communications Security
  - In a distributed database, a lot of data communication takes place owing to the diversified location of data, users and transactions. So, it demands secure communication between users and databases and between the different database environments.
- Security in communication encompasses the following −
    - Data should not be corrupt during transfer.
    - The communication channel should be protected against both passive eavesdroppers and active attackers.
    - In order to achieve the above stated requirements, well-defined security algorithms and protocols should be adopted.
- Two popular, consistent technologies for achieving end-to-end secure communications are −
  - Secure Socket Layer Protocol or Transport Layer Security Protocol.
  - Virtual Private Networks (VPN).

https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_security_distributed_databases.htm

Data Security

In distributed systems, it is imperative to adopt measure to secure data apart from communications. The data security measures are −

- Authentication and authorization − These are the access control measures adopted to ensure that only authentic users can use the database. To provide authentication digital certificates are used. Besides, login is restricted through username/password combination.
- Data encryption − The two approaches for data encryption in distributed systems are −
  - Internal to distributed database approach: The user applications encrypt the data and then store the encrypted data in the database. For using the stored data, the applications fetch the encrypted data from the database and then decrypt it.
  - External to distributed database: The distributed database system has its own encryption capabilities. The user applications store data and retrieve them without realizing that the data is stored in an encrypted form in the database.
- Validated input − In this security measure, the user application checks for each input before it can be used for updating the database. An un-validated input can cause a wide range of exploits like buffer overrun, command injection, cross-site scripting and corruption in data.

https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_security_distributed_databases.htm

Data Auditing

A database security system needs to detect and monitor security violations, in order to ascertain the security measures, it should adopt. It is often very difficult to detect breach of security at the time of occurrences. One method to identify security violations is to examine audit logs. Audit logs contain information such as −

- Date, time and site of failed access attempts.
- Details of successful access attempts.
- Vital modifications in the database system.
- Access of huge amounts of data, particularly from databases in multiple sites.

All the above information gives an insight of the activities in the database. A periodical analysis of the log helps to identify any unnatural activity along with its site and time of occurrence. This log is ideally stored in a separate server so that it is inaccessible to attackers.

https://www.tutorialspoint.com/distributed_dbms/distributed_dbms_security_distributed_databases.htm

Firewalls

- A firewall is a software/hardware component that limits and controls:
  - 1. The data traffic into the company's database, and
  - 2. The data that are allowed to move outside the company's boundaries.
- Using a firewall company's can secure their internal data from outside unauthorized access.

What is a Database Firewall?

- Database Firewalls are a kind of application firewall that monitors database traffic to detect and protect against database-specific attacks that primarily seek to access sensitive data held in the databases.
- Database Firewalls also allow for the monitoring and auditing of all access to cloud databases via logs.
- Database Firewalls, in general, are security-hardened software that is installed either in line with the database server directly before the database server or near the network gateway when safeguarding numerous databases on multiple servers.

https://satoricyber.com/data-security/database-firewall-101-everything-there-is-to-know-about-a-database-firewall/

How Database Firewalls Protect Databases

- Database Firewalls come with pre-defined, adjustable security audit policies that can detect database threats based on previous occurrences or threat patterns known as "signatures."
- As many tasks inside a database get implemented as a sequence of executable SQL statements, the SQL statements or queries get compared to these signatures, which are updated often by the manufacturers to identify known database attacks.
- However, not all database attacks are well-known. As a result, Database Firewalls in certain cases create or include an allow list of safe SQL statements. All input commands get checked against this whitelist, and only those that match the whitelist receive access to the Database.

How Database Firewalls Protect Databases

- Many Database Firewalls can also detect database, operating system, and protocol vulnerabilities in databases and alert the administrator, who can then take appropriate action to repair them.
- Nonetheless, certain Database Firewalls can monitor database answers to prevent data leakage. Instead of immediately banning questionable operations, database firewalls can alert users.

How Database Firewalls Protect Databases

- Certain Database Firewalls can assess criteria such as IP address, time, location, type of applications, and others from which irregular database access requests originate and then decide whether to block them depending on the administrator’s policies.
- However, False positives and False negatives in Database Firewalls are a concern.

Database Firewalls Use Cases

- Prevents the Transmission of Unwanted Information
- Secures Smooth Business Operations
- Ensures Security Based on Protocol and IP Address

Database Firewall vs. Web Application Firewall

- On the other hand, a network firewall safeguards a secure local-area network from unauthorized access and attacks. Its main goal is to distinguish a secure zone from a less secure zone and control communication between them.
  - Without it, every computer with a public Internet Protocol (IP) address is vulnerable to attack from outside the network.
- In a nutshell, Database Firewalls are a category of application firewalls that monitor traffic specific to databases to detect and protect against database-specific attacks.
  - Which typically seek to access sensitive data held in databases.

Individual Project – Week 2

A fragmentation strategy that meets the objectives of the organization must be developed using careful planning. To that end, to ensure correctness of fragmentation, 3 rules must be adhered to during the planning of a fragmentation strategy: (1) completeness, (2) reconstruction, and (3) disjointness.

Distributed Database Implementation Plan: New Content

In 4-5 pages, write a Distributed Query Processing Plan section that includes the following:

Diagram and label the component architecture for the DDBMS including the following:

- Local database management system (LDBMS) component
- Data communications (DC) component
- Global system catalog (GSC)
- Distributed DBMS (DDBMS) component
- Create a logical, and physical design diagram for the proposed distributed database.
- Describe the fragmentation strategy and the rationale (based on usage) for each. Be sure to include fragments (subrelations) and types (horizontal or vertical) that you are proposing.
- Describe the quantitative and qualitative information used to justify the fragmentation strategy that will be used to support query processing.

All sources should be cited both in-text and in the References, section using APA format.

Name the document "yourname_CS685_IP2.doc.

References

Ozsu, M. T., & Valduriez, P. (2020). Principles of Distributed Database Systems (4 ed.). https://doi.org/10.1007/978-3-030-26253-2

Rob, P., & Coronel, C. (2000). Database Systems: Design, Implementation and Management (4th ed.). Course Technology.