Data & Database Security

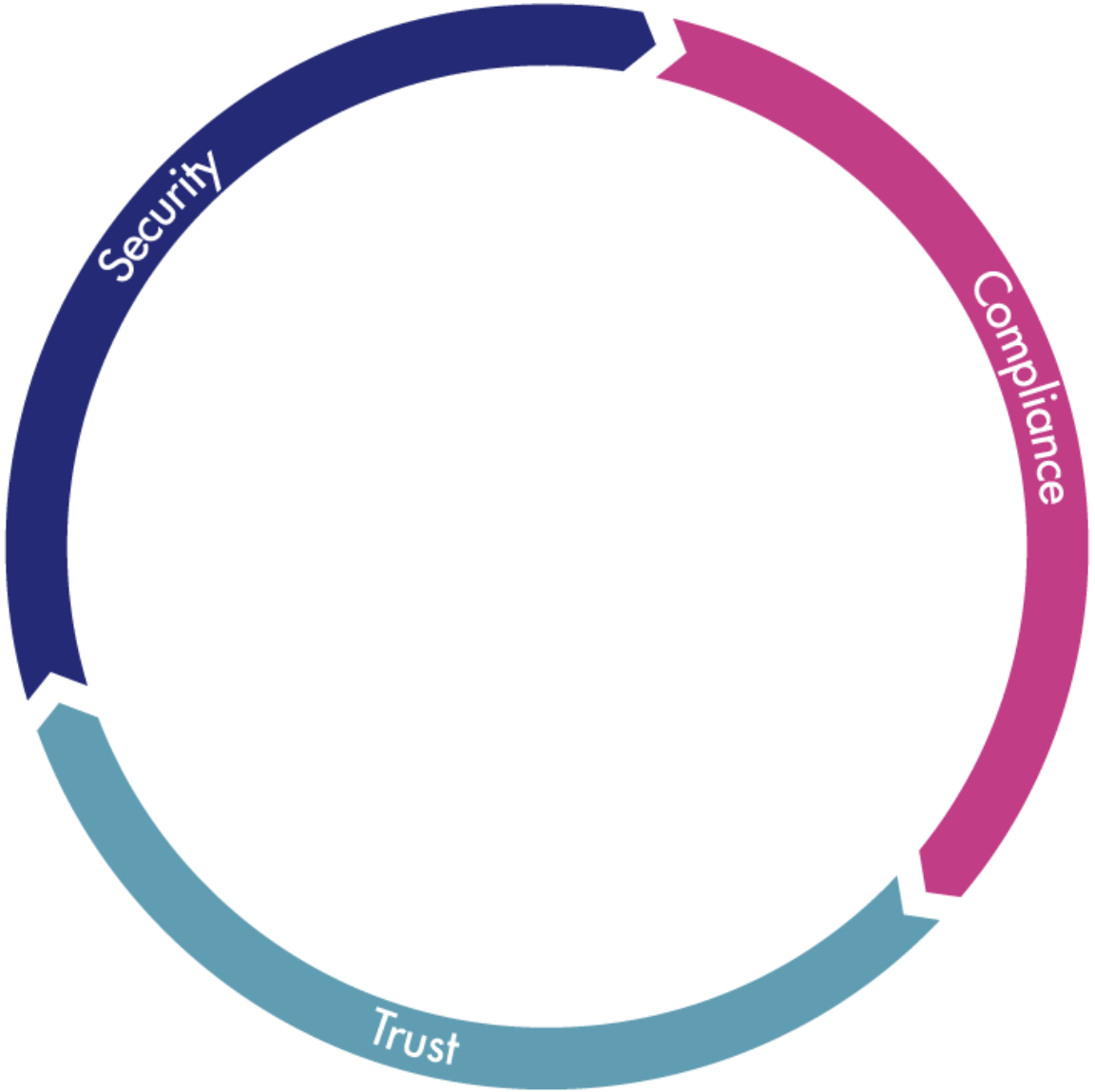
Theory & Practice

Nouredine TAMANI – nouredine.tamani@isep.fr

Objectives

- The role of databases in an IS
- Basic architecture of a relational database management system (RDBMS)
- Vulnerabilities raised by aggregation and inference in databases
- Vulnerabilities & Security in NoSQL Databases
- Identify threats to data storage systems

Objectives



Databases in IS

- Databases contain information essential to business functions:
  - Customer's contact details,
  - Order tracking data,
  - Human resources and benefits information,
  - Sensitive trade secrets, etc.
- Databases contain personal information:
  - Identification: SSN, names, IDs, etc.
  - Credit card usage activities, travel habits, purchases, phone records, etc.
  - CNIL authorization to create a DB containing personal information in France
  - In Europe, GDPR rules must be respected

Databases in IS: security levels

- Ensuring that adequate security controls exist:
  - Unauthorized access,
  - Falsification,
  - Destruction.
- Security at 3 levels: physical, engine and scheme
- Other database risks and vulnerabilities:
  - Poly instantiation,
  - ODBC (Open Database Connectivity),
  - Aggregation,
  - Inference and data mining.

1. Security mechanisms in DBMS

RDBMS

ODBC

Database Management System Architecture: 3 models

- Relational Database Management Systems (RDBMS)
  - Most widespread architecture
- Hierarchical and distributed architecture
  - A hierarchical data model combines records and linked fields in a logical tree
  - The distributed data model has data stored in multiple logically connected databases
- NoSQL database architecture

Relational databases

- A relational database is a set of two-dimensional tables composed of rows and columns
- Row and column structure provides one-to-one data mapping relationships
- Table (or Relation): the main building block of the relational database
- Each table contains a set of records
- Example: a 'sales' database might contain:
  - **Customer** table: contact information for all the clients
  - **Product** table: information about the goods the store sales
  - **Purchase** table: orders placed by each customer

Purchase table

Transaction ID	Customer ID	Product ID	Purchange date
1112	24221	8977	03-22-2010
1113	24222	8978	03-22-2010
1114	24223	8979	03-22-2010

Customer ID	Customer	Address
24221	Bob	123 East street
24222	Alice	223 Main street
24223	Martha	465 North street

Customer table

Product ID	Name	Price
8977	Banana	.79
8978	TV	400
8979	Watch	50

Product table

Relational databases

- Example: The following **Customer** table has
  - a cardinality of 3 (= number lines, rows, tuples, or records)
  - a degree of 8 (= number of columns).

Company ID	Company Name	Address	City	State	ZIP Code	Telephone	Sales Rep
1	Acme Widgets	234 Main Street	Columbia	MD	21040	(301) 555-1212	14
2	Abrams Consulting	1024 Sample Street	Miami	FL	33131	(305) 555-1995	14
3	Dome Widgets	913 Sorin Street	South Bend	IN	46556	(574) 555-5863	26

FIGURE Tables Customers extracted from a relational database

- It is common for the cardinality of a table to change during normal activities:
  - Adding/deletion
- The degree of a table is stable over time (does not change frequently)
  - Typically requires the intervention of the database administrator

Relational databases

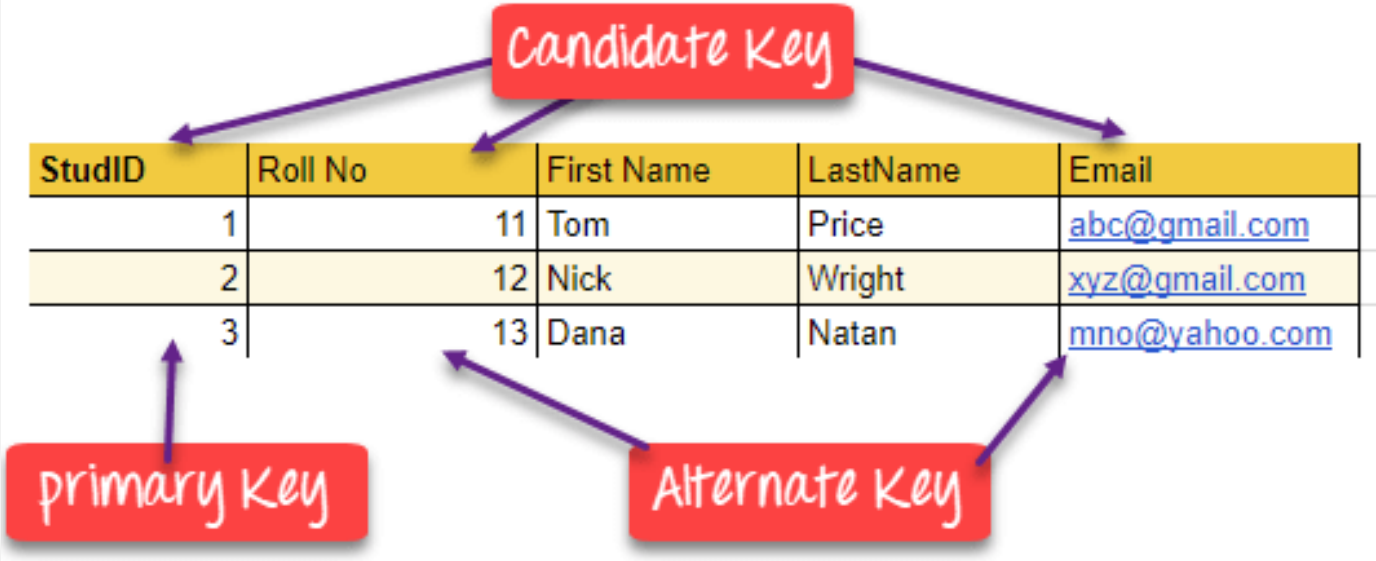
- Relationships between tables are defined to identify linked records
- Example: there is a link between the table 'Customer' and the table 'Sales Rep’
  - Each customer is assigned to a sales representative and each sales representative is assigned to one or more customers.

Company ID	Company Name	Address	City	State	ZIP Code	Telephone	Sales Rep
1	Acme Widgets	234 Main Street	Columbia	MD	21040	(301) 555-1212	14
2	Abrams Consulting	1024 Sample Street	Miami	FL	33131	(305) 555-1995	14
3	Dome Widgets	913 Sorin Street	South Bend	IN	46556	(574) 555-5863	26

- Records are identified using various keys:
  - A key is a subset of fields that uniquely identifies records,
  - Keys are also used to join tables when it is necessary to cross-reference information.

Relational databases: 4 Key Types

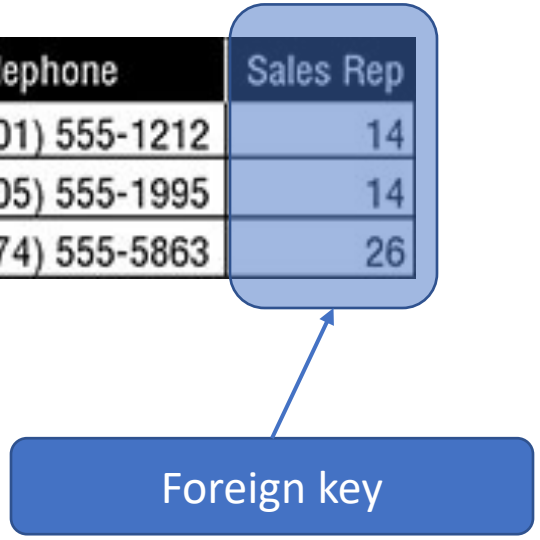
- **Candidate** keys:
  - A candidate key ensures that no two rows in the same table contain the same values
  - Each table can have one or more candidate keys
- **Primary** keys
  - A primary key is selected among all the candidate keys for a table: it is a design choice
  - Each table has only one primary key
  - The RDBMS applies the uniqueness of primary keys
  - Integrity constraints
- **Alternate** keys
  - A candidate key which is selected as a primary key



Relational databases: 3 Key Types

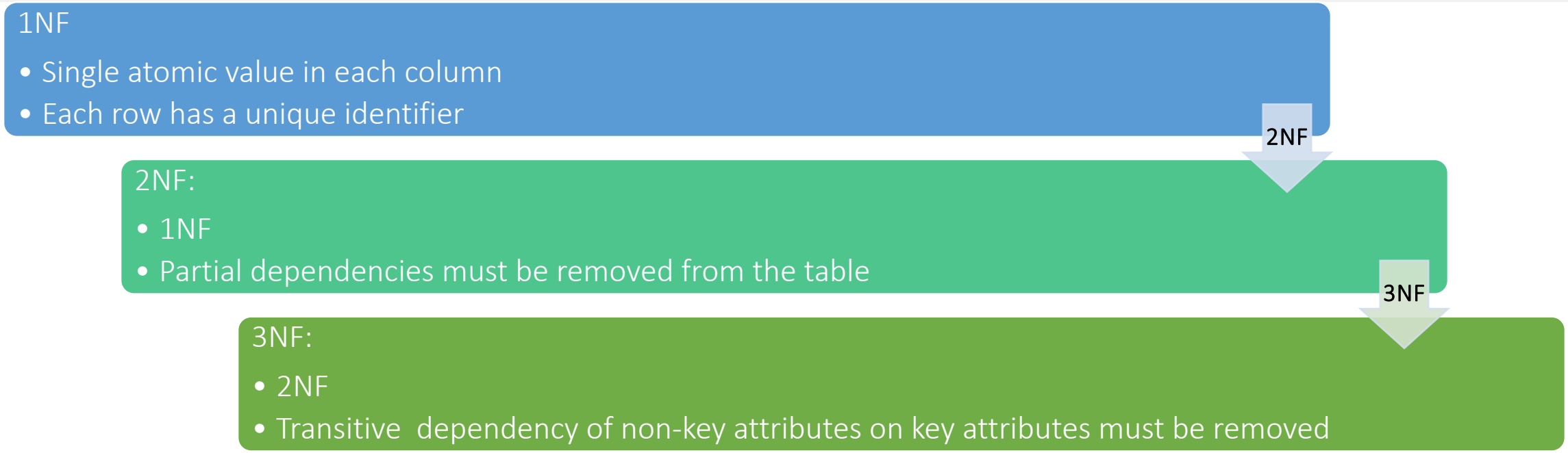
- **Foreign** keys
  - A foreign key is used to implement relationships between two tables, aka **referential integrity**
  - **Referential integrity**: if a table contains a foreign key, it must match an existing primary key in the other table in the relationship
  - It ensures that no record/tuple/line contains a reference to a primary key of a non-existent record/tuple/line
  - 'Sales Rep' field is a foreign key referencing the primary key of the 'Sales Reps' table

Company ID	Company Name	Address	City	State	ZIP Code	Telephone	Sales Rep
1	Acme Widgets	234 Main Street	Columbia	MD	21040	(301) 555-1212	14
2	Abrams Consulting	1024 Sample Street	Miami	FL	33131	(305) 555-1995	14
3	Dome Widgets	913 Sorin Street	South Bend	IN	46556	(574) 555-5863	26



Database normalization

- Database developers strive to create well-organized and efficient databases
- Normal forms:
  - Database definition rules that ensure data consistency and reduce redundancy in tables
  - The process of conforming a database table to normal shapes is called **normalization**
- The 3 most important normal forms are:

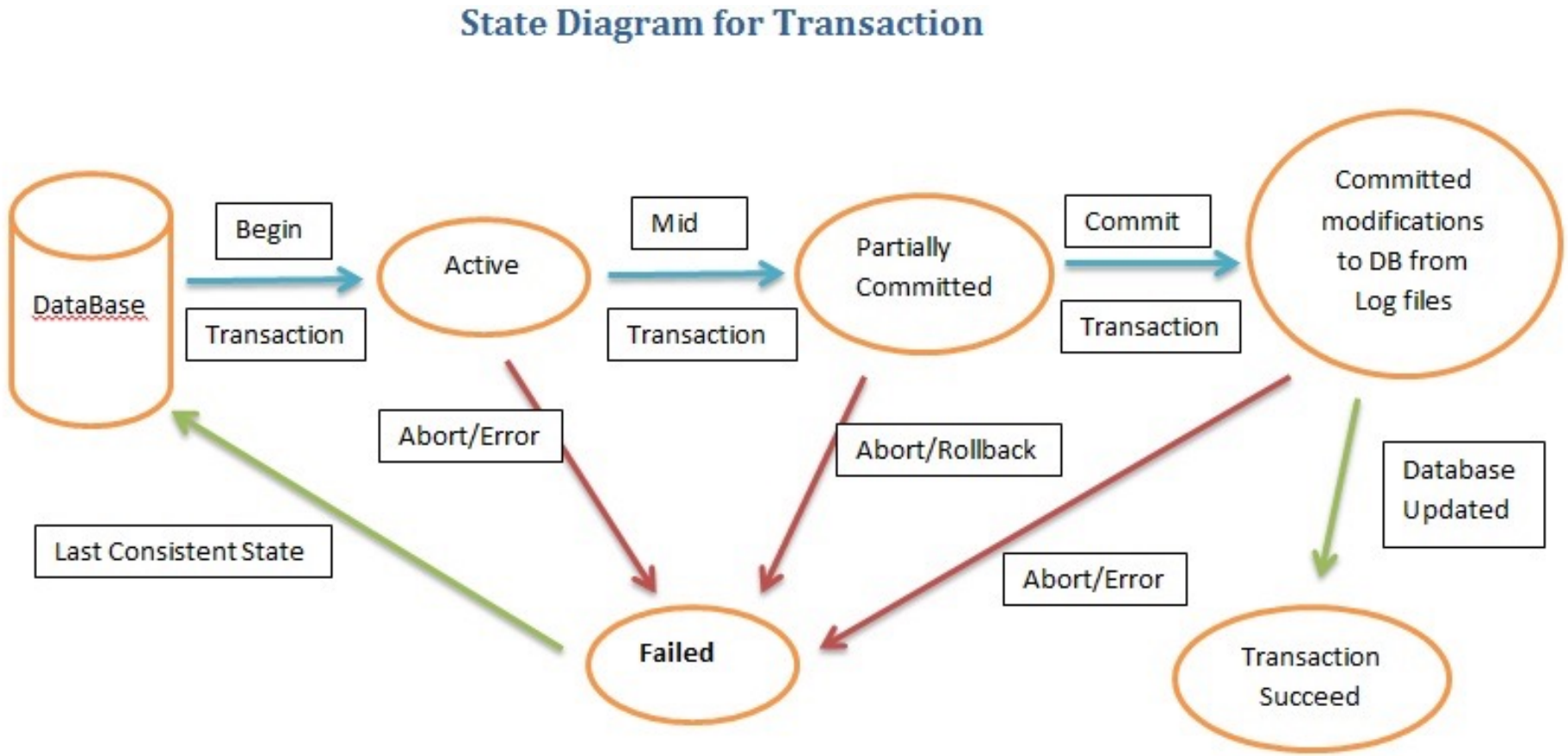


Relational databases: Query Language

- All relational databases use a standard language, Structured Query Language (SQL)
  - A consistent interface for DB creation, storage, retrieval, data modification, and administrative control of the DBMS
- SQL is divided into two distinct sub-languages:
  - DDL (Data Definition Language) allows the creation and modification of a database (known as a schema),
  - DML (Data Manipulation Language) allows users to interact with the data contained in this schema.
- Each DBMS provider implements a slightly different version of SQL
  - (like Microsoft's Transact-SQL and Oracle's PL/SQL)
- Graphical database interfaces extend the standard SQL interface for the DBMS with additional functions
- **Authorization Granularity:** The Main Security Feature of SQL
  - More detailed definition of permissions,
  - Definition of user access by table, row, column or even an individual cell in some cases.

Database transactions

- Relational databases support the explicit and implicit use of transactions to ensure data integrity
- Each transaction is a discrete set of SQL statements executed as a unit: total pass/fail
  - It is not possible for one part of a transaction to succeed while another part fails

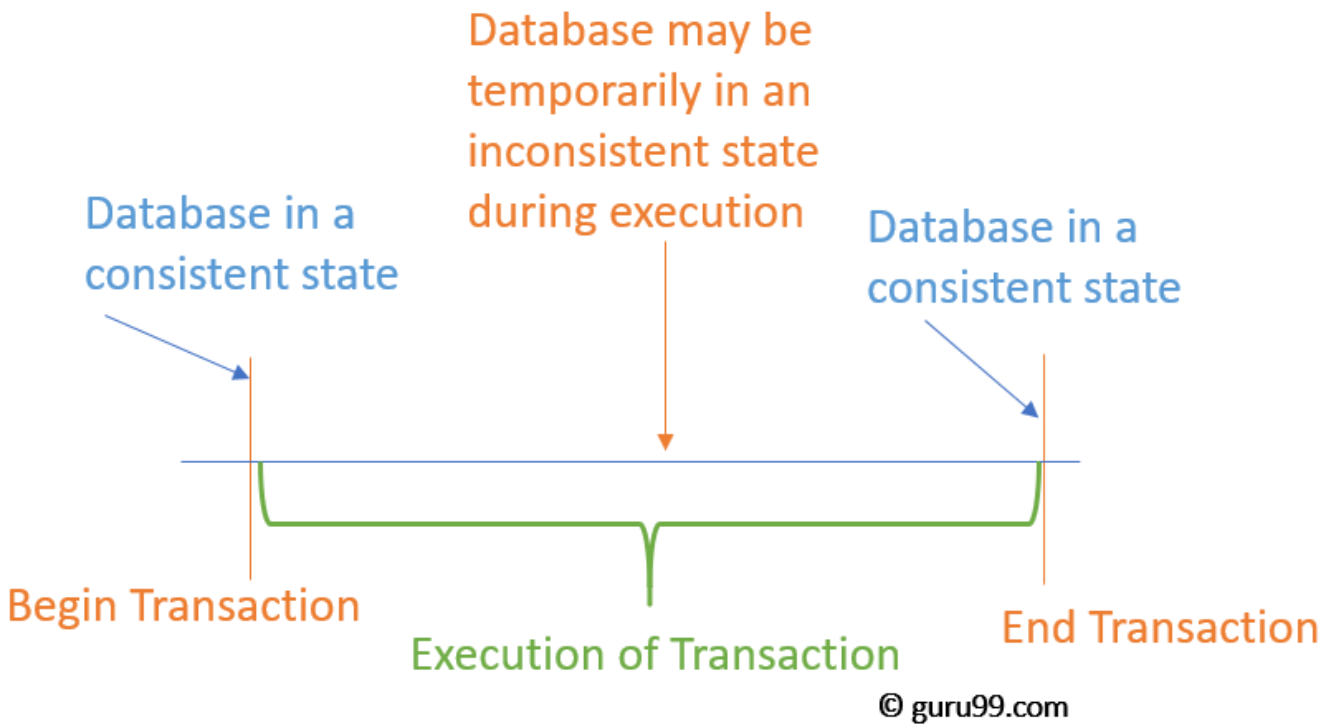


Database transactions: ACID

- When a transaction completes successfully, it is said to be committed to the database and cannot be rolled back
- Transaction Validation:
  - Can be explicit, using the SQL COMMIT command,
  - Can be implicit if the transaction is successfully reached.
- If a transaction needs to be aborted, it can be rolled back explicitly using the ROLLBACK command or implicitly on the event of a hardware or software failure
- When a transaction is restored, the database returns to the state it was in before the transaction began
- All database transactions have four required characteristics: **Atomicity, Consistency, Isolation, and Durability**

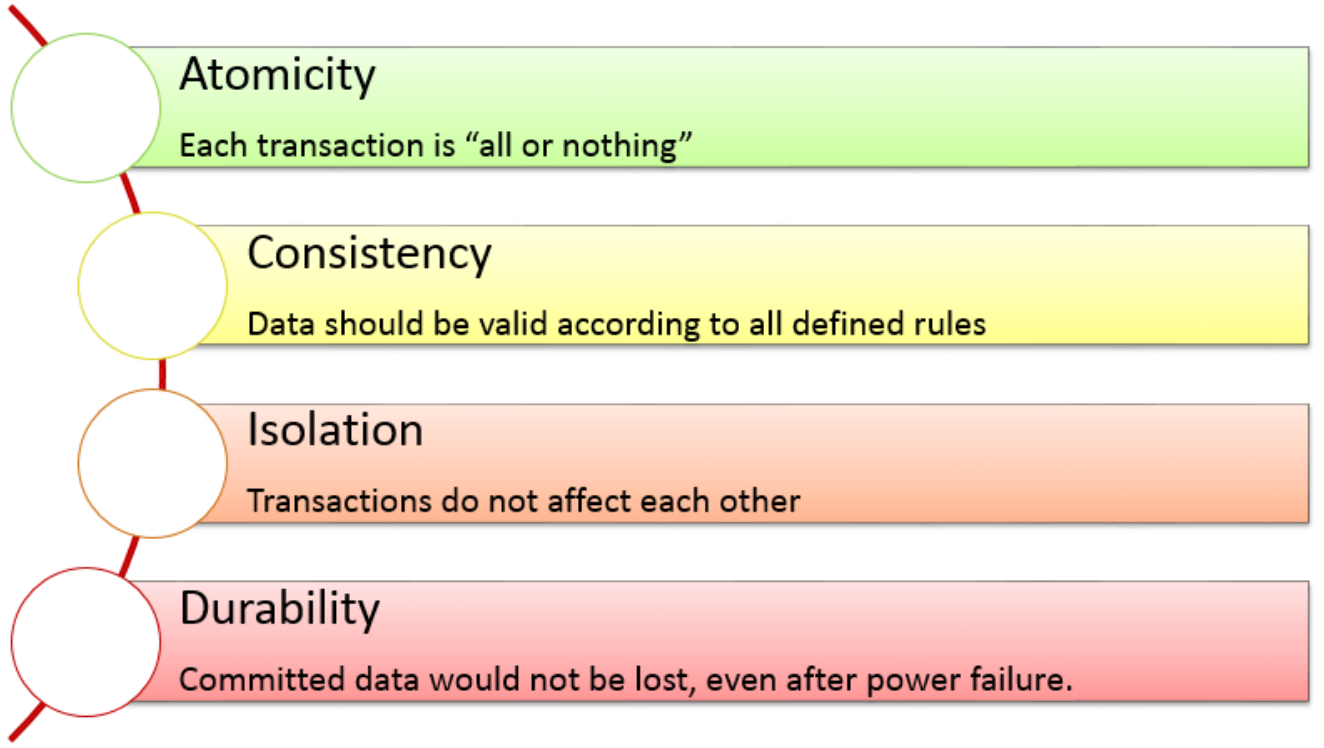
ACID Properties

- Atomicity:
  - A transaction must be an "all or nothing",
  - If part of the transaction fails, the entire transaction should be restored as if it never happened
- Coherence:
  - All transactions must begin in an environment that is consistent with all the rules in the database.
  - When the transaction is complete, the database must again be consistent with the defined rules,
  - No other transaction should be able to use inconsistent data generated during the execution of another transaction.



Atomicity Consistency Isolation Durability Properties

- Isolation:
  - Transactions are executed separately from each other,
  - If a DB receives 2 SQL transactions that modify the same data, one transaction completes before the other transaction is allowed to modify the same data.
- Durability:
  - Once validated in the database, the data must be committed,
  - Databases ensure durability using backup mechanisms, such as transaction logs.

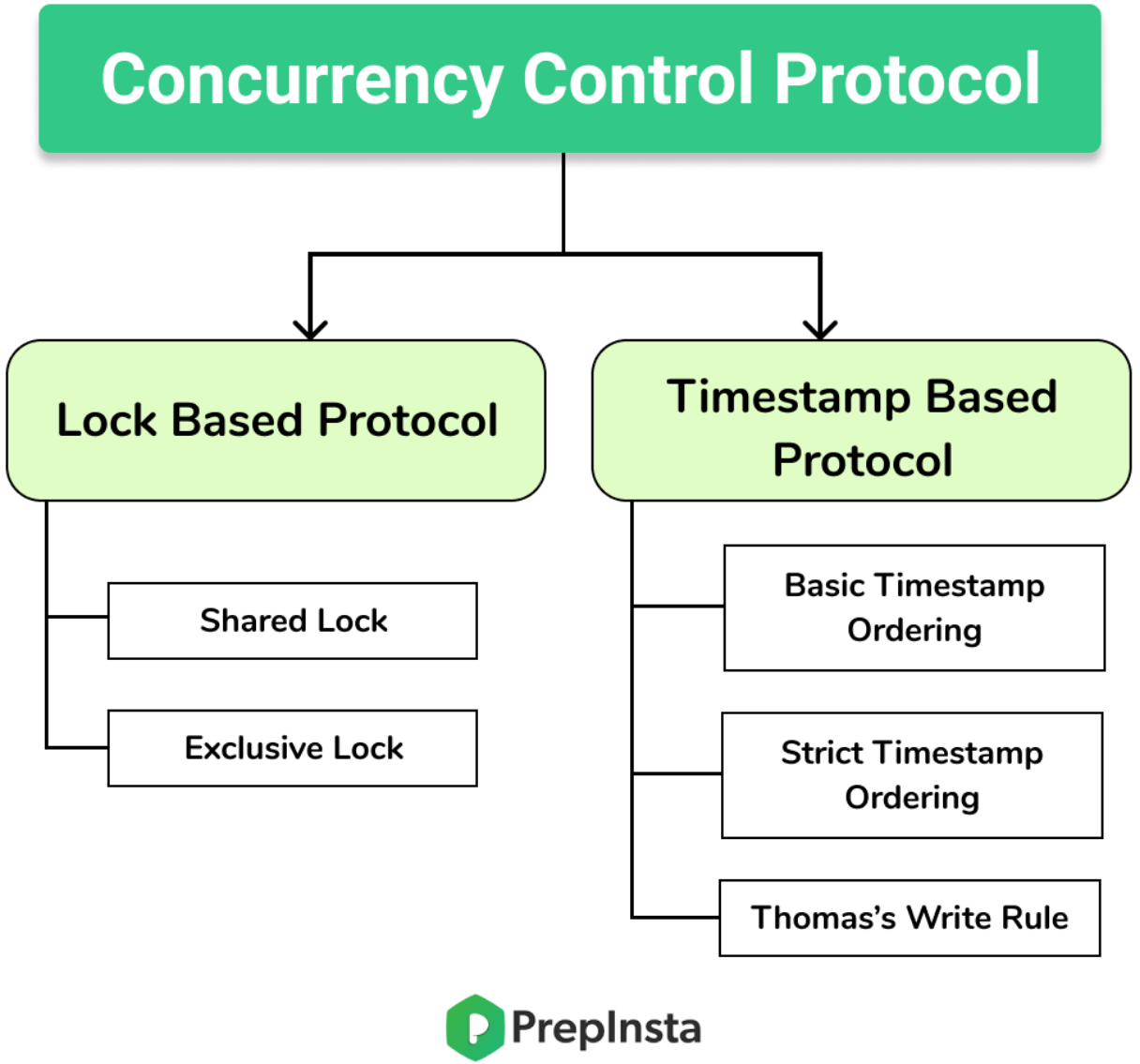


Concurrency

- Concurrency, or editing control, is a preventive security mechanism that ensures that:
  - the data stored in the database is always correct,
  - their integrity and availability are protected.
- Concurrency uses a "lock" feature to allow a user to make changes:
  - Deny other users access to views or make changes to data items at the same time,
  - Once the changes are made, an "unlock" feature restores other users' ability to access the data they need.
- In some cases, administrators use concurrency with audit mechanisms to track changes to documents and/or fields.

Concurrency

- When this recorded data is examined, concurrency becomes a detection control
- Databases that fail to implement concurrency correctly can suffer from the following issues:
  - Lost updates
  - Incorrect readings



Other security mechanisms: semantic integrity

- Semantic integrity ensures that user actions do not violate any structural rules:
  - All stored data types are in valid domain ranges,
  - Only logical (consistent) values exist,
  - The system complies with all uniqueness constraints.
- Administrators can use timestamps to maintain data integrity and availability:
  - Timestamps often appear in distributed database systems
  - All distributed or replicated change transactions are applied to all members of the DB in the correct chronological order

Other security mechanisms: Access controls

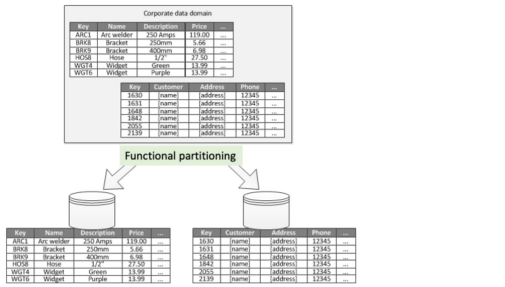
- Content-dependent access control is an example of granular object control
  - Content-dependent access control is based on the content or payload of the object being accessed
  - It increases processing overhead: because decisions must be made object by object,
  - Another form of granular control is cell deletion: Cell deletion: hiding individual fields or database cells
- Context-dependent access control evaluates the whole data context to make access control decisions
  - The key factor in context-dependent access control is how each object or field relates to the activity

Other security mechanisms: Partitioning - Poly-instantiation

- Database partitioning is the process of splitting a single database into several parts:
  - Each with a unique and distinct security level or content type
- Poly-instantiation occurs when two or more rows in the same RDB table appear to have identical primary key elements:
  - But contain different data for use at different classification levels
- It is often used as a defense against certain types of inference attacks

Different types of database partitioning

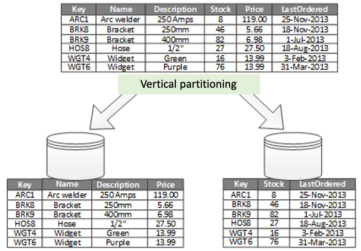
Different types of database partitioning: Functional Partitioning



23/05/2023 Database security 25

Functional partitioning

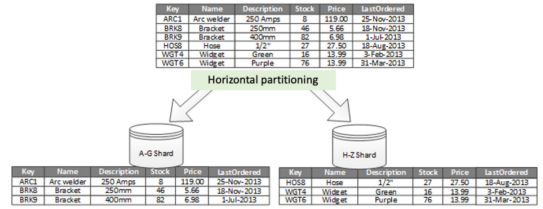
Different types of database partitioning: Vertical Partitioning



23/05/2023 Database security 26

Vertical partitioning

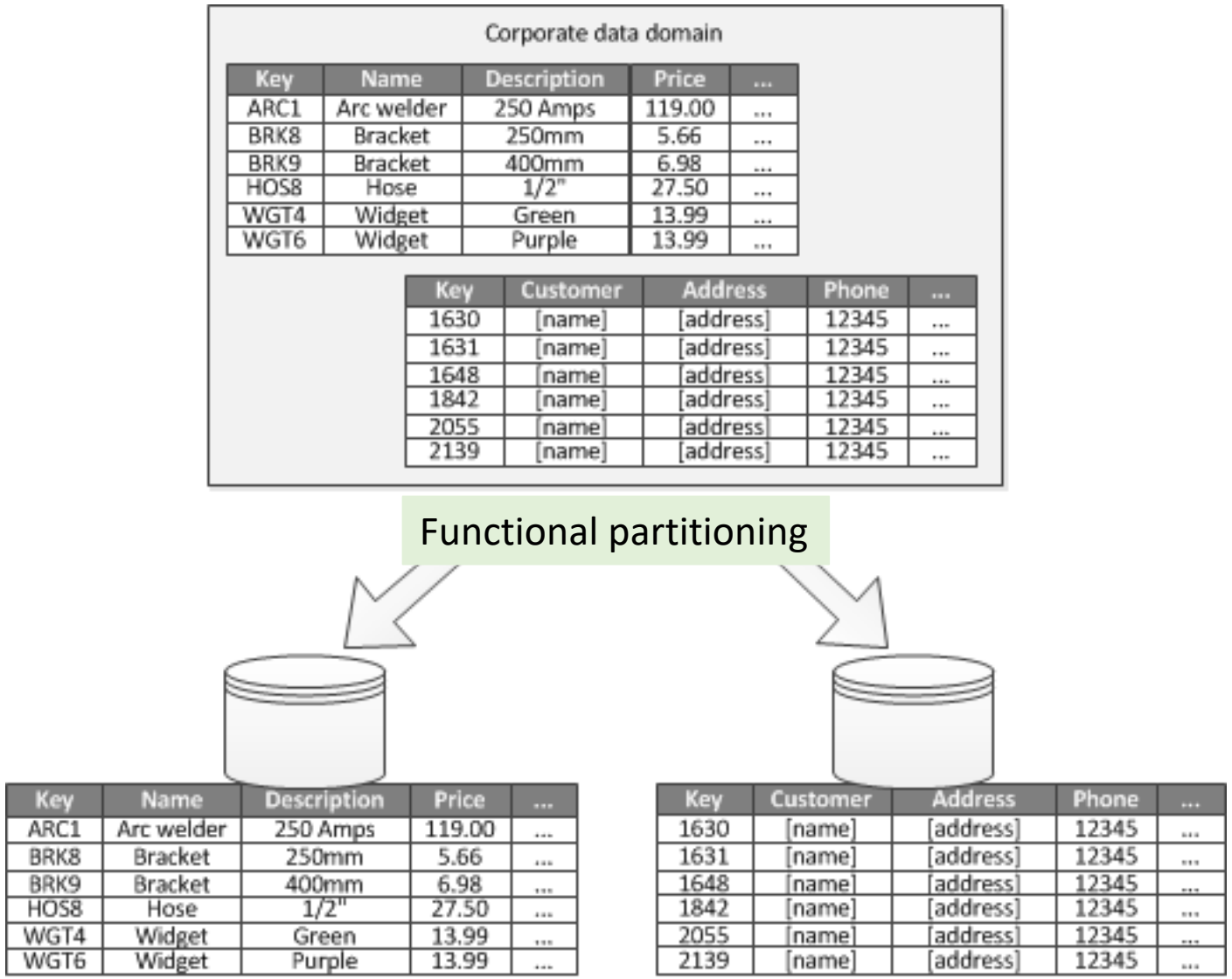
Different types of database partitioning: Horizontal Partitioning



23/05/2023 Database security 27

Horizontal partitioning

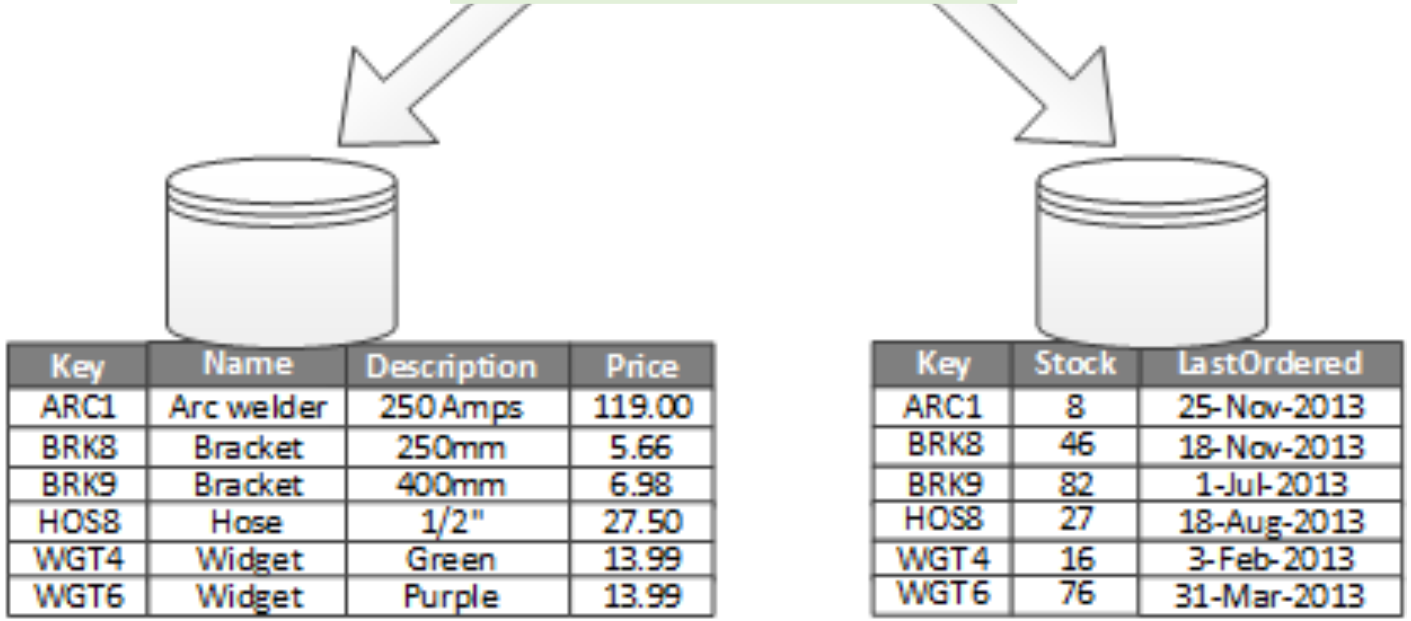
Different types of database partitioning: Functional Partitioning



Different types of database partitioning: Vertical Partitioning

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Vertical partitioning



Key	Name	Description	Price
ARC1	Arc welder	250Amps	119.00
BRK8	Bracket	250mm	5.66
BRK9	Bracket	400mm	6.98
HOS8	Hose	1/2"	27.50
WGT4	Widget	Green	13.99
WGT6	Widget	Purple	13.99

Key	Stock	LastOrdered
ARC1	8	25-Nov-2013
BRK8	46	18-Nov-2013
BRK9	82	1-Jul-2013
HOS8	27	18-Aug-2013
WGT4	16	3-Feb-2013
WGT6	76	31-Mar-2013

Different types of database partitioning: Horizontal Partitioning

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

Horizontal partitioning

Key	Name	Description	Stock	Price	LastOrdered
ARC1	Arc welder	250 Amps	8	119.00	25-Nov-2013
BRK8	Bracket	250mm	46	5.66	18-Nov-2013
BRK9	Bracket	400mm	82	6.98	1-Jul-2013

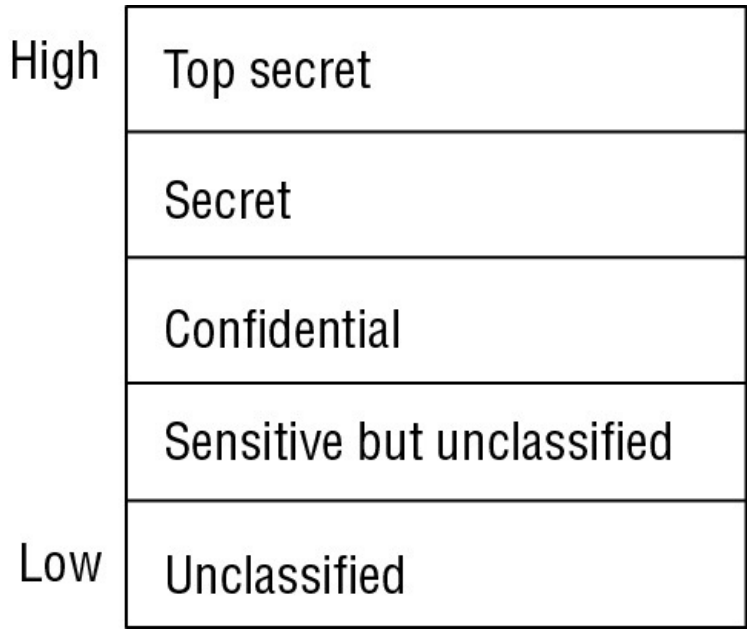
Key	Name	Description	Stock	Price	LastOrdered
HOS8	Hose	1/2"	27	27.50	18-Aug-2013
WGT4	Widget	Green	16	13.99	3-Feb-2013
WGT6	Widget	Purple	76	13.99	31-Mar-2013

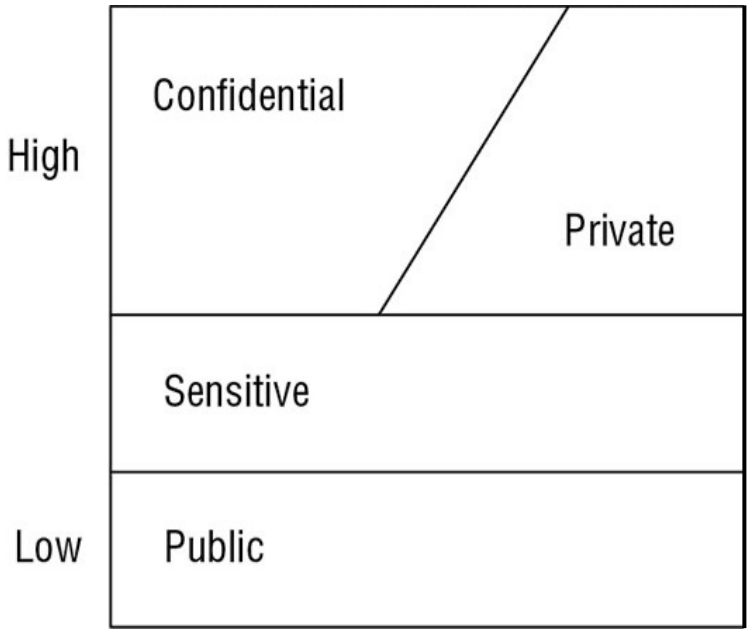
Other security mechanisms: Poly-instantiation

- Example of poly-instantiation:
  - Consider a BDD table containing the location of various warships on patrol,
  - Normally, it contains the exact position of each vessel stored at the "secret" classification level,
  - But one ship in particular, the USS UpToNoGood, is on an undercover mission in a top-secret location,
  - Military commanders do not want anyone to know that the ship has deviated from its normal patrol,
  - If the DB administrators change the classification of UpToNoGood's location to top secret, a user with secret permission would know that something unusual was happening when they could not query the location of the ship.
- If poly-instantiation is used, two records can be inserted into the table:
  - The first, classified at the top-secret level, would reflect the actual location of the ship,
  - The second is at the secret level would indicate that the ship was under routine patrol.

Other security mechanisms: Poly-instantiation

- Multi-level database security
- Many organizations use data classification schemes
- Prerequisite for setting and enforcing access control restrictions





Multi-level database security

- Multi-level security databases contain information at different classification levels
- They must verify the labels assigned to users
- In response to user requests, the DB provides only the appropriate information
- This concept becomes a little more complicated when examining the security of a database:
  - When designing the database: separate data with different security requirements.
  - Database contamination: when mixing data with different classification levels and/or need-to-know requirements

Other safety mechanisms: noise or disturbance

- Noise and disturbance:
  - Administrators can insert false or misleading data into a DBMS to redirect or thwart information privacy attacks
  - Extreme caution should be applied when using this technique
  - It must be ensured that the noise inserted into the database does not affect the company's operations

Open Database Connectivity

- ODBC (Open Database Connectivity) is a DB API that allows applications to communicate with different types of databases,
- ODBC acts as a proxy between applications and back-end database drivers (Figure 2),
- Decoupling of business aspects from given aspects in development,
- JDBC (Java Data Base Connectivity) is a Java implementation of ODBC,
- There are many vulnerabilities related to ODBC-JDBC.

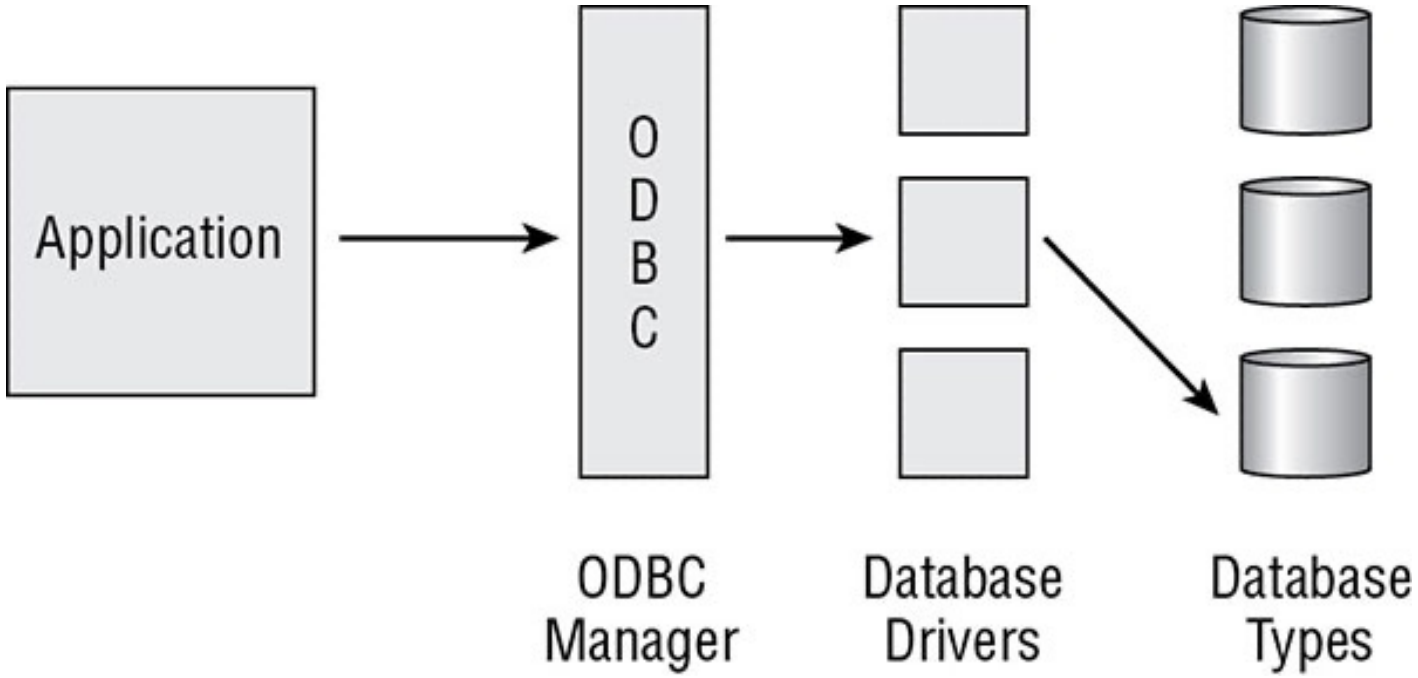


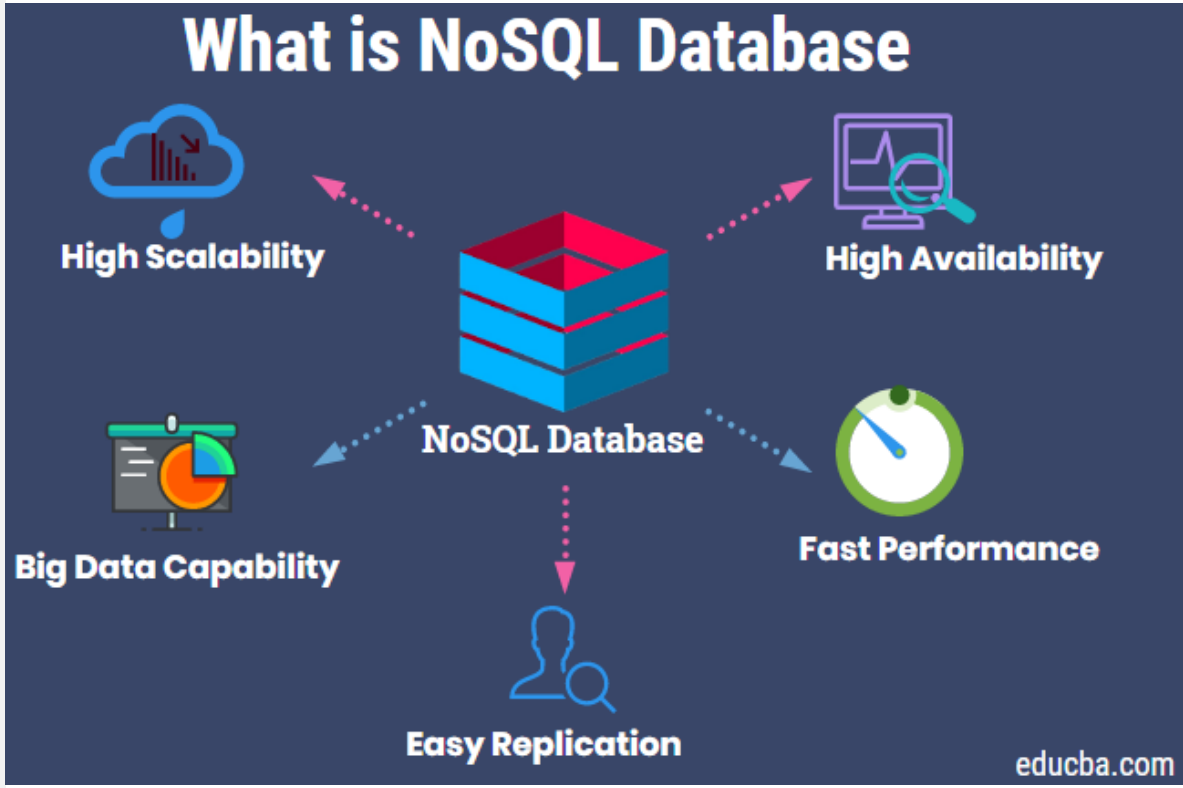
FIGURE 2 ODBC as the interface between applications and the DBMS

2. NoSQL Database Security

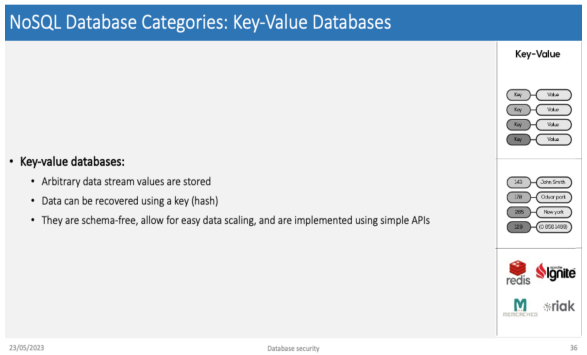
- NoSQL DB definition
- Vulnerabilities
- Best practices

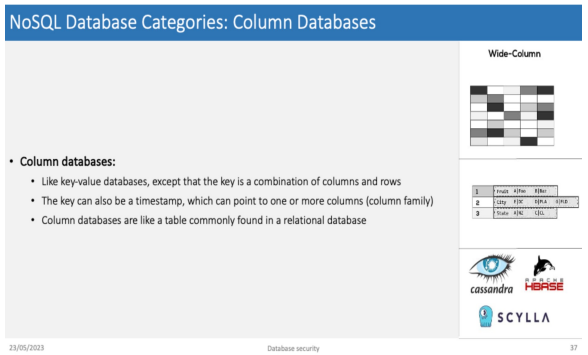
NoSQL database

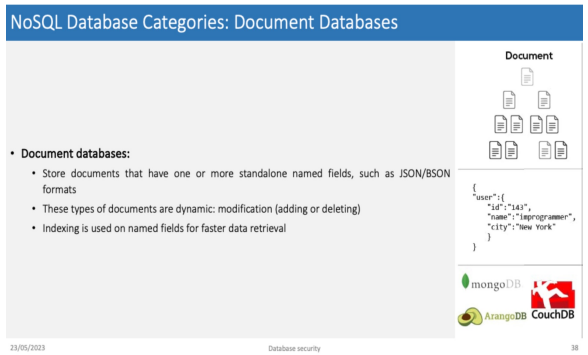
- NoSQL databases are non-relational distributed databases
- NoSQL DB is an alternative for applications:
  - requiring increased performance (reduce latency),
  - their data does not fit into the table format: graphs, documents, key-value, etc.
- Big Data requirements: Volume, Velocity, Variety
- They support huge data storage on various storage clusters
- Widely used in Web 2.0 applications: Amazon, Google, Netflix, eBay, etc.
- Security models used by NoSQL BDs can be intriguing

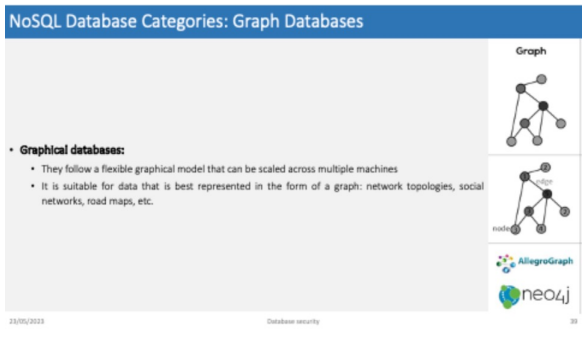


NoSQL Database Categories

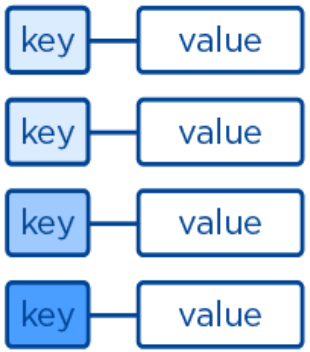




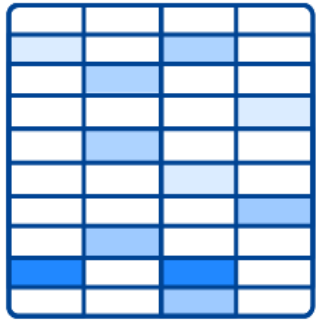




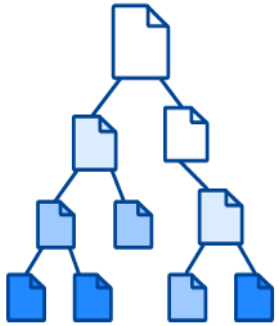
Key-Value



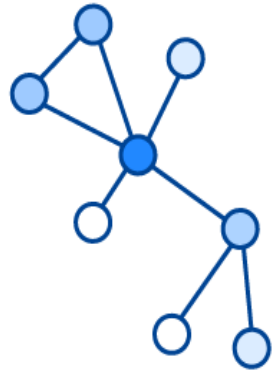
Wide-column



Document

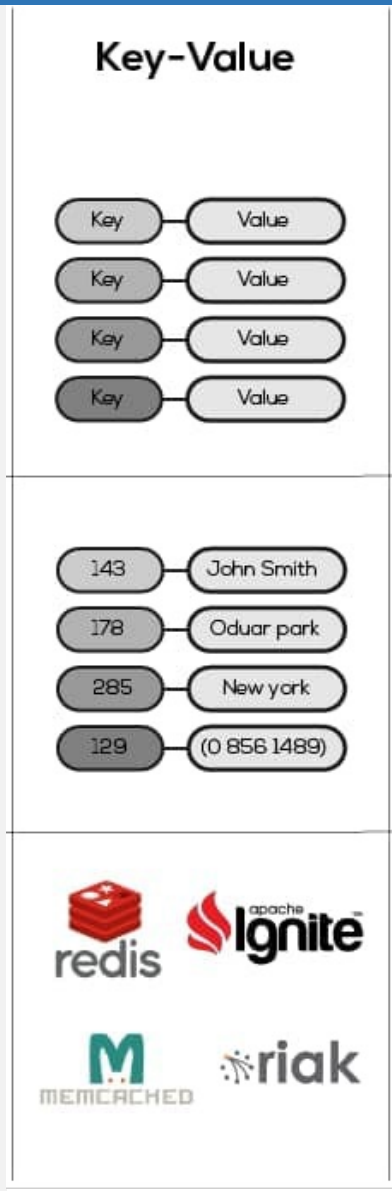


Graph



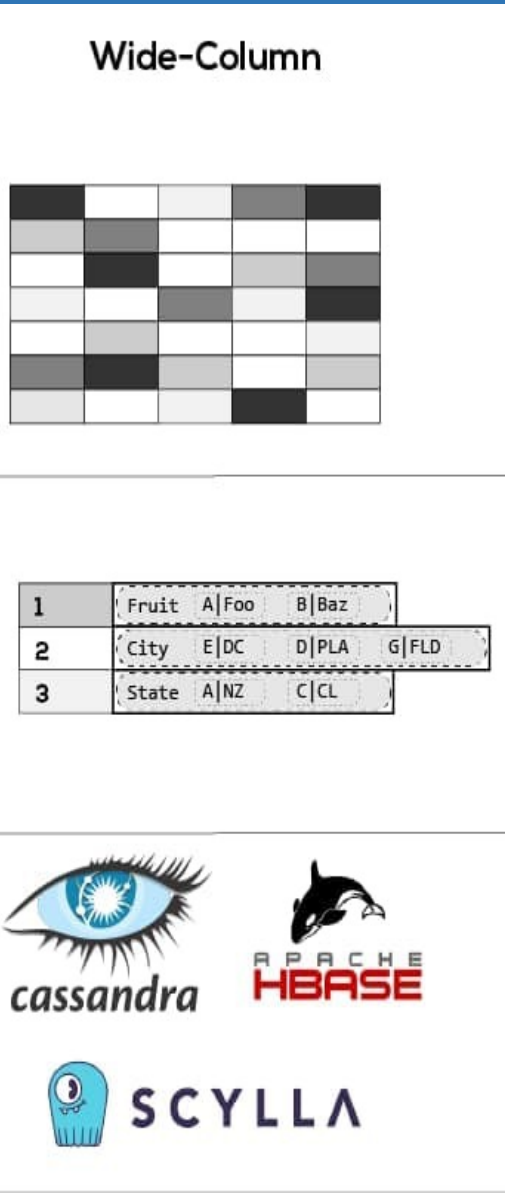
NoSQL Database Categories: Key-Value Databases

- Key-value databases:
  - Arbitrary data stream values are stored
  - Data can be recovered using a key (hash)
  - They are schema-free, allow for easy data scaling, and are implemented using simple APIs



NoSQL Database Categories: Column Databases

- Column databases:
  - Like key-value databases, except that the key is a combination of columns and rows
  - The key can also be a timestamp, which can point to one or more columns (column family)
  - Column databases are like a table commonly found in a relational database



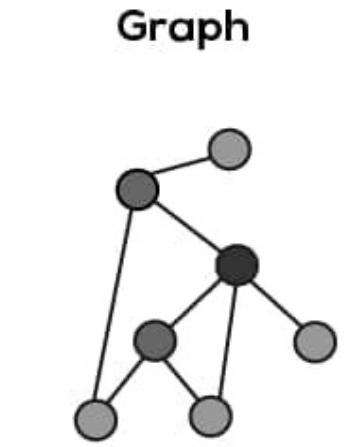
NoSQL Database Categories: Document Databases

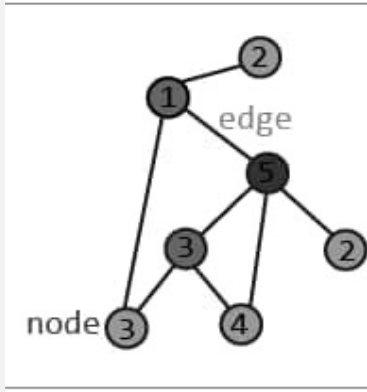
- Document databases:
  - Store documents that have one or more standalone named fields, such as JSON/BSON formats
  - These types of documents are dynamic: modification (adding or deleting)
  - Indexing is used on named fields for faster data retrieval


```
{
"user":{
    "id":"143",
    "name":"improgrammer",
    "city":"New York"
    }
}
```

NoSQL Database Categories: Graph Databases

- Graphical databases:
  - They follow a flexible graphical model that can be scaled across multiple machines
  - It is suitable for data that is best represented in the form of a graph: network topologies, social networks, road maps, etc.





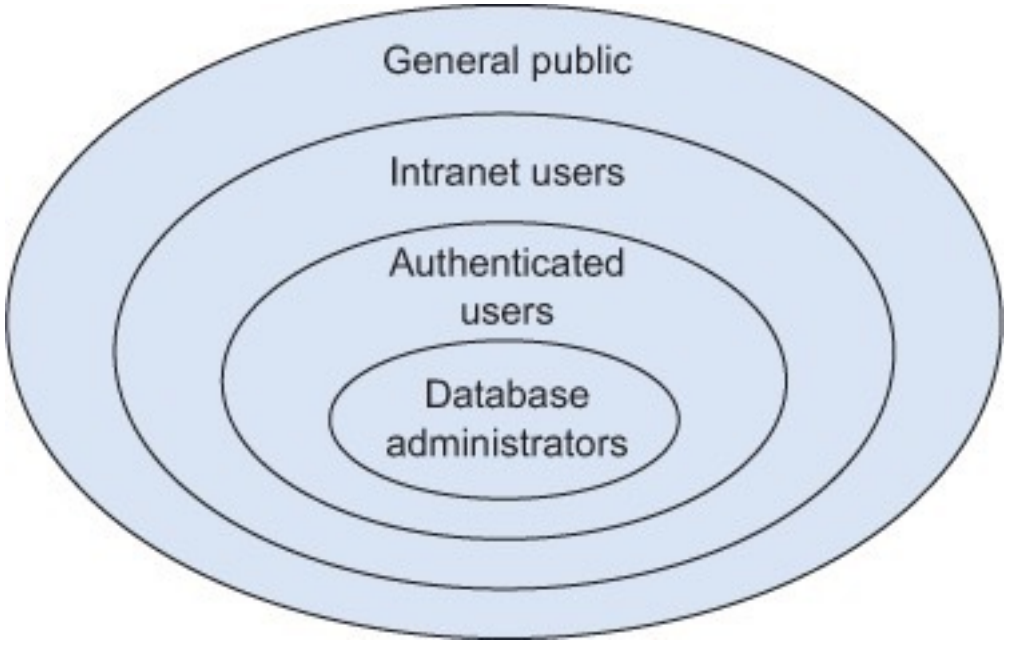
AllegroGraph

neo4j

NoSQL Database Security Issues

Security is still a concern for NoSQL DB users

- Less effort was put into safety during the design phase: NoSQL does not provide built-in security features
- Developers need to integrate security-related solutions into middleware
- NoSQL database clustering functionality adds challenges in implementing security practices
- The authentication mechanism in a NoSQL database is applied at the local node, but fails on all servers
- Kerberos can be used to authenticate clients and data nodes:
  - Malicious clients can gain unauthorized access by duplicating the Kerberos ticket.

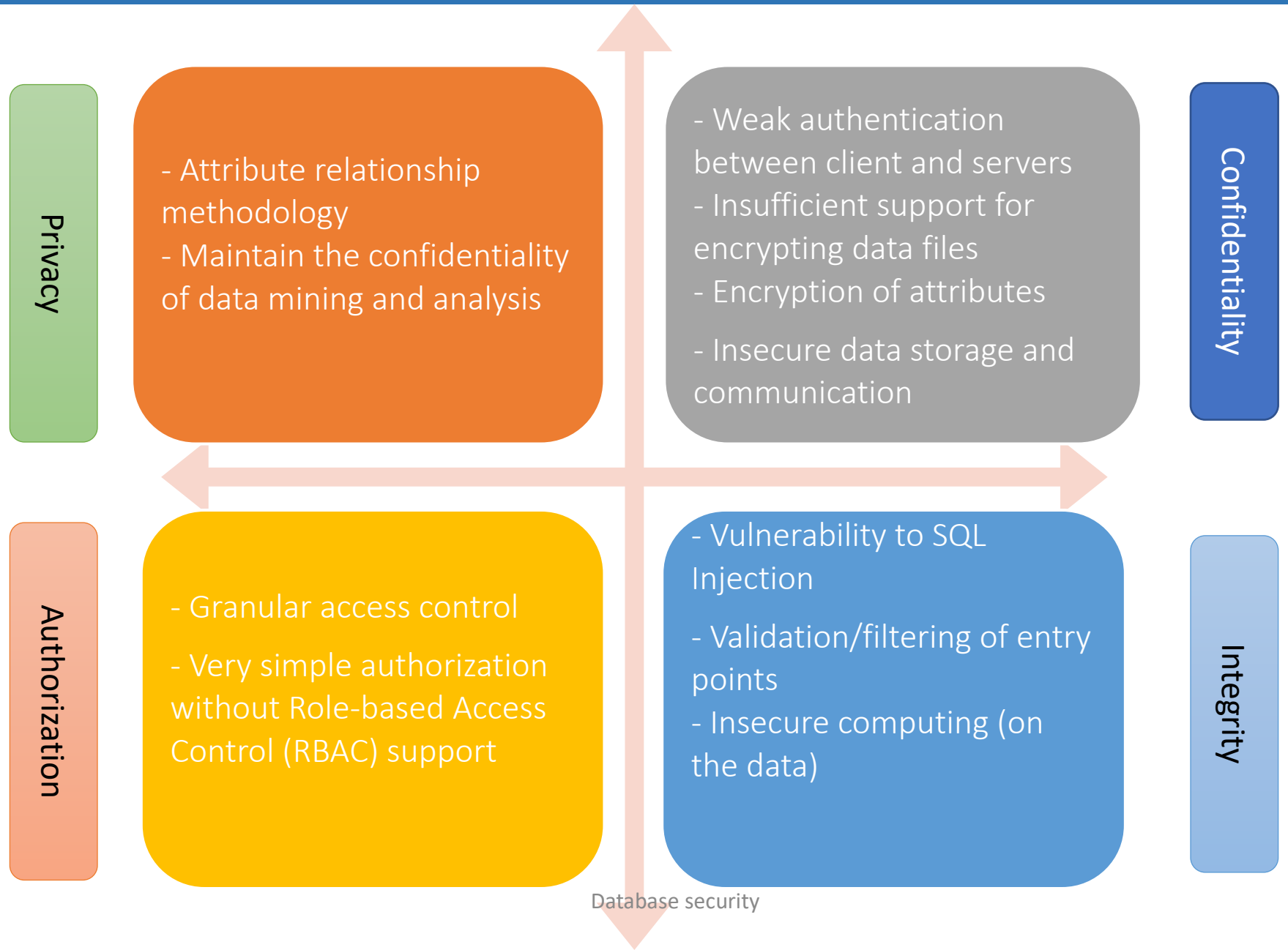


https://livebook.manning.com/book/making-sense-of-nosql/chapter-11/14

NoSQL Database Security: Facts

- No two NoSQL database products are the same
  - They are designed to meet the explicit requirements of certain cloud-based applications
- NoSQL databases lack various security features such as authentication, authorization, and integrity,
  - Sensitive data is more secure in traditional RDBMS
- RDBMS have adopted highly reliable mechanisms to provide security services:
  - They also face many security threats: cross-site scripting, SQL injections, root kits, etc.
- NoSQL inherited the security issues of traditional RDBMS, in addition to its own issues
- Data moves from one node to another, which leads to data sharing and increases the risk of theft
- NoSQL databases lack confidentiality, privacy and integrity

NoSQL Database Security: Major Security Concerns



NoSQL Database Security: Major Security Concerns

- The granular access controls required to separate user roles and responsibilities are not provided by many systems
- Transaction logs:
  - To prevent unauthorized access to data
  - To maintain secure data availability, data storage
- Protecting Valuable Information:
  - Attribute relationship methodology
  - The attribute with higher relevance is more important than the other attributes
- Attribute-based encryption:
  - Data Access Control Method for NoSQL Databases
  - It allows data owners to encrypt data as part of an access policy
  - Only users authorized to access the data can decrypt it

NoSQL Database Security: Major Security Concerns

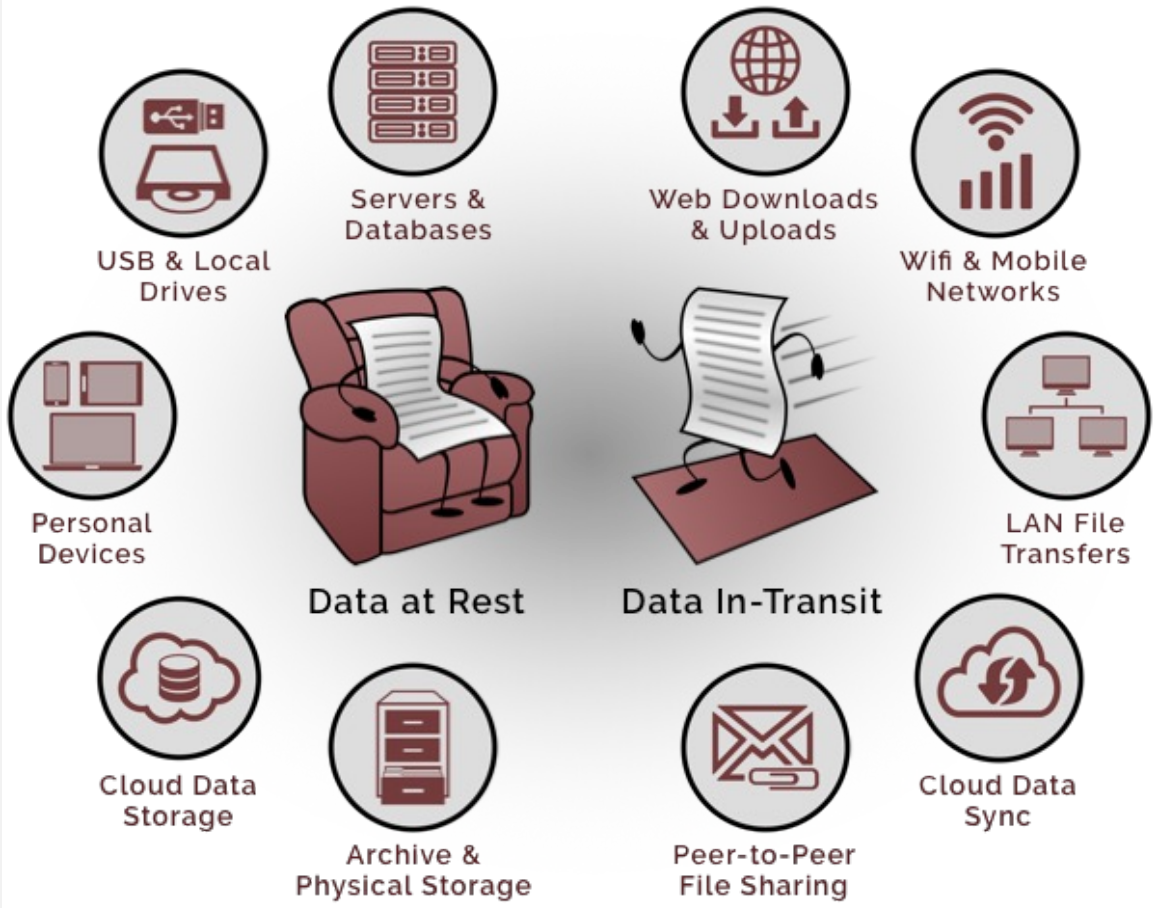
- Integrity Protection:
  - Security requirement to ensure data protection against unauthorized modification
  - Difficult to protect data integrity due to the heterogeneous nature of NoSQL databases
  - NoSQL databases do not have a schema, so permissions on a table, column, or row cannot be separated
- This can lead to multiple copies of the same data, making it difficult to ensure the consistency of the data
- Maintaining transactional integrity is therefore also very difficult in NoSQL databases
- NoSQL databases can never be used for financial transactions

Fine-grained authorization and access control

- NoSQL databases have no schema and store heterogeneous data together
  - It is difficult to implement authorization on a table as a whole
- Fine-grained permission enables object-level security
- Object-level security:
  - Row-level security
  - Field-level security
- Fine-grained access control is not allowed due to the schema-free nature of NoSQL databases
- Most NoSQL databases allow permission at the column family level
- In the NoSQL database, data is grouped according to its security level
  - Data can be classified as confidential, secret, or even not classified at all
- Role-based access control is difficult to implement because the NoSQL database has a schema-free structure

Protection of data at rest and in motion (transit)

- **Data at rest:** data written to a storage medium
- **Data in transit:** data exchanged during a communication
- In the case of NoSQL DBs, moving data is classified into two categories:
  - Client-node communication
  - Communication between nodes
- Most NoSQL DBs do not use any techniques to protect data at rest
- Only a few NoSQL DBs provide encryption mechanisms



https://spanning.com/blog/saas-data-encryption-data-at-rest-data-in-transit/saas-data-encryption-data-at-rest-vs-data-in-transit/

Data at rest

- Cassandra uses Transparent Data Encryption (TDE) to protect data at rest
- In Cassandra databases, encryption certificates are stored locally
  → a secure file system is required to implement TDE
- The Validation log of the Cassandra database is not encrypted, which also results in a security breach
- MongoDB does not provide any method to encrypt the data file
- Data files can be encrypted at the application level before writing the data to the database

Data in transit

- Client-node communications:
  - They are not encrypted in Cassandra
  - Encryption is done by generating valid server certificates at the SSL layer
  - MongoDB does not support SSL client mode communication
  - To encrypt data in client-SSL node communication mode, MongoDB must be recompiled by configuring SSL communication
- Communications between nodes:
  - Cassandra does not support encrypted communications between nodes
  - Using the Cassandra.yaml file, server encryption options can be changed to configure SSL between nodes
  - MongoDB does not support internode communication at all

User data Privacy

- Maintaining the confidentiality of sensitive information is the primary concern of any database administrator
- If a single node is compromised, malicious data spreads to the entire system
  - There is no centralized security management
- The distributed nature of the database can also lead to security breaches
- The main privacy issues are related to:
  - **Unauthorized access,**
  - **Deletion of data,**
  - **Backups,**
  - **Isolation failed,**
  - **Inadequate monitoring,**
  - **Audit.**

Lack of expertise and buggy applications

- NoSQL is an emerging technology → very few experts holistically understand its security aspects
- Industry needs security experts to take care of NoSQL databases
- Lack of a standard security model for NoSQL databases leads to complex implementation of security controls
- Cloud-based implementation of NoSQL databases:
  - Third-party security solutions are integrated into NoSQL databases
  - Cloud APIs are frequently updated, resulting in the introduction of new bugs that add security flaws in applications

Problems implementing traditional security solutions for NoSQL

1. NoSQL DBs rely primarily on external or vendor-compatible security that must be implemented in a middleware
2. NoSQL DB clustering capabilities add challenges to ensure robust security practices
3. Data in NoSQL DBs is stored as plain text and no encryption mechanism is implemented
4. Passwords in NoSQL DBs are encrypted using MD5 or PBKDF2 algorithms that are not very secure
5. Most NoSQL DBs deal with security issues in an ad hoc way: attack and then fix the problem
6. Weak authentication and password storage methods expose NoSQL to replay and brute-force attacks
7. Inability of NoSQL databases to apply authentication on cluster nodes
8. Authorization mechanisms are ineffective in NoSQL databases
9. NoSQL DB are vulnerable to injection attacks: JSON, schema injection, REST injection, view injection, GQL injection, etc.
10. NoSQL databases are prone to internal attacks
11. A NoSQL database does not support online auditing

Injection attacks on the NoSQL database

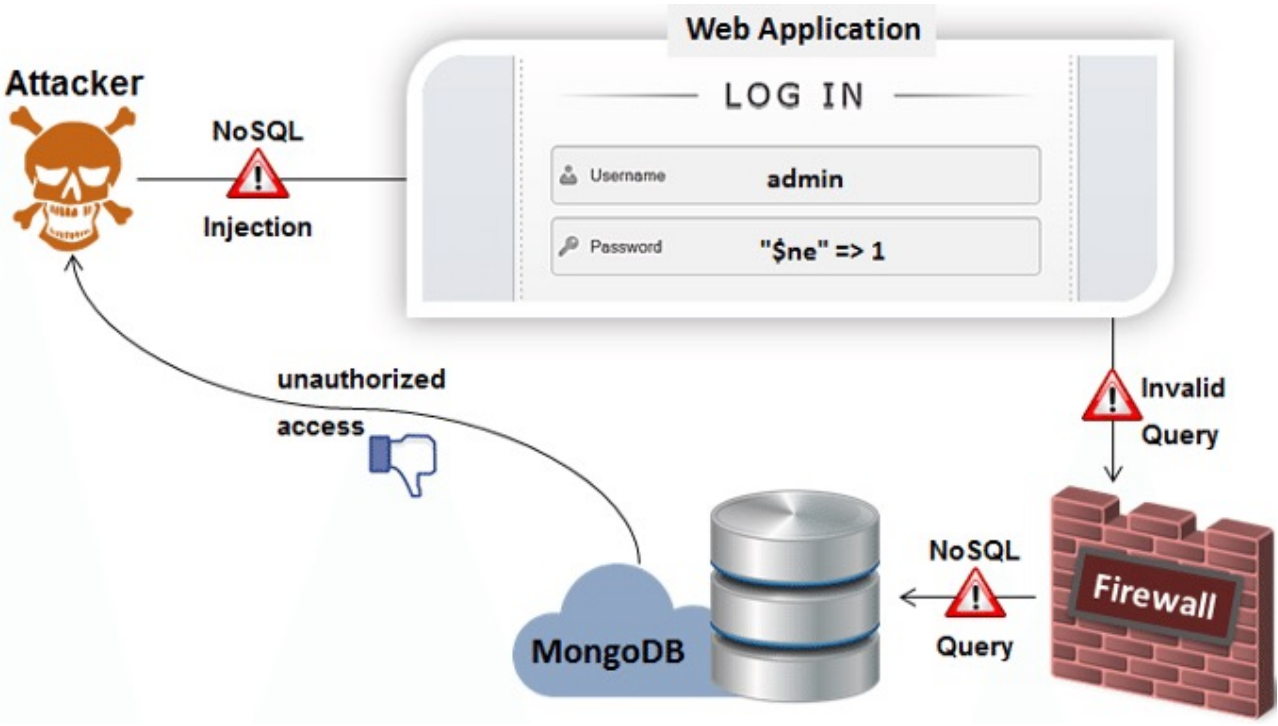
- NoSQL databases are not free of injection attacks
- Let's consider the following authentication request in a MongoDB:

```
db.accounts.find({username:
username, password: password});
```

- The attacker can inject this request using the JSON object as follows:

```
{
"username" :  "admin",
"password" : {$ne: 1}
}
```

- In MongoDB, `$ne:1` allows to select documents such that the password value is not equal to 1 and then access administrative documents



JavaScript Injections

- In JavaScript applications, the injection problem occurs when unsanitized data is concatenated to create a new structure
- Attacker sends simple text-based attacks that exploit the syntax of the targeted interpreter
- Almost any data source can be an injection vector, including internal sources
- Recommendation: to stick to APIs that do not involve string concatenation
- MangoDB example:

```
db.accounts.find({
        username:'admin', $where:function(){return 1}})//',
        password: password
});
```

Union Query Attack (type of SQL injection attack)

- The UNION query is inserted into a sensitive parameter that causes the original query to be unionized with the inserted query
  - bypass authentication pages and extract data
- Example: a login form receives the username and password via an http post:

```
string query = "{ username:'" + post_username + "', password:'" + post_password + "' }"
```

- If the username is "happy" and the password is "human", the request becomes:

```
{ username: 'happy', password: 'human' }
```

- A malicious user can transform this request to ignore the password:

```
username ='happy', $or:[{}, {'a': 'a&password =` }]
```

→ `{ username: 'happy', $or: [ {}, { 'a': 'a', password:'' } ] }`

- If the username is correct, the query succeeds
- In SQL, the similar query is:

```
SELECT * FROM logins WHERE username = 'happy' AND (TRUE OR ('a'='a' AND password='' ))
```

Piggybacked query

- The attacker injects additional requests into the original query:
  - Multiple database queries are received by the database
  - The original query is not modified, but additional queries are added to the original query
- The first query is executed normally, while subsequent queries are executed to satisfy the attack
- MangoDB examples:

```
db.accounts.find({username:''});db.user.drop(); db.users.insert({username:'ntamani',
password:'mypassword'})
```

DBMS and NoSQL RDBD Security Comparison: to sum up

Relational Databases

- Relational databases are a collection of tables that define data categories and constraints
- Relational databases use SQL as a tool to access data and are based on ACID properties
- Relational DBs have adopted highly secure mechanisms, but they face security threats such as root kits, SQL injections, cross-site scripts, etc.

NoSQL Databases

- NoSQL DBs are non-relational that provide elastic scaling running on low-cost hardware
- NoSQL has inherited the security issues of traditional RDBMS in addition to its specific vulnerabilities
- NoSQL databases are still prone to attack due to the unstructured nature of the data
- In NoSQL DBs, nodes are distributed to enable parallel computing that increases the attack surface

3. Challenges in designing, implementing, and deploying NoSQL DBs

- Technical challenges
- Non-technical challenges

Technical challenges (1/3)

- **Differences between data models** → lack of a standard model
  - Diverse data models, each with its own features, different versions of the query language, different tools,
  - No tools for passing from one NoSQL model to another
- **Lack of security**
  - NoSQL database lacks built-in security features
  - Security in middleware
- **Table join issues and indexing support**
  - NoSQL databases do not support traditional join operation
  - Developers must write join code according to database requirements
  - This can lead to inconsistency in the database
  - NoSQL databases do not support data indexing

Technical challenges (2/3)

- Limited ACID transaction support
  - Transactions not supported in NoSQL databases
  - Cassandra supports Atomicity, Durability and Isolation (AID) with adjustable consistency
- Transactional control
  - NoSQL databases work on transactional control of data written to a node
  - They give the user the flexibility to choose the level of inconsistency between multiple nodes
  - 100% consistency can be achieved by waiting for each write to be completed on each node
  - This process can introduce significant latency
- Open-Source Projects
  - Most NoSQL databases are open-source projects supported by startups
  - Lifespan of these databases depends on the lifespan of their startup

Technical challenges (3/3)

- Connectivity with common business intelligence tools
  - Most NoSQL databases do not provide connectivity with BI tools
  - Some features related to query and ad hoc analysis are available with the evolution of HIVE and PIG
- Schema flexibility
  - NoSQL models do not require a fixed schema or static column and row format
    - One entry can have 20 integers, and another can have 12 related strings
  - This flexibility of the scheme can speed up development, but also induces complexity.
- Data administration issues
  - NoSQL databases are designed to provide a zero-administration solution
  - But NoSQL requires a lot of skills and maintenance efforts due to the complexity of these databases

Non-technical challenges

- Supplier Viability Concerns
  - Several NoSQL databases are developed by small startups
  - Limited customer base: slow evolution
  - Ability to provide long-term support functions
- Maturity
  - NoSQL databases are in the pre-production phase with many key features to implement
  - NoSQL databases will take time to mature and be stable

NoSQL Security Reference Architecture: NoSQL Cluster Security

- NoSQL offers no security within the NoSQL cluster
  - Database and data security also depends on the network and application ecosystem
  - This security model is easy to implement
    → No degradation of performance or functionality
- Disadvantages:
  - Misuse of data by an accredited user
  - Exposing the system to the malicious user in the event of an application or firewall failure
- Some of the built-in security tools available in the NoSQL cluster:
  - SSL/TLS for encrypted communication,
  - Kerberos for node authentication,
  - Data security at rest using transparent encryption.
- These cluster security tools are difficult to implement and are expensive

NoSQL Cluster Security: Difference Between RDBMS and NoSQL

NoSQL Cluster Security: RDBMS Architecture

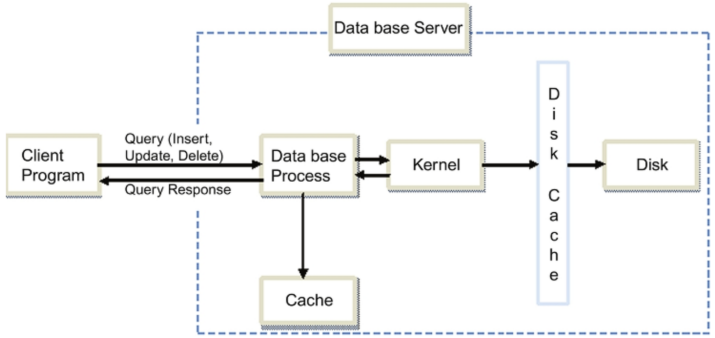


Fig. RDBMS architecture — ACID compliance.

23/05/2023 Database security 65

RDBMS — ACID compliance

NoSQL Cluster Security: NoSQL Architecture

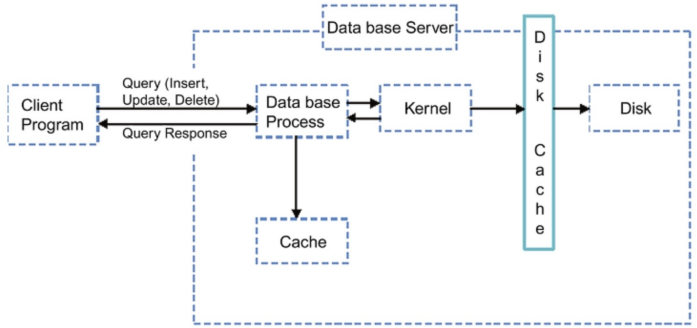


Fig. NoSQL—Fire and Forget.

23/05/2023 Database security 66

NoSQL — Fire and Forget

NoSQL Cluster Security: RDBMS Architecture

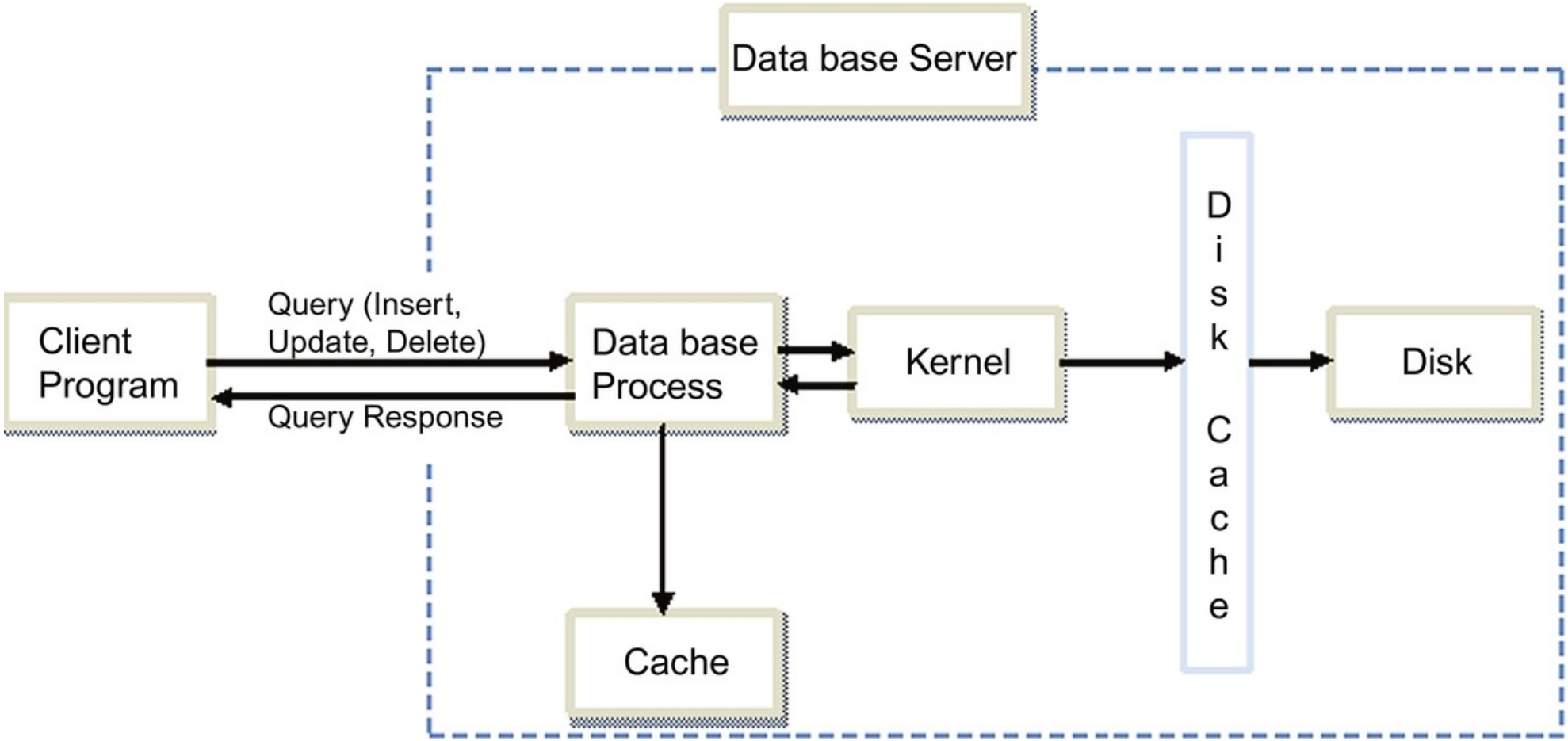


Fig. RDBMS architecture — ACID compliance.

NoSQL Cluster Security: NoSQL Architecture

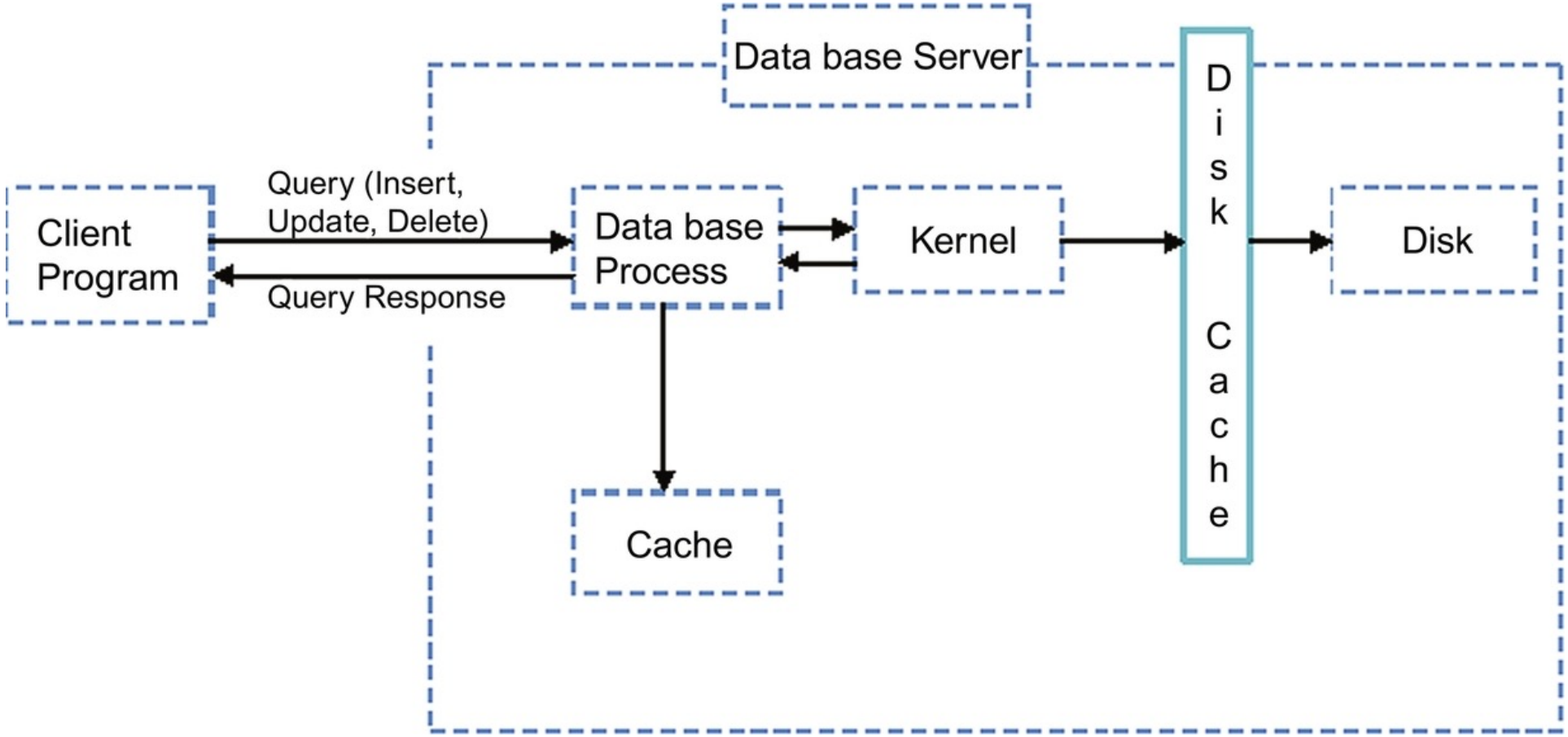


Fig. NoSQL—Fire and Forget.

NoSQL Cluster Security: Difference Between RDBMS and NoSQL

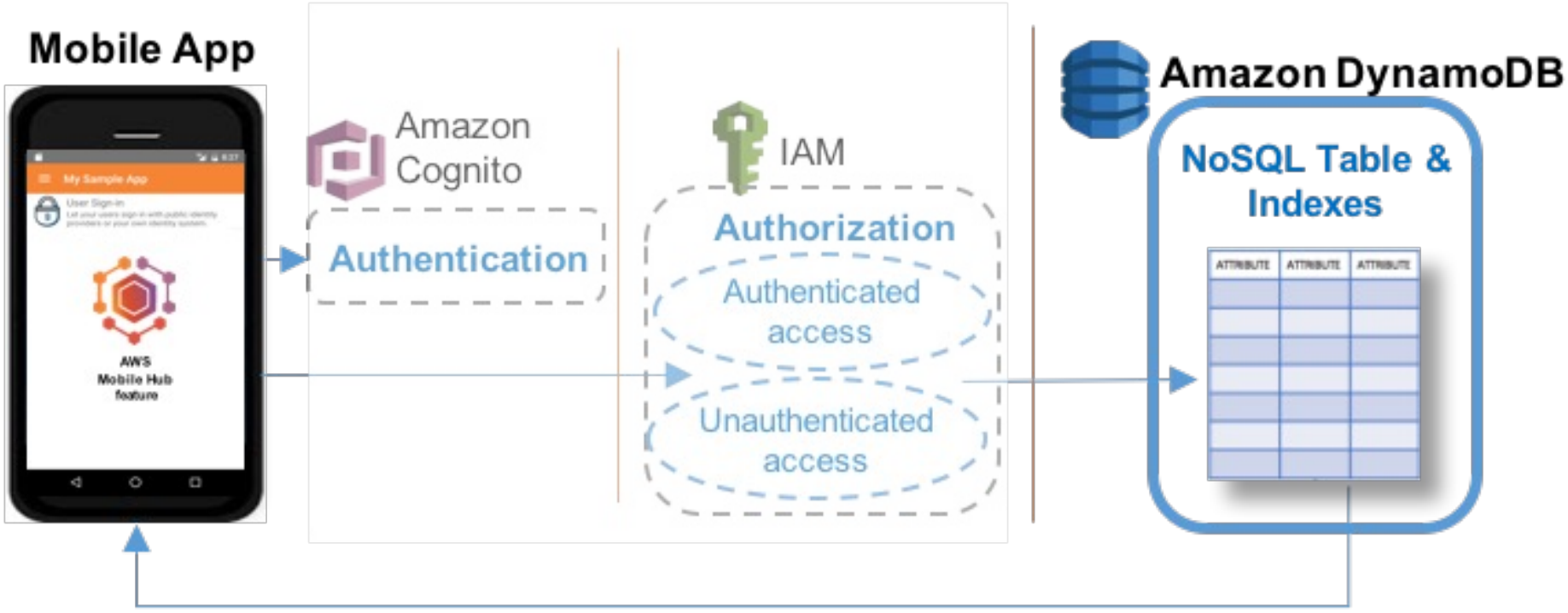
- A typical NoSQL DB compromises on some ACID properties
- In general, a complete security architecture should cover the following:
  1. User Access Management
  2. Logging operations
  3. Data protection
  4. Environmental and process control

NoSQL Cluster Security: Authentication

- Authentication is the process of identifying a person (subject) based on a username and a password
  - Users who need to access the database
  - Administration
  - Software systems
  - Physical and logical nodes
- Best Practices for Managing User Access:
  - Create login credentials: avoid creating a single administrator connection shared by all users
  - Centralized user access management
    - Provide the ability for databases to manage authentication within the database itself
  - Enforce password policies: Meet at least the minimum password complexity requirements

NoSQL Cluster Security: Authorization

- Authorization is the authenticated individual access process to system objects
- Best practices for authorization:
  - Grant minimal access to entities
  - Group common access privileges into roles
  - Control the actions of each individual entity
  - Control access to sensitive data



NoSQL Cluster Security: Audit

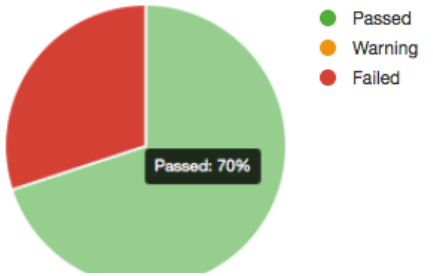
- Audit process detects attempts to access unauthorized data and logs can also be used for compliance
- Audit best practices include:
  - Track changes to the database configuration
  - Tracking data changes

Status ▲	Plugin Name	Plugin Family	Count
FAILED	Configure RBAC - DB Roles - 'Roles with anyAction privilege'	MongoDB Compliance Checks	1
FAILED	Configure RBAC - DB Roles - 'Roles with anyResource privilege'	MongoDB Compliance Checks	1
FAILED	Require Authentication - DB Users - 'User with userAdminAnyDatabase role'	MongoDB Compliance Checks	1
PASSED	Configure RBAC - Role Privileges - 'Roles with createRole privilege'	MongoDB Compliance Checks	1
PASSED	Configure RBAC - Role Privileges - 'Roles with createUser privilege'	MongoDB Compliance Checks	1
PASSED	Configure RBAC - User Roles - 'Users assigned the builtin dbAdminAnyDat…	MongoDB Compliance Checks	1
PASSED	MongoDB isMaster	MongoDB Compliance Checks	1
PASSED	MongoDB ServerStatus	MongoDB Compliance Checks	1
PASSED	Require Authentication - DB Users - 'User authenticated by MONGODB-CR'	MongoDB Compliance Checks	1
PASSED	Require Authentication - DB Users - 'User List'	MongoDB Compliance Checks	1

Scan Details

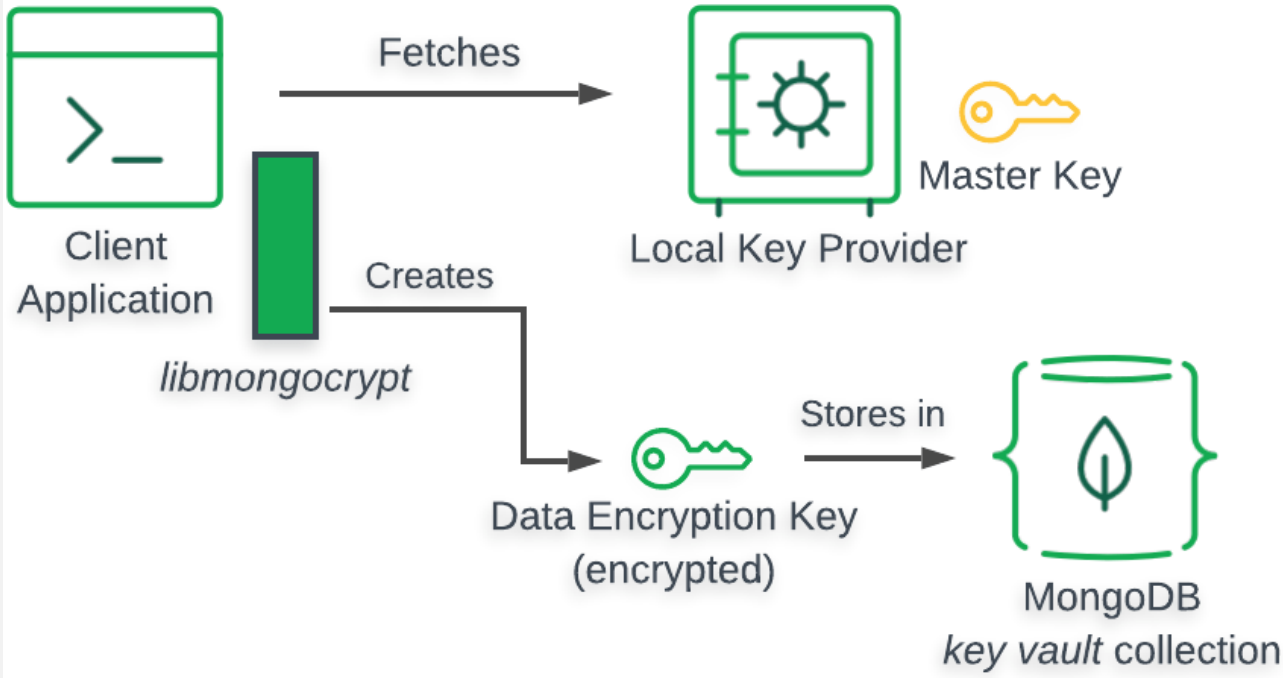
Name:	MongoDB Audit
Folder:	My Scans
Status:	Completed
Policy:	Policy Compliance Auditing
Scanner:	Local Scanner
Targets:	172.26.25.17
Start time:	December 05, 2014 13:23:57
End time:	December 05, 2014 13:24:44
Elapsed:	a minute

Compliance



NoSQL Cluster Security: Encryption

- Encryption is defined as the translation of data into a secret code
- Best practices for encryption:
  - Encrypting database connections
  - Encryption of data at rest
  - Applying strong encryption
  - Signing and rotating encryption keys



NoSQL Cluster Security: Environment and Process Control

- Protecting the underlying infrastructure is also very important
- Good protection practices include:
  - Installing Firewalls
  - Network Configuration
  - Setting file permissions

Data-centric approach to security

- NoSQL security is provided:
  - Through the network,
  - By applications that surround the data
  - By third-party tools available in a NoSQL cluster
- In data-centric security:
  - Security controls are part of the data, not the database
  - Data protection takes place before the data is saved in the database
- Three basic tools that support data-centric security:
  - Tokenisation
  - Masking
  - Encrypting data elements

Data-centric approach to security

- Tokenisation:
  - Replace sensitive data with data tokens to ensure security
  - A token has no base value; it only carries a reference to the original value of the database
  - Example: Credit card processing systems use a tokenization technique to replace credit card numbers
- Masking:
  - Replace an original data value with a random value to protect the data
  - Keep the original value for the analysis
  - Examples: a person's name, date of birth, etc.
- Data encryption:
  - Encrypt data and only authorized users can decrypt data using keys

Techniques to mitigate attacks on NoSQL DBDs

- Using firewalls:
  - Firewalls are used to protect the system from malicious and intentional attacks
  - Firewalls can be integrated near the data center to strengthen the user authentication process
- Security Analysis:
  - SQL injections can be detected by performing dynamic (DAST) or static (SAST) tests on the source code
  - DAST technology is more reliable because it uses backend inspection technology that improves detection reliability
  - Security testing should not be performed only during the testing phase
- API Protection: (Mitigation of Risks related to REST API or CSRF attacks)
  - Accept only JSON in the content type
  - Limit HTML forms to the URL-encoded content type → block HTML form attacks for CSRF (Cross-site request forgery)
  - Use of Ajax requests, which are blocked by the browser
  - Ensure that JSONP (JSON with Padding) and CSRF are disabled in the server API, so that no action can be performed directly from a browser

Techniques to mitigate attacks on NoSQL DBDs

- Audit and logging:
  - Database auditing should be done very frequently to identify incidents as early as possible
  - Third-party tools such as Scribe and Logstash can be integrated into the database to backup transaction data
  - Auditing using Scribe helps identify data theft and backing up transaction records can help guard against data misuse
- Authentication:
  - Authentication provides strong control over the user of the data
  - To be associated with password mechanisms
- Validation of entries:
  - Filter JavaScript that will help eliminate string injection and concatenation attacks

Techniques to mitigate attacks on NoSQL DBDs

- Access control:
  - Access control helps impose restrictions on data access
  - Access to data can be granted based on the organization's security policy and permission levels
- Segregation of Duties:
  - Access to data can be granted based on the role and responsibility of the user
  - Segregation of duties helps limit data theft and data misuse
- Encryption:
  - Transforms data into a format that can only be decrypted by users with an authorized key
  - It helps protect sensitive data

Security and privacy solutions for NoSQL DBs

- **Validation/filtering of endpoint entries:**
  - Validate entry when collecting data
  - Validate input sources
  - Filter malicious data, especially with the BYOD model
  - Data validation and filtering algorithms must be created
- **Real-time security and compliance monitoring:**
  - Big Data technologies are monitored using real-time security mechanisms
  - Data can be used to provide real-time anomaly detection based on scalable security analytics

Security and privacy solutions for NoSQL DBs

- Privacy:
  - Data anonymization is a way to protect privacy
  - It is important to design robust and scalable algorithms to prevent inadvertent access to private data
- Implement logical filters in the application space:
  - With granular access control mechanisms, applications can access data for many different purposes on behalf of a user
  - All strategies that prevent data combinations for cross-referencing
  - At the application level, properly separate the different uses of the data
- Enhanced access control by cryptography:
  - NoSQL databases store sensitive unencrypted data
  - Large datasets are difficult to encrypt because of their all-or-nothing recovery strategy
  - The problem of encryption can be solved by designing a cryptographically secure communication network

4. Storage of data and information

Security of storage media

Storage of data and information

- DBMS security covers access to information through traditional "gateway" channels
- Data is also processed through a computer's storage resources, both memories and physical media
- Precautions must be put in place to ensure that these core resources are also protected against threats

Types of storage

- Systems balance different types of storage to meet an organization's IT needs
- Several common types of storage:
  - Primary or "real" memory
    - Main memory resources directly available to a system's processor
    - Main memory: volatile random-access memory (RAM) and generally the most powerful storage resource available for a system
  - Secondary storage
    - Non-volatile and inexpensive storage resources available for a system for long-term use
    - Magnetic and optical media, such as tape, disks, hard drives, flash drives, and CD/DVD storage
- The following storage categories can also be distinguished:
  - Virtual memory
  - Virtual Storage
  - Random/sequential access storage
  - Volatile/Non-volatile storage

Storage Media Threats

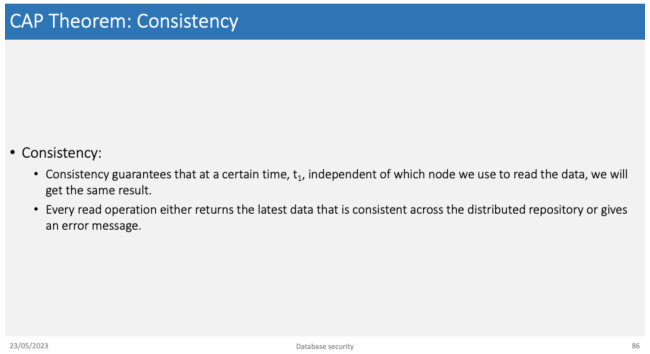
- The threat of illegitimate access to storage resources exists regardless of the type of storage used
- Without proper file system access controls, an intruder can access sensitive data simply by browsing the file system
- In more sensitive environments, you need to protect against attacks that bypass operating system controls
  - Directly access the physical storage medium to recover data
- Best practice:
  - Use an encrypted file system, which can only be accessed through the primary operating system
- In a multi-level security environment, adequate controls should be provided:
  - Ensure that shared memory and storage resources are configured in such a way that data at a classification level is not readable at a lower classification level

Storage Media Threats

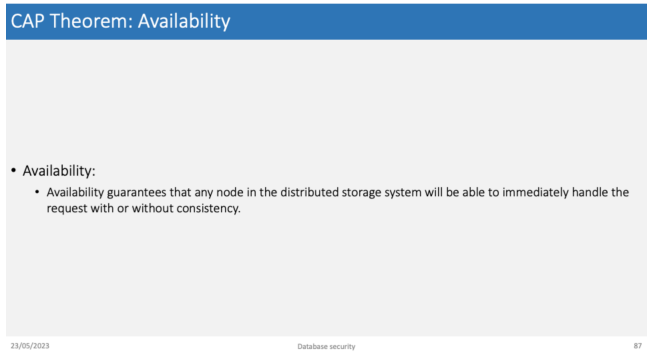
- Errors in storage media access controls become dangerous in cloud environments:
  - A single misconfiguration can publicly expose sensitive information on the web
- Organizations operating cloud storage systems, such as Amazon's Simple Storage Service (S3), must:
  - Be sure to set strong default security settings that restrict public access
  - Carefully monitor any changes to this policy that allow for public access
- Secret channel attacks are the second main threat to data storage resources
- Clandestine storage channels allow the transmission of sensitive data between classification levels through direct or indirect manipulation of shared storage media
- Writing sensitive data to a part of memory or physical storage shared inadvertently
- More complex secret storage channels can be used to manipulate the amount of free space available on a disk or the size of a file

CAP Theorem for designing Distributed Storage Systems

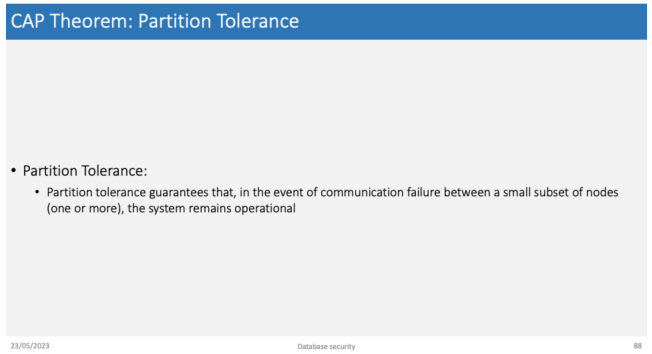
- Introduced in 1998 by Eric Brewer
- Trade-offs involved in designing a distributed storage system
- CAP: Consistency, Availability, and Partition tolerance



Consistency



Availability



Partition Tolerance

CAP Theorem: Consistency

- Consistency:
  - Consistency guarantees that at a certain time, t_1, independent of which node we use to read the data, we will get the same result.
  - Every read operation either returns the latest data that is consistent across the distributed repository or gives an error message.

CAP Theorem: Availability

- Availability:
  - Availability guarantees that any node in the distributed storage system will be able to immediately handle the request with or without consistency.

CAP Theorem: Partition Tolerance

- Partition Tolerance:
  - Partition tolerance guarantees that, in the event of communication failure between a small subset of nodes (one or more), the system remains operational

CAP Theorem for designing Distributed Storage Systems

- CAP theorem: in a storage system, we can only have two of the following characteristics: consistency, availability, and partition tolerance.
- The CAP theorem also means that we can have three types of distributed storage systems:
  - A CA system (implementing Consistency-Availability)
  - An AP system (implementing Availability-Partition Tolerance)
  - A CP system (implementing Consistency-Partition Tolerance)

Homework 1

- Exercise 1: Attack the http://testphp.vulnweb.com website
  - Objective:
    - Find username+passeword of a vulnerable website user with SQLMAP or JSQL
    - Connect to the site with the information exfiltrated
  - Steps:
    - If you do not have a Linux on your computer, then install Kali Linux Virtual Machine and the tools SQLMAP or JSQL
    - Find the link that will serve as the entry point
    - Find the database name
    - Find the name of the list of tables in the database
    - Find columns in the user table
    - Find the value of the columns corresponding to the username and password.
    - Access the account of the found user

Appendix 1: Comparison of RDBMS security and NoSQL security (1/2)

Point of difference	RDB	NoSQL DB
Transaction Reliability	Guarantees very high transaction reliability as they fully support ACID properties	Do not guarantee very high reliability as they range from BASE to ACID properties
Performance	Caching must be done with special infrastructure support	Performance is enhanced by caching data into system memory
Indexing	Index available on multiple column	Single index, key-value store
Data Model	Data Model is very specific and well organized. Columns and rows are described by well-defined schema	Data model does not use the table as storage structure and is schema less. Data model is very efficient in handling unstructured data as well
Scalability	Scalability is challenge in databases due to the dependency on scalability vertical greatest relational	NoSQL databases depend on horizontal scalability
Cloud	Not suitable for cloud environment as these databases do not support data search on full content. Relational databases are also very hard to extend beyond a limit	Well suited for cloud databases. All characteristics of NoSQL databases are highly desirable for cloud databases

Appendix 1: Comparison of RDBMS security and NoSQL security (2/2)

Point of difference	RDB	NoSQL DB
Handling big Data	Big Data handling is a challenging issue for relational databases	NoSQL databases are designed to handle Big Data
Data Warehouse	When the size of stored data increases, problems for relational databases	NoSQL databases are not designed to serve data warehouse. NoSQL databases are faster than data warehouse
Complexity	Complexity arises due to non-fixture of data into tables	NoSQL databases have the capabilities to store unstructured data
Crash Recovery	They guarantee crash recovery through recovery manager	NoSQL databases use replication method as backup to recuperate from crash
Authentication	Relational databases come very with authentication mechanism	A NoSQL database does not have strong authentication mechanism and are dependent on external method for this
Data Integrity	Relational databases ensure data integrity	A NoSQL database does not support data integrity on every occasion
Confidentiality	Data confidentiality is a well-known feature of relational database	Generally, data confidentiality is not achieved in NoSQL databases
Auditing	Relational databases provide mechanisms to audit database	Most of the NoSQL databases do not provide mechanism for auditing the database

Further Readings

- MongoDB Topology Design - Scalability, Security, and Compliance on a Global Scale by Nicholas Cottrell. ISBN 978-1-4842-5816-3e-ISBN 978-1-4842-5817-0. https://doi.org/10.1007/978-1-4842-5817-0. Apress Editor. 2020.
- **NoSQL Security** by Neha Gupta; Rashmi Agrawal, Manav Rachna. Chap 4 in A Deep Dive into NoSQL Databases: The Use Cases and Applications, Edited by Pethuru Raj, Ganesh Chandra Deka, 2018.
- https://www.ibm.com/topics/data-security