OBJECT ORIENTED TESTING METHODS

[redacted]
Computer science and Engineering

CHAPTER-3

OBJECT ORIENTED TESTING METHODS

Topics to be Covered

- Applicability of Conventional Test Case Design Methods
- Issues in Object Oriented Testing,
- Fault-Based Testing
- Scenario-Based Testing
- Random Testing
- PartitionTestingfor Classes
- Interclass Test Case Design

Object Oriented Testing

- Find the greatest possible number of errors with a manageable amount of effort applied over a realistic time span.
- The nature of OO programs changes both testing strategy and testing tactics.
- Do we need less efforts for testing because of greater reuse of design patterns?
- Answer is no! Binder argues that more testing is needed to obtain high reliability in OO systems, since each reuse is a new context of usage.

Testing Object-Oriented Applications

Why its different

- No sequential procedural executions
- No functional decomposition
- No structure charts to design integration testing
- Iterative O-O development and its impact on testing and integration strategies

Objectives

To cover the strategies and tools associated with object oriented testing

- Analysis and Design Testing
- Unit/Class Tests
- Integration Tests
- Validation Tests
- System Tests

To discuss test plans and execution for projects

Test Strategies

Issues to address for a successful software testing strategy:

- Specify product requirements long before testing commences
  For example: portability, maintainability, usability
  Do so in a manner that is unambiguous and quantifiable
- Understand the users of the software, with use cases
- Develop a testing plan that emphasizes “rapid cycle testing”
  Get quick feedback from a series of small incremental tests
- Build robust software that is designed to test itself
  Use assertions, exception handling and automated testing tools (Junit).
- Conduct formal technical reviews to assess test strategy & test cases “Who watches the watchers?”

Testing OOA and OOD Models

- The review of OO analysis and design models is especially useful because the same semantic constructs (e.g., classes, attributes, operations, messages) appear at the analysis, design, and code level.
- Therefore, a problem in the definition of class attributes that is uncovered during analysis will circumvent side effects that might occur if the problem were not discovered until design or code (or even the next iteration of analysis).
- By fixing the number of attributes of a class during the first iteration of OOA, the following problems may be avoided:
  Creation of unnecessary subclasses.
  Incorrect class relationships.
  Improper behavior of the system or its classes.
- If the error is not uncovered during analysis and propagated further more efforts needed during design or coding stages.

Applicability of Conventional Test Case Design

- A collection of layered subsystems that encapsulate collaborating classes results in the design of object-oriented applications.
- Each of these components of the system (subsystems and classes) carries out functions that help to fulfil system requirements.
- In an attempt to discover errors, it is important to evaluate an OO system at a number of different levels.
- As classes collaborate with each other and subsystems connect through architectural layers, this can happen.

Applicability of Conventional Test Case Design

•OO testing is similar strategically to the testing of traditional systems, but it is distinct tactically.

•Since the OO research and design models are identical to the resulting OO software in structure and content, "testing" starts with.

•The interpretation of these models. OO testing starts "in the small" with class testing once the code has been created.

•When one class collaborates with other classes, a series of tests are designed to conduct class operations and investigate whether errors occur.

Applicability of Conventional Test Case Design

•With the development of research and design models, the construction of object-oriented software begins.
•The evolutionary essence of the model of OO software engineering is attributed to
•These models begin as relatively informal representations of the specifications of the system and grow into comprehensive class models.
•Connections and relationships of class, machine design and allocation, and design of objects.
•At each point, in an effort to uncover errors before their propagation to the next iteration, the models can be checked

Applicability of Conventional Test Case Design

•It can be argued that, because of the same semantic constructs, the study of OO analysis and design models is particularly helpful.

•A problem in the class attribute description that is discovered during the study will bypass side effects

•Effects that could occur if before design, the issue was not discovered.

•The class's testing will take more time than necessary.

•Once the problem is eventually discovered, the device must be updated.

Applicability of Conventional Test Case Design

1.Identify each test case uniquely

- Associate test case explicitly with the class and/or method to be tested

2.State the purpose of the test

3.Each test case should contain:

a. list of specified states for the object that is to be tested

b. A list of messages and operations that will be exercised as a consequence of the test

c. A list of exceptions that may occur as the object is tested

d. A list of external conditions for setup (i.e., changes in the environment external to the software that must exist in order to properly conduct the test)

e. Supplementary information that will aid in understanding or implementing the test

4.Automated unit testing tools facilitate these requirements

Boundary Value Analysis (BVA)

•Boundary value analysis is based on checking between partitions at the boundaries.

•This includes maximum, minimum, limits inside or outside, typical values, and error values.

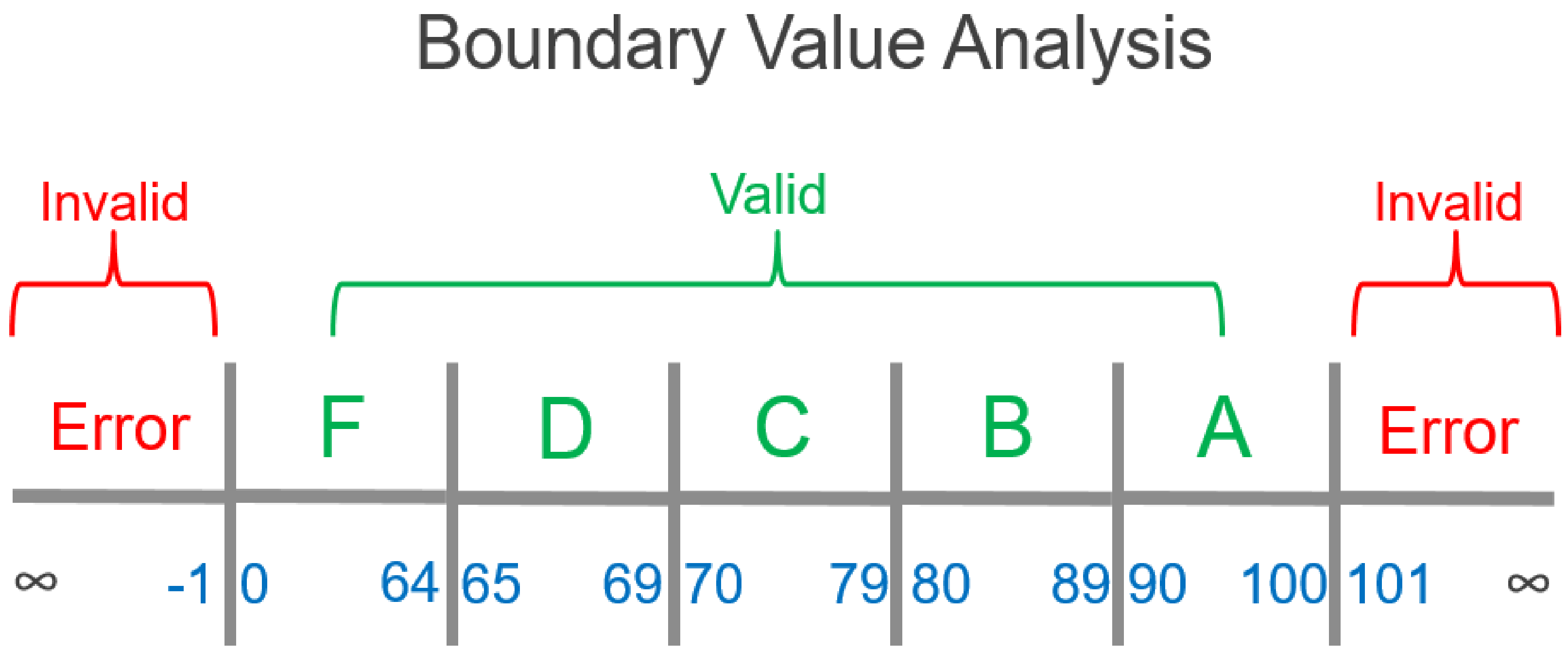
•A large number of errors are generally seen to occur at the boundaries of the given input values rather than at the middle.

• It is also referred to as BVA and offers a number of test cases that exercise bounding values.

Boundary Value Analysis (BVA)

•This black box testing technique complements equivalence partitioning..
•This software testing technique base on the principle that, if a system works well for these particular
•values then it will work perfectly well for all values which comes between the two boundary values.

Boundary Value Analysis (BVA)



Guidelines for Boundary Value Analysis (BVA)

•If an input condition is limited to x and y values, then the test cases should be constructed with x and y values, as well as values above and below x and y values.

•If the input condition is a large number of values, the test case that the minimum and maximum numbers must be exercised should be established.

•Values above and below the minimum and maximum values are tested here as well.

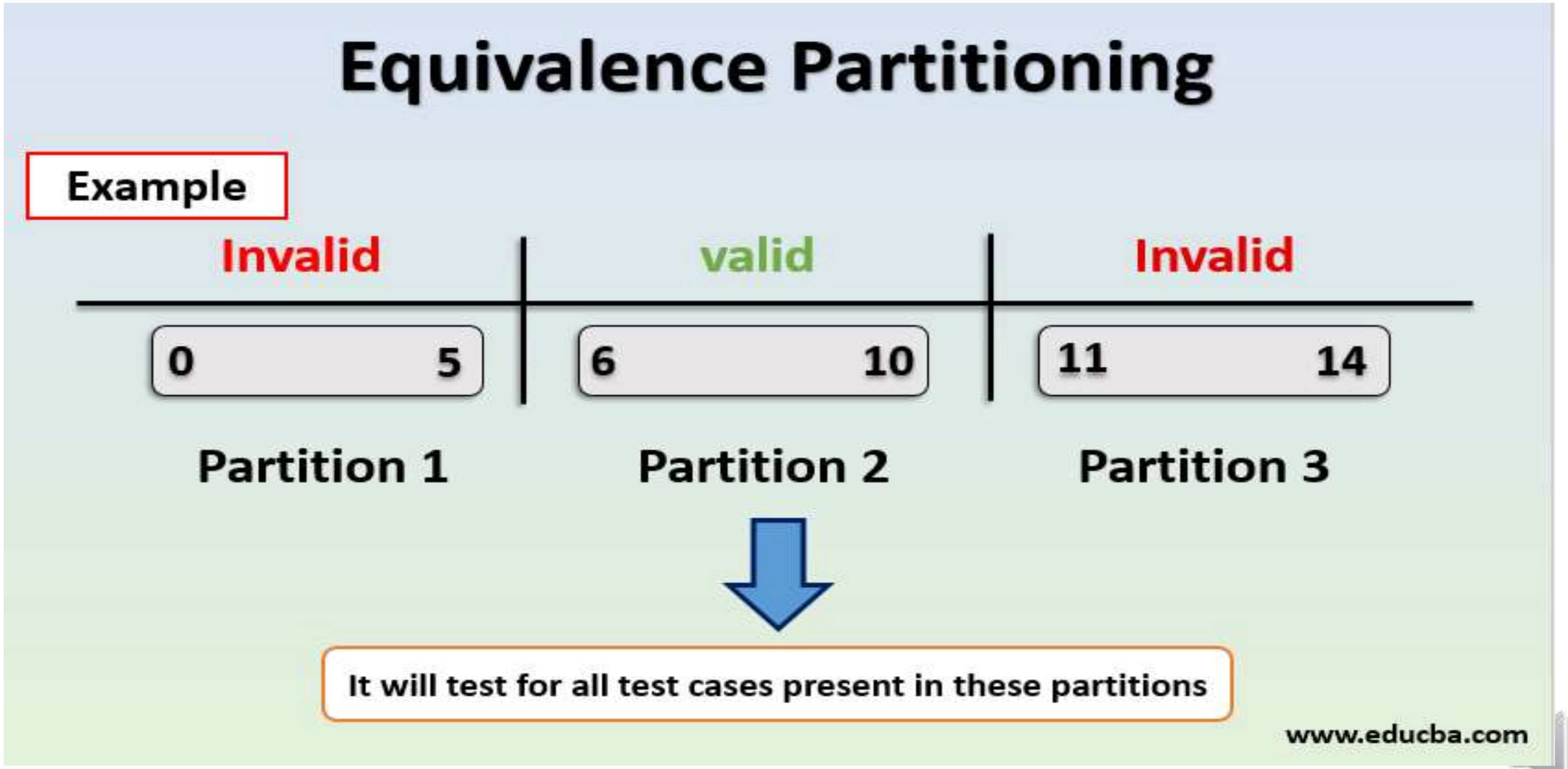
•For performance conditions, apply guidelines 1 and 2. This produces an output that represents the minimum and the highest.

Equivalence Class Partitioning

- Equivalent Class Partitioning enables you to break the test condition set into a partition that should be treated as the same.
- This approach for software testing divides a programme's input domain into groups of data from which test cases should be designed.
- The idea behind this strategy is that the test case is equal to a test with some other value of the same class with a representative value
- It is of each class that a value will be used.
- It allows both true and invalid equivalence groups to be defined.
- Example: Input conditions are valid between

  1 to 10 & 20 to 30

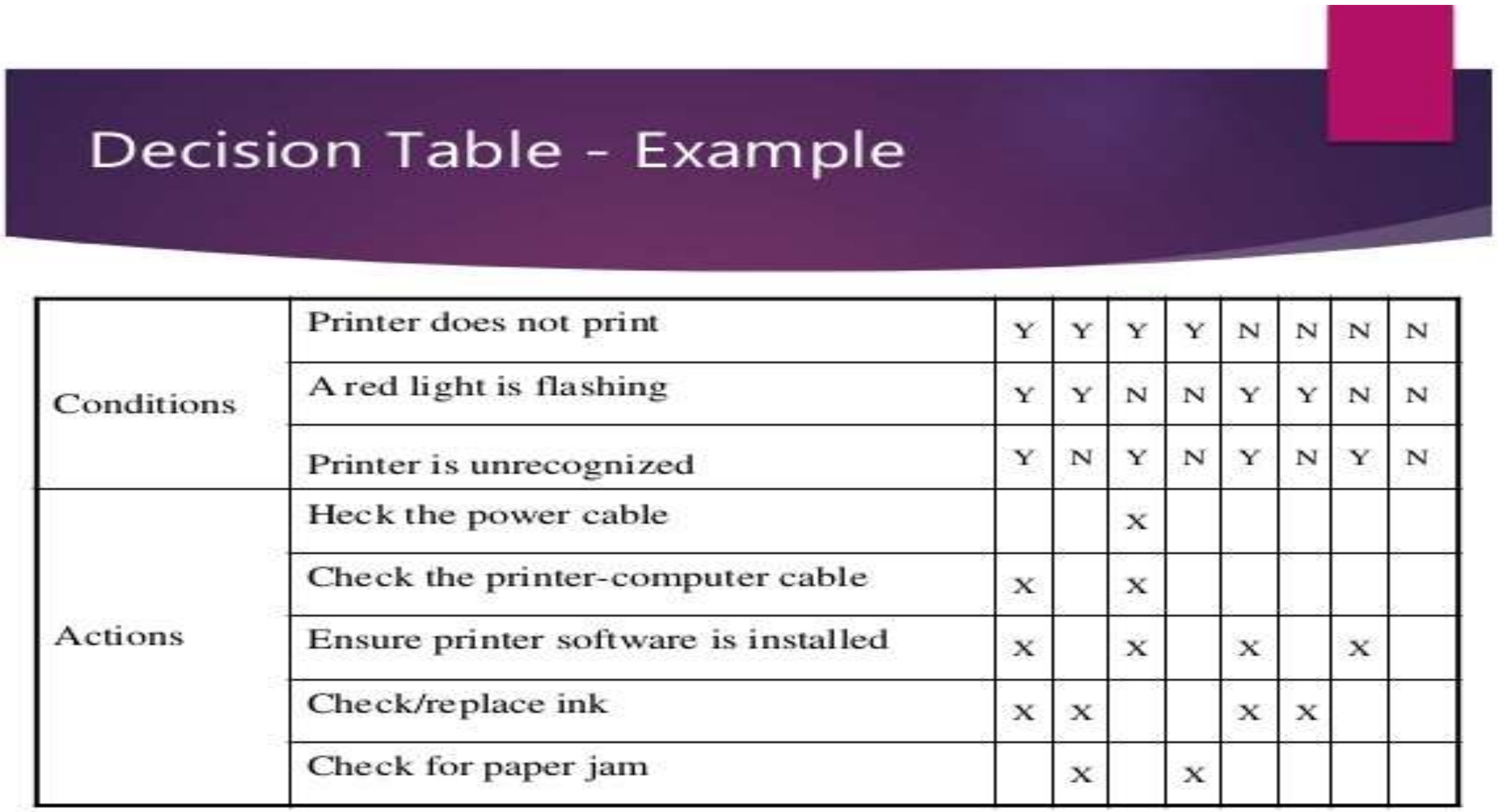
Equivalence Class Partitioning



Decision Table Based Testing.

- The Cause-Effect map is also known as a decision table.
- For functions which respond to a combination of inputs or events, this software testing technique is used.
- For example, the submit button should be activated if all the appropriate fields have been entered by the user.
- The first task is to define functionalities where a combination of inputs depends on the output.
- If there are broad combinations of inputs, then split them into smaller subsets that are helpful for handling a table of decisions.
- You need to construct a table for every function and list all types of input combinations and their respective outputs.
- This helps to describe a condition that the tester overlooks.

Decision Table Based Testing.



Decision Table - Example

Conditions	Printer does not print	Y	Y	Y	Y	N	N	N	N
	A red light is flashing	Y	Y	N	N	Y	Y	N	N
	Printer is unrecognized	Y	N	Y	N	Y	N	Y	N
Actions	Heck the power cable			x					
	Check the printer-computer cable	x		x					
	Ensure printer software is installed	x		x		x		x	
	Check/replace ink	x	x			x	x		
	Check for paper jam		x		x				

Issues in Object Oriented Testing

• As they include OO principles, conventional research approaches are not explicitly applicable to OO programmes.

•This includes encapsulation, polymorphism, and inheritance.

•Basic unit of monitoring for units.

•For unit test case design, the class is the natural unit.

•Apart from their class, the strategies are useless.

•A class in isolation can be tested by evaluating a class instance (an object).

•When independently validated classes are used in an application framework, they generate more complex classes.

Issues in Object Oriented Testing

• Before it can be deemed to be checked, the entire subsystem must be evaluated as a whole.

•2. Encapsulation Implication.

•Encapsulation of class attributes and methods can build barriers during testing.

•Since methods are invoked by the object of the corresponding class, it is not possible to perform testing without an object.

•The state of the object influences its actions at the time of the method's invocation.

•Testing is not only dependent on the object, but also on the state of the object, which is very complicated.

Issues in Object Oriented Testing

3Inheritance Implication.
Inheritance presents issues not present in conventional software.
The derived class does not always refer to test cases built for the base class.
In order to work properly in an OO environment, most testing methods need some kind of adaptation.
4. Polymorphism Consequences
A separate collection of test cases is needed for each possible binding of the polymorphic variable.
Before a client class can be checked, several server classes can need to be i

Issues in Object Oriented Testing

- Determining such bindings is difficult.
- It complicates preparation and testing for integration.

- 5. Implications for processes of research

- Both research methods and procedures need to be re-examined.

Fault Based Testing

•In an OO method, the purpose of fault-based testing is to design tests that have a high probability of uncovering probable faults.

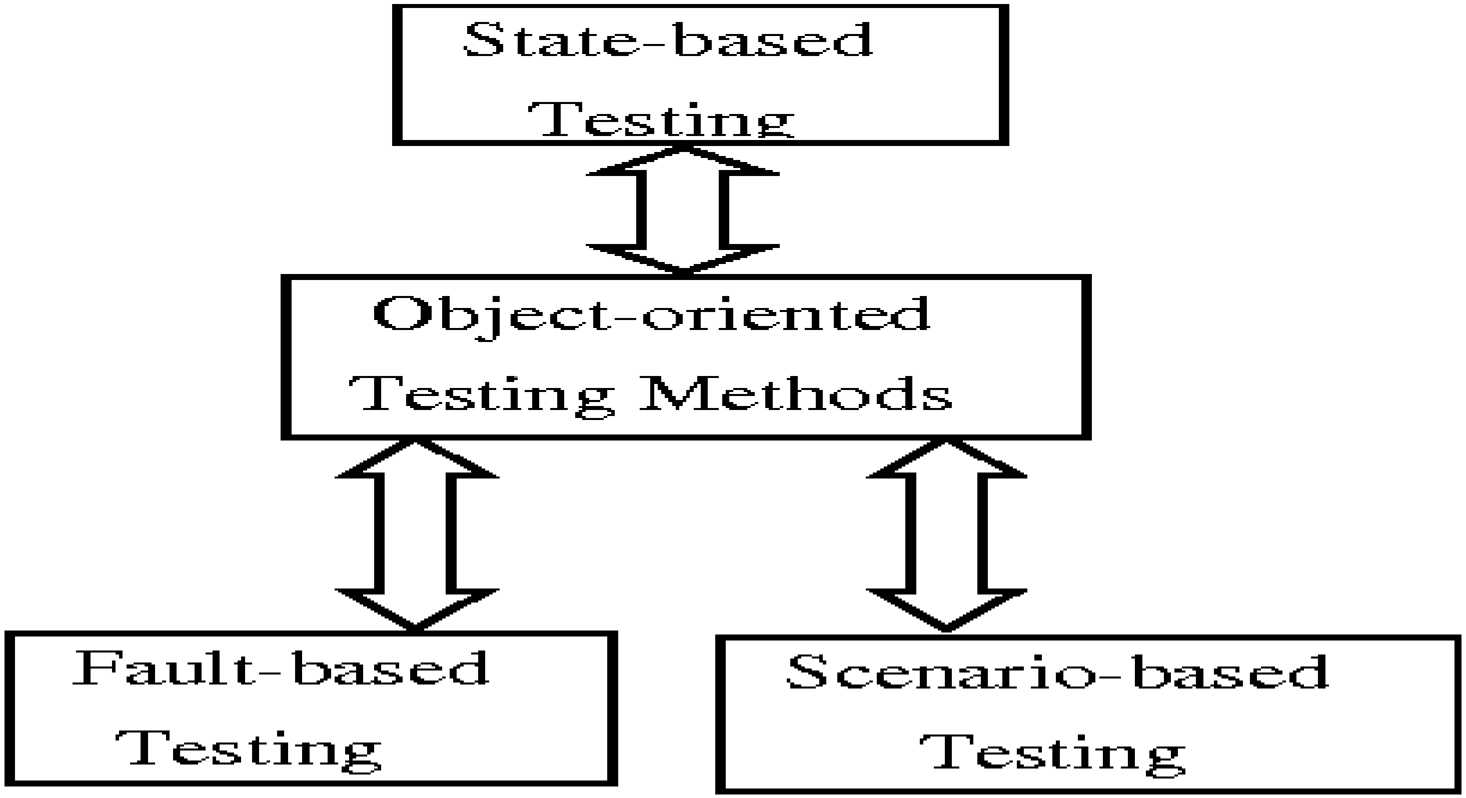
•The product or device must adhere to the requirements of the consumer.

•With the research model, the preliminary preparation needed to conduct fault-based testing starts.

•Plausible faults (i.e. aspects of system implementation that may result in defects) are looked for by the tester.

•Test cases are designed to exercise the practise of these faults to assess if these faults exist design or code.

Fault Based Testing



Fault Based Testing

•We design test cases for each plausible fault that will cause the incorrect expression to fail.

•The efficacy of these strategies depends on how a "plausible defect" is viewed by testers.

•If real faults are considered to be "unplausible" in an OO system, then this method is really no better than any random testing technique.

•If design and analysis may provide insight into what is likely to go wrong.

•With relatively low effort expenditures, it may find large numbers of errors.

Fault Based Testing

.

- An object's "behaviours" are specified by the values assigned by its attributes.
- To decide whether proper values exist for different types of object actions, testing can use the attributes
- It is important to remember that checking for integration seeks to identify errors in the client object, not the server.
- Integration testing focuses on deciding whether there are bugs in the calling code, not the code that is called.

Scenario Based Testing

•Two key forms of errors are ignored by fault-based testing

•(i)incorrect criteria and conditions

•(ii)The interactions between subsystems

•The product does not do what the consumer needs when mistakes associated with incorrect requirements occur.

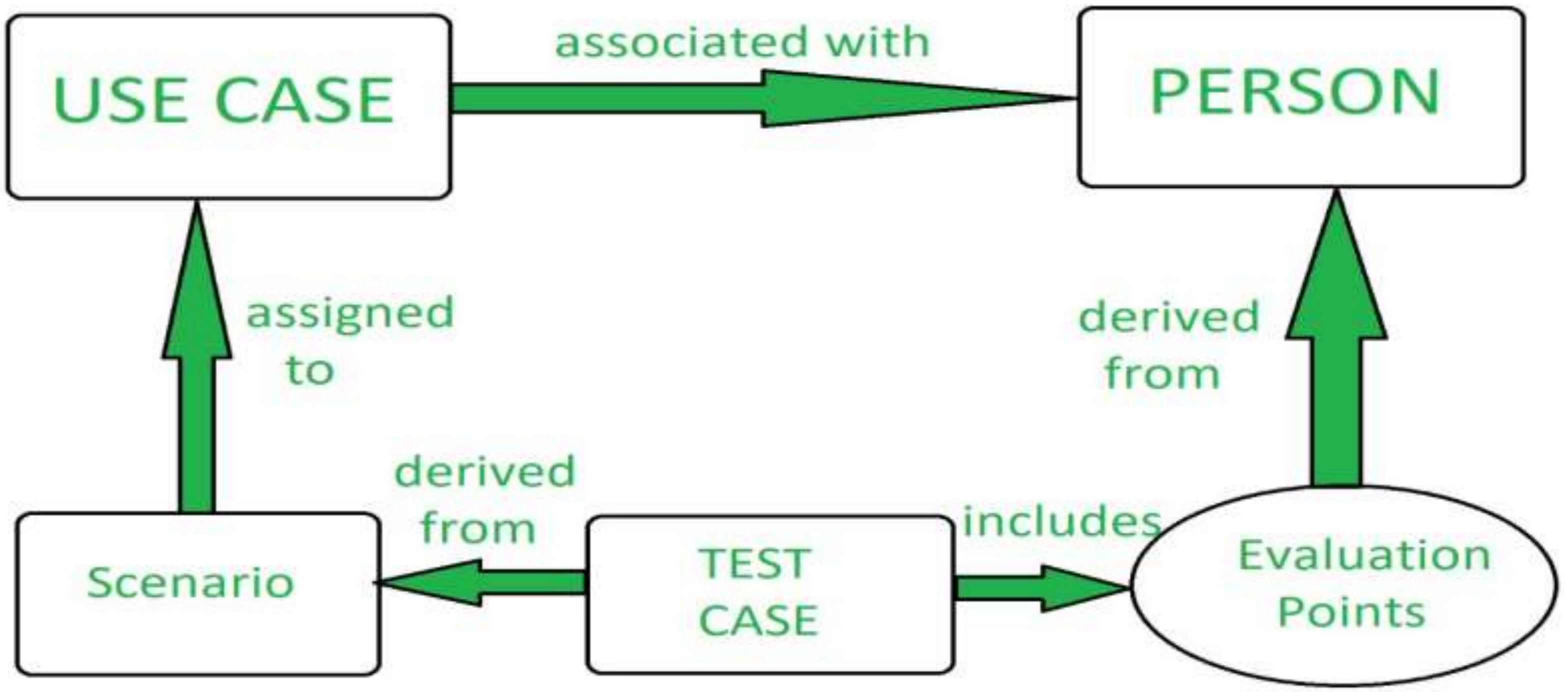
•It could do the wrong thing or it could omit substantial features.

•But in either case, consistency (compliance with requirements) suffers from

•Errors associated with subsystem interaction arise when one subsystem 's activity happens.

•This generates situations(events, data flow) that cause another subsystem to fail

Scenario Based Testing



Scenario Based Testing

•Scenario-based research focuses on what the consumer is doing, not what the object is doing.

•Scenarios uncover mistakes in conversation.

•Yet test cases must be more complicated and more practical than fault-based experiments to achieve this.

•In a single test, scenario-based training aims to exercise different subsystems.

•Think of the creation of scenario-based tests for a text editor as an example. Usage cases follow cases

•Use-Case: Print a copy of a new one

Scenario Based Testing

Background: Someone asks for a new copy of the document from the user.
1. Open this document.
2. Just print it.
3. Close down a text.
The approach to research is reasonably clear.
Except that it didn't come out of nowhere in this text.
In an earlier mission, it was generated. Will this assignment impact this one?
But this example implies a possible flaw in the specification.
The editor does not do what the consumer wants it to do fairly.
The review noted in phase 3 above is frequently missed by customers.

Scenario Based Testing

- When they trot off to the printer and find one page when they want 100, they will then be irritated.
- Annoyed customers report bugs in specifications
- This reliance in test design may be ignored by a test case designer
- But during testing, it is possible that the issue will emerge.
- Then the tester will have to deal with the likely answer, "That's the way it's supposed to work,"

Random Testing

•A type of functional black box testing is Random Testing, also referred to as monkey testing.
•That is achieved when there is not enough time for the tests to be written and executed.
•It is assisted by the development of a random test sequence that attempts the minimum number of operations typical of the categories' actions.
•Find a banking application to include brief examples of these approaches.
•In which the following operations are performed by an account class: open, setup, deposit, withdraw, balance, overview.

Random Testing

•For example, the account must be opened before it is possible to apply and close other operations after all operations have been completed.
•Even with these restrictions, the operations have several permutations.
•The minimum history of behavioural life of an account example contains the following operations:
•Open • configuration • deposit • withdraw •close
•This reflects the minimum account checking sequence.
•Within this series, however, a wide variety of other behaviours may occur:
•Open Setup Deposit Deposit • Open Setup Deposit.

Random Testing

You can randomly produce a number of different procedure sequences. For instance:
Test case r1: open • configuration • deposit • deposit • balance • summarise • withdraw • close • close
Test case r2: open • configuration • deposit • withdraw • deposit • balance • credit Cap • withdraw • close

This and other random order experiments are carried out to carry out distinct life histories of class instances.

Partition Testing

- Partition training decreases the number of test cases that are appropriate for the class to exercise.
- In much the same way as traditional device equivalency partitioning.
- Input and output are classified and test cases are planned for each group to be exercised.
- State-based partitioning classifies class operations on the basis of their ability to modify the class state.
- Again, state operations require deposit and withdrawal, given the account class.
- Whereas non-state behaviours include equilibrium, summarize, and credit Limit.

Partition Testing

- Tests are structured in a way that conducts state-changing operations and those that do not separately alter the state. Accordingly,
- Test case p1: open • setup • deposit • deposit • withdraw • withdraw • close Configuration • deposit • withdraw
- Test case p2: open • configuration • deposit • summarise • credit cap • withdraw • close • close
- Test case p1 alters status, while test case p2 conducts operations that do not modify status (other than those in the minimum test sequence).
- For each partition, test sequences are then built.
- Category-based partitioning categorizes class operations based on the generic function that each performs.

Interclass Test Case Design

•As integration of the OO system starts, test case design becomes more complicated.
•It is at this point that testing must begin for partnerships between classes.
•We extend the banking example introduced in Section 23.5 to include the classes in order to demonstrate 'interclass test case generation.'
•The orientation of the arrows in the figure shows the direction of the messages, and the operations are shown by the dot.
•As a result of the partnerships implied by the messages, that are invoked.

Interclass Test Case Design

•Like the testing of individual students, it is possible to conduct class collaboration testing by applying randomly.

•Methods of partitioning, as well as scenario-based research and testing of actions.

Refrences

[1]. Roger Pressman, Software engineering- A practitioner’s Approach, McGraw-Hill International Editions
[2]. Yogesh Singh, Software Testing, Cambridge University Press.
[3]. William E. Perry, Effective Methods for Software Testing, Second Edition, John Wiley & Sons
[4]. Ron Paton, Software Testing, second edition, Pearson education.
[5] Dohorthy Graham foundation of Software Testing,ISTQB Certification
[6] Softwaretesting. Tutorialspoint. https://www.tutorialspoint.com/Software testing

Parul® University | NAAC GRADE A++

https://paruluniversity.ac.in/

Parul® University

DIGITAL LEARNING CONTENT

Characteristics of NoSQL

- Does not follow ACID properties
- Less complexity as of SQL queries
- Schema less
- Fast development
- Large data volumes
- Easy and frequent DB changes

When to use NoSQL

•When Traditional RDBMS model provides restriction
•When ACID properties is not supported
•Temporary data(Like Wishlist, shopping cart, session data etc)
•When joins are expensive in RDBMS(product cost, hardware, maintenance)

When not to use NoSQL

•Financial Data, Data requiring strict ACID properties, Buisness critical data

CAP Theorem

•**Consistency**-This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.

•**Availability**-This means that the system is always on (service guarantee availability), no downtime.

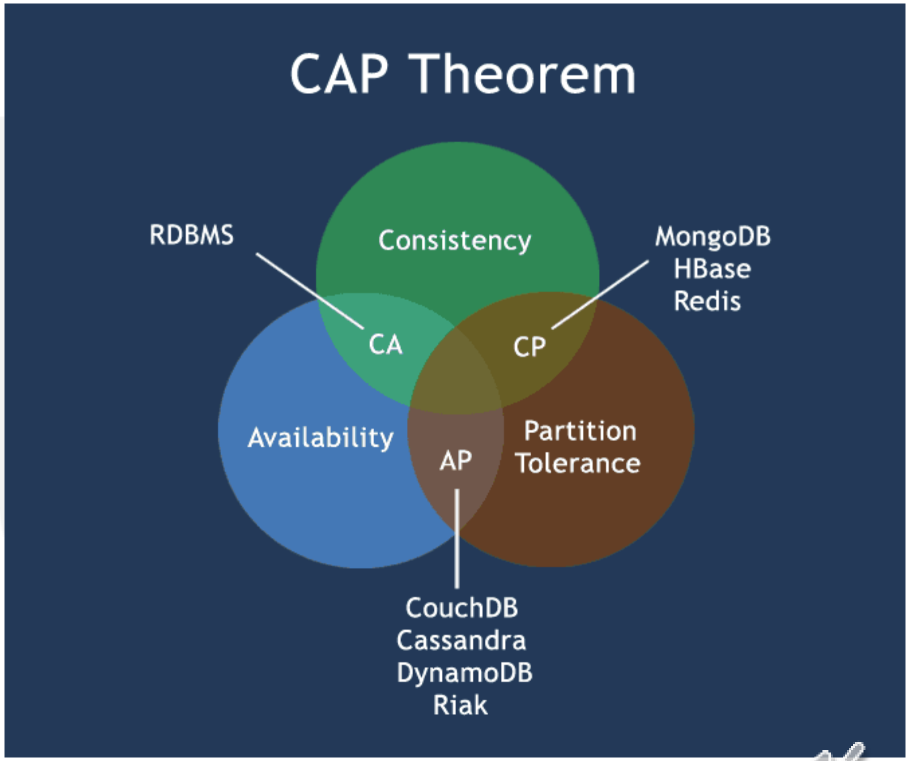
•**Partition Tolerance**-This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another. Here, if part of the database is unavailable, other parts are always unaffected.

CAP Theorem

•**CA** -Single site cluster, therefore all nodes are always in contact. When a partition occurs, the system blocks.

•**CP**-Some data may not be accessible, but the rest is still consistent/accurate.

•**AP**-System is still available under partitioning, but some of the data returned may be inaccurate.



The BASE

- The CAP theorem states that a distributed computer system cannot guarantee all of the following three properties at the same time:
    - Consistency
    - Availability
    - Partition tolerance
- A BASE system gives up on consistency.

The BASE

•**Basically Available** indicates that the system *does guarantee availability, in terms of the CAP theorem. i.e. DB is available all the time.*

•**Soft state** indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.

•**Eventual consistency** indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

Eventual Consistency

•The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.

•Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

Types of NoSQL

- There are 4 main types of NoSQL databases:
  - -Key value
  - -Column Family
  - -Document
  - -Graph

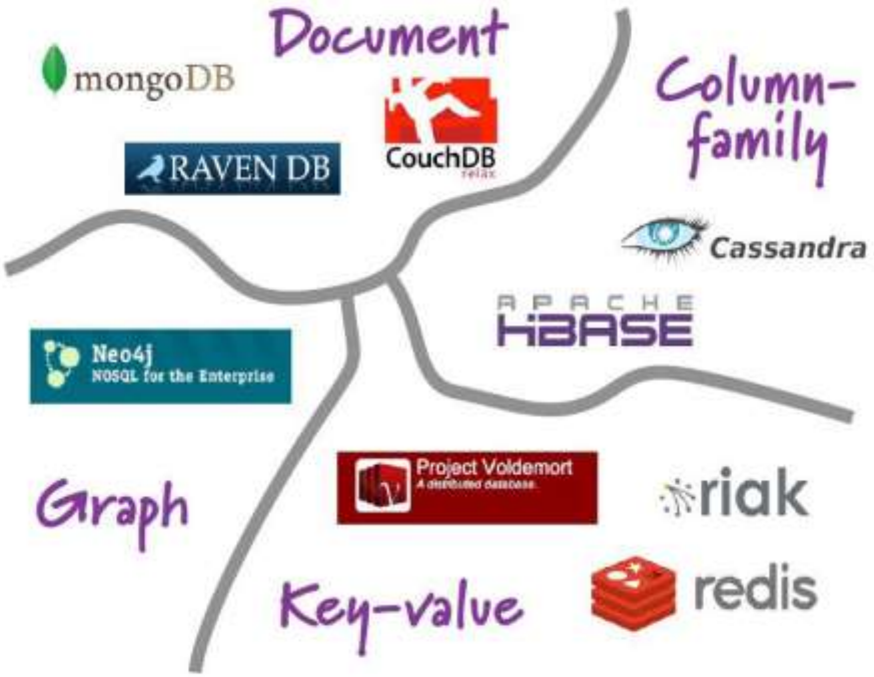


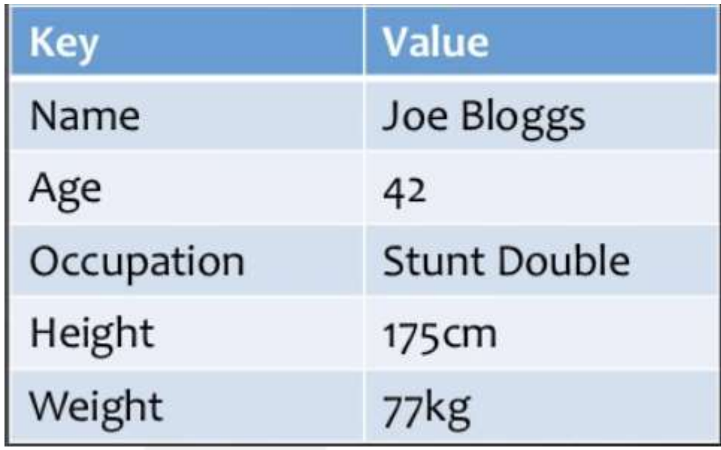
Image source : Google

Key Value

• Data is stored in Key/value pairs. It is designed in such a way that it can handle lots of data and heavy load.

•Key-value pair storage databases store data as a hash table where each key is unique and the value can be JSON, BLOB(Binary Large Object), String etc can be accessed by string called keys

•For EX: A key value pair may contain key like "University" associated with value like "Parul".



Key	Value
Name	Joe Bloggs
Age	42
Occupation	Stunt Double
Height	175cm
Weight	77kg

Key Value

• It is one of the most basic types of NoSQL databases

•This kind of database is used as a collection,

dictionaries, associative arrays etc. key value stores

help the developer to store schemaless data

• They work best for Shopping cart contents

Basic Operations are

Insert(Key,value), Fetch(key), Update(key), delete(Key).

Column- based

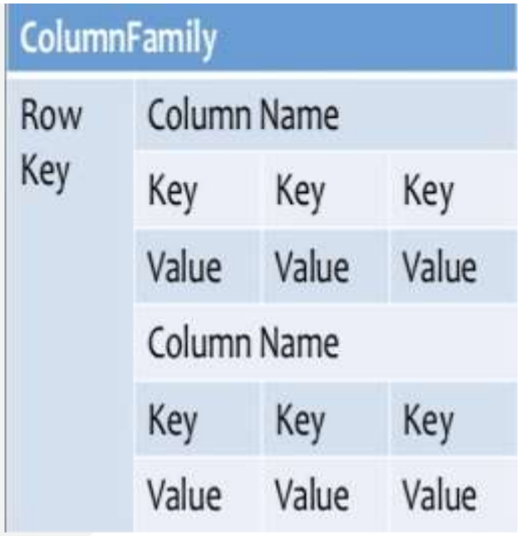
•Column oriented databases work on columns and are based on Big table paper by Google.

•The column is the lowest/ smallest instance of data.

•It is a tuple that contains a name, value and a timestamp

•Every column is treated separately.

•Values of single column databases are stored contiguously.



ColumnFamily			
Row Key	Column Name		
	Key	Key	Key
	Value	Value	Value
	Column Name		
	Key	Key	Key
	Value	Value	Value

Column based

•They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

•Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs

•HBase, Cassandra, HBase, Hypertableare examples of column based database

Some statistics about Facebook search(Using Cassandra).

MySQL>50gb data	Rewritten in Cassandra<50gb data
Writes average – 300 ms	Writes average: 0.12 ms
Reads average _ 350 ms	reads average: 15ms

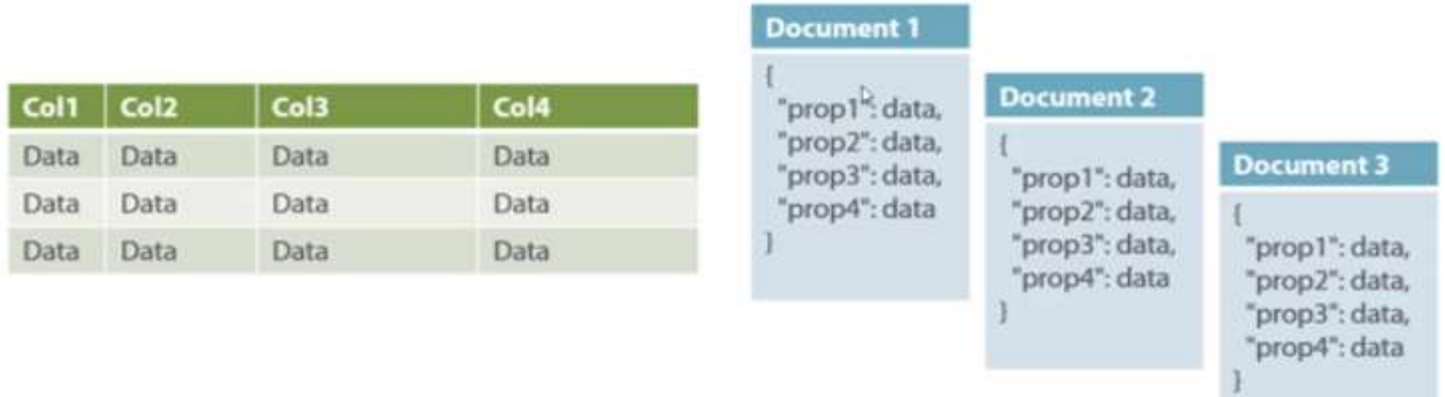
Document based

•Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document.

•The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

•Pair each key with complex data structure known as data structure.

•Documents can contain many different key-value pairs, or key-array pairs, or even nested documents.



Col1	Col2	Col3	Col4
Data	Data	Data	Data
Data	Data	Data	Data
Data	Data	Data	Data

Document 1
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document 2
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document 3
```
{
 "prop1": data,
 "prop2": data,
 "prop3": data,
 "prop4": data
}
```

Document based

•The document type is mostly used for CMS(content management system) systems, blogging platforms, real-time analytics & e-commerce applications.

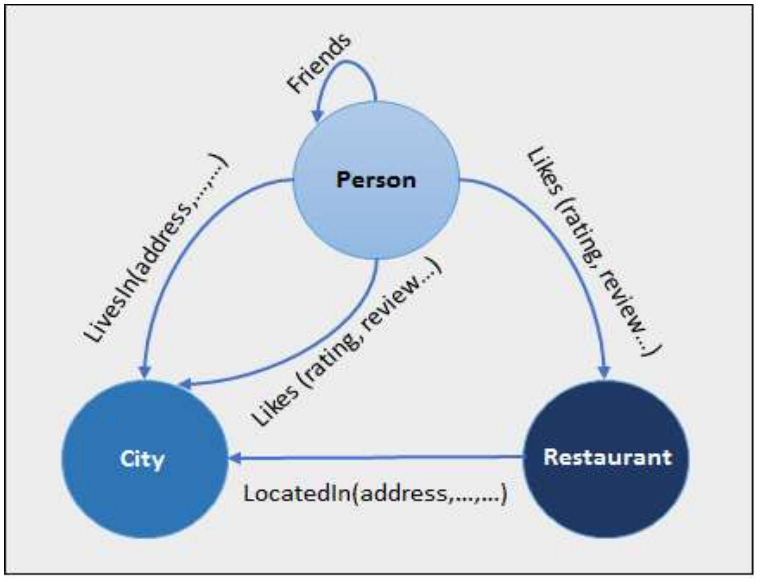
•It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

•Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes,are popular Document originated DBMS systems.

Graph based

•A graph type database stores entities as well the relations amongst those entities.

•The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.


Friends
Person
LivesIn(address,.....,...)
Likes (rating, review...)
Likes (rating, review...)
City
Restaurant
LocatedIn(address,....,...)

Graph based

•Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

•Graph base database mostly used for social networks, logistics, spatial data.

•Neo4J, Infinite Graph, OrientDB, FlockDBare some popular graph-based databases.

Parul® University

Comparison of SQL and NoSQL

1.Relational Database Management System

2. SQL is also know as Structured query language.

3.They have predefined Schema

1.Non relational management system

2.NoSQL is also known as Not only Structured query language

3.They have dynamic schema for unstructured data

Comparison of SQL and NoSQL

4.Vertically scalable	4. Horizontally scalable
5. RDBMS are Table based	5. NoSQL databases are document, key-value, graph or wide-column based
6.SQL are better for multi-row transactions.	6. NoSQL is better for unstructured data like documents or JSON
7. Example: Oracle, MySQL	7. Example: MongoDB, Cassandra, CouchDB etc.

What is NewSQL

•The term NewSQL is not exactly as wide as NoSQL.

•NewSQL systems all begin with the relational data model and the SQL query language and they all attempt to cover portion of similar types of scalability, flexibility or lack-of-focus that has driven the NoSQL development.

Why NewSQL ?

•The early counter-measures to the above approaches were a powerful single node machine that is able to handle all the transaction, a custom built middleware system to distribute queries over traditional DBMS nodes.

•But both of them are prohibitively expensive to be carried out.

Why NewSQL ?

•As a result, there was a need of an intermediate database system which combines the distributed architectures of NoSQL systems with multiple node concurrency and a whole new storage mechanism.

•Thus Newsql can be defined as a class of modern relational DBMSs that seek to provide the same scalable performance of NoSQL for OLTP workloads and simultaneously guaranteeing ACID compliance for transactions as in RDBMS.

•these systems want to achieve the scalability of NoSQL without having to discard the relational model with SQL and transaction support of the legacy DBMS.

Concepts of NewSQL

•Main Memory storage of OLTP(Online transition process) databases enables in-memory computations of databases.

•Scaling out by splitting a database into disjoint subsets called either partitions or shards, leading to the execution of a query into multiple partitions and then combine the result into a single result.

•NewSQL systems preserve the ACID properties of databases.

•Enhanced concurrency control system benefits upon traditional ones.

Concepts of NewSQL

•Presence of secondary index allowing NewSQL to support faster query processing times.

•High availability and Strong data durability is made possible with the use of replication mechanisms.

•NewSQL systems can be configured to provide synchronous updates of data over the WAN.

•Minimizes Downtime, provides fault tolerance with its crash recovery mechanism.

Comparison of NoSQL and NewSQL

1. It doesn't follow a relational model, it was designed to be entirely different from that.	1. Since the relational model is equally essential for real-time analytics.
2. Supports CAP Theorem	2. Supports ACID property
3. Supports old SQL	3. Supports old SQL and even enhanced functionality o f old SQL

Parul® University

Comparison of NoSQL and NewSQL

4. Supports OLTP database but it is not best suited.	4. Supports OLTP database and highly efficient.
5. Vertical Scaling	5. Vertical and horizontal scaling
6. Better than SQL for processing complex queries	6. Highly efficient to process complex queries and smaller queries
7. Supports distributed system	7. Supports distributed system

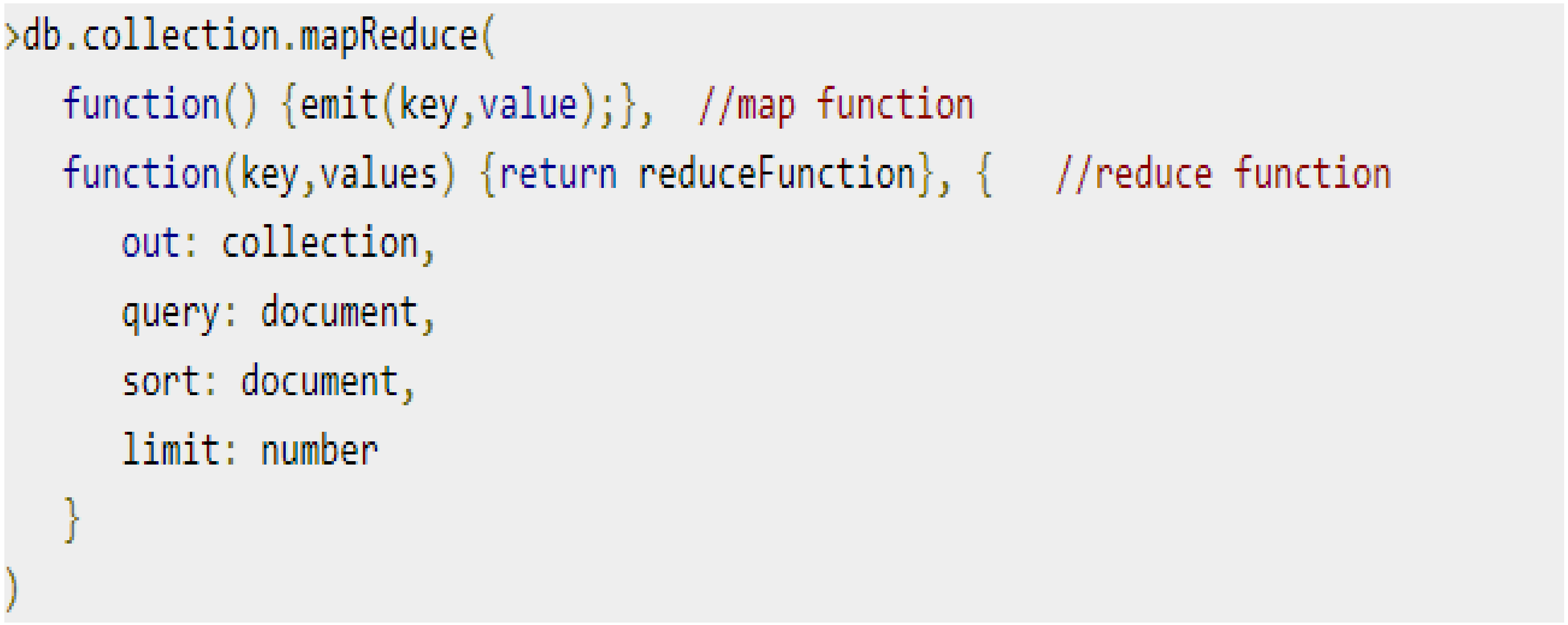
Map reduce

•Map-reduce is a data processing paradigm for condensing large volumes of data into useful aggregated results.

•MongoDB uses mapReduce command for map-reduce operations.

•MapReduce is generally used for processing large data sets.

MapReduce Command



```
>db.collection.mapReduce(
   function() {emit(key,value);},  //map function
   function(key,values) {return reduceFunction}, {   //reduce function
      out: collection,
      query: document,
      sort: document,
      limit: number
   }
)
```

MapReduce Command

The map-reduce function first queries the collection, then maps the result documents to emit key-value pairs, which is then reduced based on the keys that have multiple values.

In the above syntax –
map is a javascript function that maps a value with a key and emits a key-value pair
reduce is a javascript function that reduces or groups all the documents having the same key
out specifies the location of the map-reduce query result
query specifies the optional selection criteria for selecting documents
sort specifies the optional sort criteria
limit specifies the optional maximum number of documents to be returned

Parul® University

DIGITAL LEARNING CONTENT

Using MapReduce

Consider the following document structure storing user posts. The document stores user_name of the user and the status of post.

```
{
   "post_text": "tutorialspoint is an awesome website for tutorials",
   "user_name": "mark",
   "status":"active"
}
```

Using MapReduce

Now, we will use a mapReduce function on our posts collection to select all the active posts, group them on the basis of user_name and then count the number of posts by each user using the following code –

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },

   function(key, values) {return Array.sum(values)}, {
      query:{status:"active"},
      out:"post_total"
   }
)
```

Using MapReduce

The above mapReduce query outputs the following result –

```
{
   "result" : "post_total",
   "timeMillis" : 9,
   "counts" : {
      "input" : 4,
      "emit" : 4,
      "reduce" : 2,
      "output" : 2
   },
   "ok" : 1,
}
```

Using MapReduce

The result shows that a total of 4 documents matched the query (status:"active"), the map function emitted 4 documents with key-value pairs and finally the reduce function grouped mapped documents having the same keys into 2.

Using MapReduce

To see the result of this mapReduce query, use the find operator –

```
>db.posts.mapReduce(
   function() { emit(this.user_id,1); },
   function(key, values) {return Array.sum(values)}, {
      query:{status:"active"},
      out:"post_total"
   }

).find()
```

Using MapReduce

The above query gives the following result which indicates that both users tom and mark have two posts in active states –

```
{ "_id" : "tom", "value" : 2 }
{ "_id" : "mark", "value" : 2 }
```

In a similar manner, MapReduce queries can be used to construct large complex aggregation queries. The use of custom Javascript functions make use of MapReduce which is very flexible and powerful.

Aggregations

- Aggregations operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL count(*) and with group by is an equivalent of MongoDB aggregation.

Aggregations

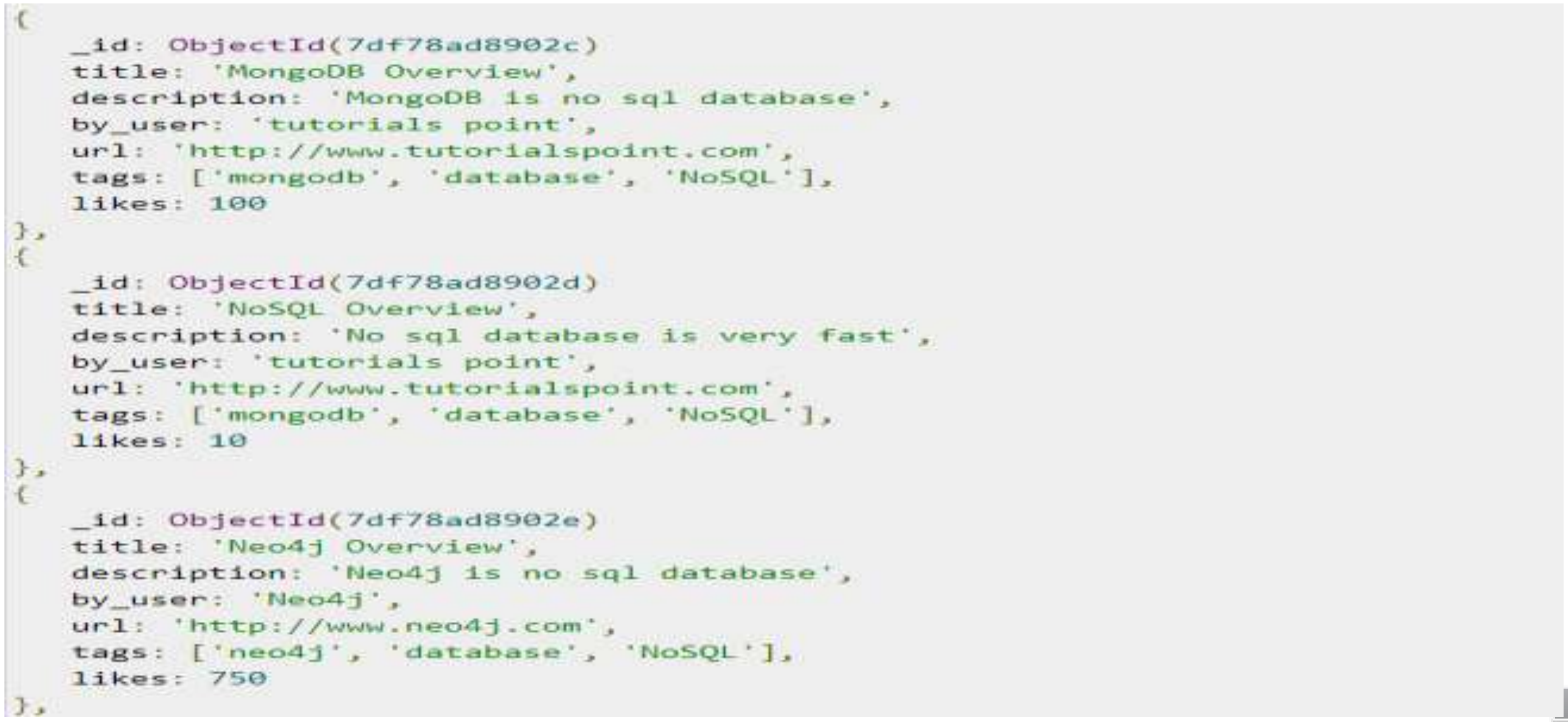
- For the aggregation in MongoDB, you should use aggregate() method.

- Syntax
- Basic syntax of aggregate() method is as follows –

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

Aggregations

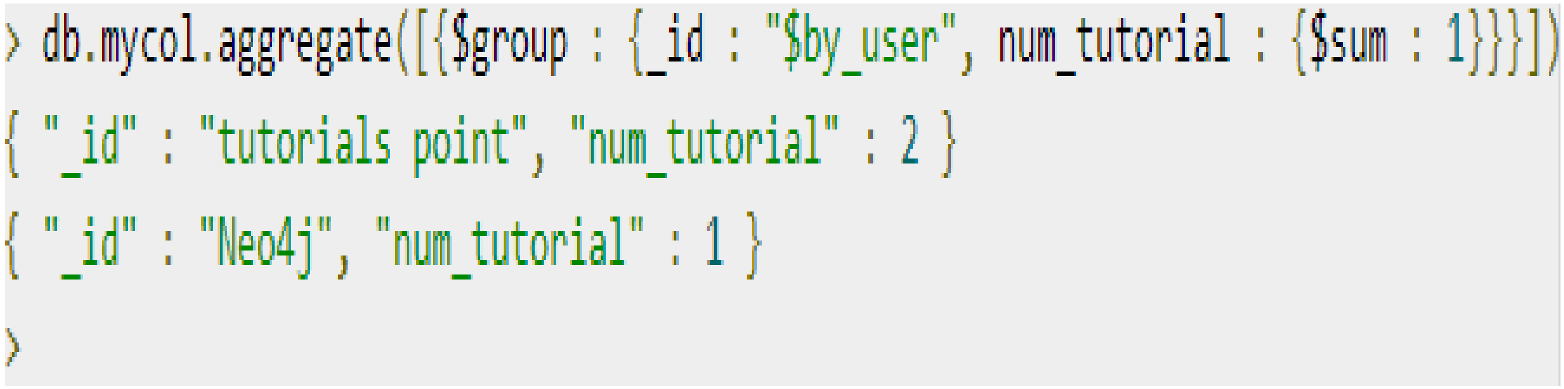
- In the collection you have the following data –



```
{
   _id: ObjectId(7df78ad8902c)
   title: 'MongoDB Overview',
   description: 'MongoDB is no sql database',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 100
},
{
   _id: ObjectId(7df78ad8902d)
   title: 'NoSQL Overview',
   description: 'No sql database is very fast',
   by_user: 'tutorials point',
   url: 'http://www.tutorialspoint.com',
   tags: ['mongodb', 'database', 'NoSQL'],
   likes: 10
},
{
   _id: ObjectId(7df78ad8902e)
   title: 'Neo4j Overview',
   description: 'Neo4j is no sql database',
   by_user: 'Neo4j',
   url: 'http://www.neo4j.com',
   tags: ['neo4j', 'database', 'NoSQL'],
   likes: 750
},
```

Aggregations

- Now from the above collection, if you want to display a list stating how many tutorials are written by each user, then you will use the following aggregate() method –



```
> db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : 1}}}])
{ "_id" : "tutorials point", "num_tutorial" : 2 }
{ "_id" : "Neo4j", "num_tutorial" : 1 }
>
```

Aggregations

- Sql equivalent query for the above use case will be select by_user, count(*) from mycol group by by_user.

- In the above example, we have grouped documents by field by_user and on each occurrence of by user previous value of sum is incremented.

Aggregations

Expression	Description	Example
$sum	Sums up the defined value from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}])
$avg	Calculates the average of all given values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}])
$min	Gets the minimum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}])

Aggregations

Expression	Description	Example
$max	Gets the maximum of the corresponding values from all documents in the collection.	db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}])
$push	Inserts the value to an array in the resulting document.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}])
$addToSet	Inserts the value to an array in the resulting document but does not create duplicates.	db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}])

Aggregations

Expression	Description	Example
$first	GGets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}])
$last	Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage.	db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}])

Parul® University

DIGITAL LEARNING CONTENT

Aggregations

- In UNIX command, shell pipeline means the possibility to execute an operation on some input and use the output as the input for the next command and so on.

- MongoDB also supports same concept in aggregation framework. There is a set of possible stages and each of those is taken as a set of documents as an input and produces a resulting set of documents (or the final resulting JSON document at the end of the pipeline). This can then in turn be used for the next stage and so on.

Aggregations

- The MongoDB aggregation pipeline consists of stages. Each stage transforms the documents as they pass through the pipeline.

- Pipeline stages do not need to produce one output document for every input document; e.g., some stages may generate new documents or filter out documents.

Aggregations

- Following are the possible stages in aggregation framework −
- $project − Used to select some specific fields from a collection.
- $match − This is a filtering operation and thus this can reduce the amount of documents that are given as input to the next stage.
- $group − This does the actual aggregation as discussed above.
- $sort − Sorts the documents.
- $skip − With this, it is possible to skip forward in the list of documents for a given amount of documents.

Partitioning/Sharding

- Breaking large datasets into smaller ones and distributing datasets and query loads on those datasets are requisites to high scalability.

- If a dataset becomes very large for a single node or when high throughput is required, a single node cannot suffice.

- We need to partition/shard such datasets into smaller chunks and then each partition can act as a database on its own.

Parul® University

DIGITAL LEARNING CONTENT

Partitioning

- Thus, a large dataset can be spread across many smaller partitions/shards and each can independently execute queries or run some programs.

- This way large executions can be parallelized across nodes(Partitions/Shards)

Why Partitioning ?

- The purpose behind partitioning is to spread data so that execution can be distributed across nodes.
- Along with partitioning, if we can ensure that every partition node takes a fair share, then at least in theory, 5 nodes should be able to handle 5 times as much data and 5 times as much read and write throughput of a single partition.
- If sharding is unfair, then a single node might be taking all the load and other nodes might sit idle. This defeats the purpose of sharding/partitioning. A good partition strategy should avoid Hot spots.

Partition by key-range

- Partition by key-range divides partitions based on certain ranges.
- For example, dividing an Organization based on the first letter of their name.
- So A-C, D-F, G-K, L-P, Q-Z is one of the ways by which whole Organization data can be partitioned in 5 parts.
- Now we know the boundaries of such partition so we can directly query a particular partition if we know the name of an employee.

Partition by key-range

- Ranges are not necessarily evenly spaced.
- As in the above example, partition Q-Z has 10 letters of the alphabet but partition A-C has only three.
- The reason behind such division is to allocate the same amount of data to different ranges.
- Due to the fact that most people have the name starting from letters between A-C, as compared to Q-Z, so this strategy will result in near equal distribution of data across the partition.

Partition by key-range

- One of the biggest benefits of such partitioning is the range queries.
- Suppose I need to find all people whose name starts from letters between R-S, then I only need to send the query to partition R-S.

Partition by key hash

- Key range partition strategy is quite prone to hot spots hence many distributed datastore employs another partitioning strategy.

- The hash value of the data's key is used to find out the partition. A good hash function can distribute data uniformly across multiple partitions.

- Cassandra, MongoDB, and Voldemort are databases employing a key hash-based strategy.

Parul® University

DIGITAL LEARNING CONTENT

Partition by key hash

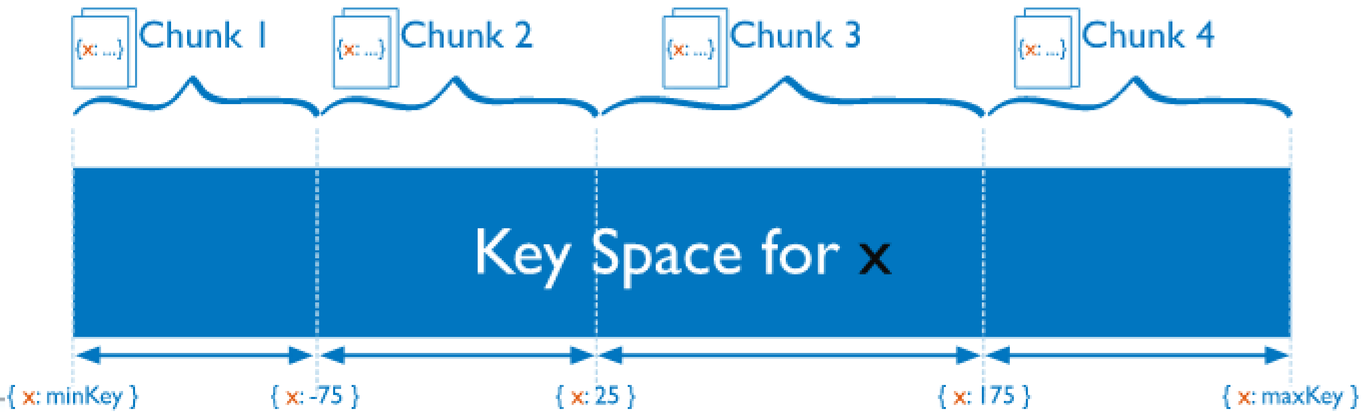
- Hash functions exercised for the purpose of partitioning should be cryptographically strong.
- Java’s Object.hashCode() is usually avoided for this purpose as such implementation can lead to a large number of hash collisions.
- Each partition is then assigned a range of key hashes (Rather than range of keys) and all keys which fall within the parameter of partitions range will be stored on that partition. Partition ranges can be chosen to be evenly spaced or can be chosen from a strategy like consistent hashing.

Partition by key hash

- Sadly with the hash of the key, we lose a nice property of key-range partitioning: efficient querying data with some ranges.
- As keys are now scattered to different partitions instead of being adjacent to a single partition.
- Hashing on key indeed reduces hot spots, however, it doesn’t eliminate it completely. In case read and writes are for the same key, all requests still end up on the same partition.

Partitioning with Chunks

- MongoDB uses the shard key associated to the collection to partition the data into chunks.
- A chunk consists of a subset of sharded data.
- Each chunk has a inclusive lower and exclusive upper range based on the shard key.



Partitioning with Chunks

- MongoDB splits chunks when they grow beyond the configured chunk size. Both inserts and updates can trigger a chunk split.

- The smallest range a chunk can represent is a single unique shard key value. A chunk that only contains documents with a single shard key value cannot be split.

Partitioning with Chunks

- Initial Chunks

- The sharding operation creates the initial chunk(s) to cover the entire range of the shard key values. The number of chunks created depends on the configured chunk size.

- After the initial chunk creation, the balancer migrates these initial chunks across the shards as appropriate as well as manages the chunk distribution going forward.

Partitioning with Chunks

- Chunk Size

- The default chunk size in MongoDB is 64 megabytes. You can increase or reduce the chunk size. Consider the implications of changing the default chunk size:

Partitioning with Chunks

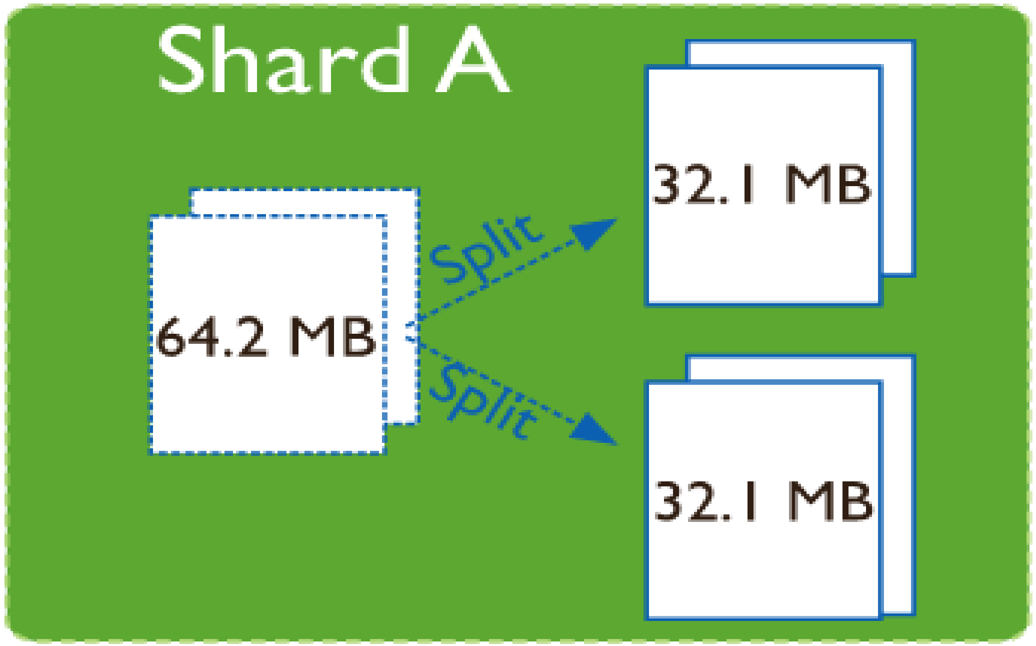
- Small chunks lead to a more even distribution of data at the expense of more frequent migrations. This creates expense at the query routing (mongos) layer.

- Large chunks lead to fewer migrations. This is more efficient both from the networking perspective and in terms of internal overhead at the query routing layer. But, these efficiencies come at the expense of a potentially uneven distribution of data.

Parul® University

DIGITAL LEARNING CONTENT

Partitioning with Chunks



Partitioning with Chunks

- Chunk Migration
- MongoDB migrates chunks in a sharded cluster to distribute the chunks of a sharded collection evenly among shards. Migrations may be either:
- Manual. Only use manual migration in limited cases, such as to distribute data during bulk inserts.
- Automatic. The balancer process automatically migrates chunks when there is an uneven distribution of a sharded collection’s chunks across the shards.

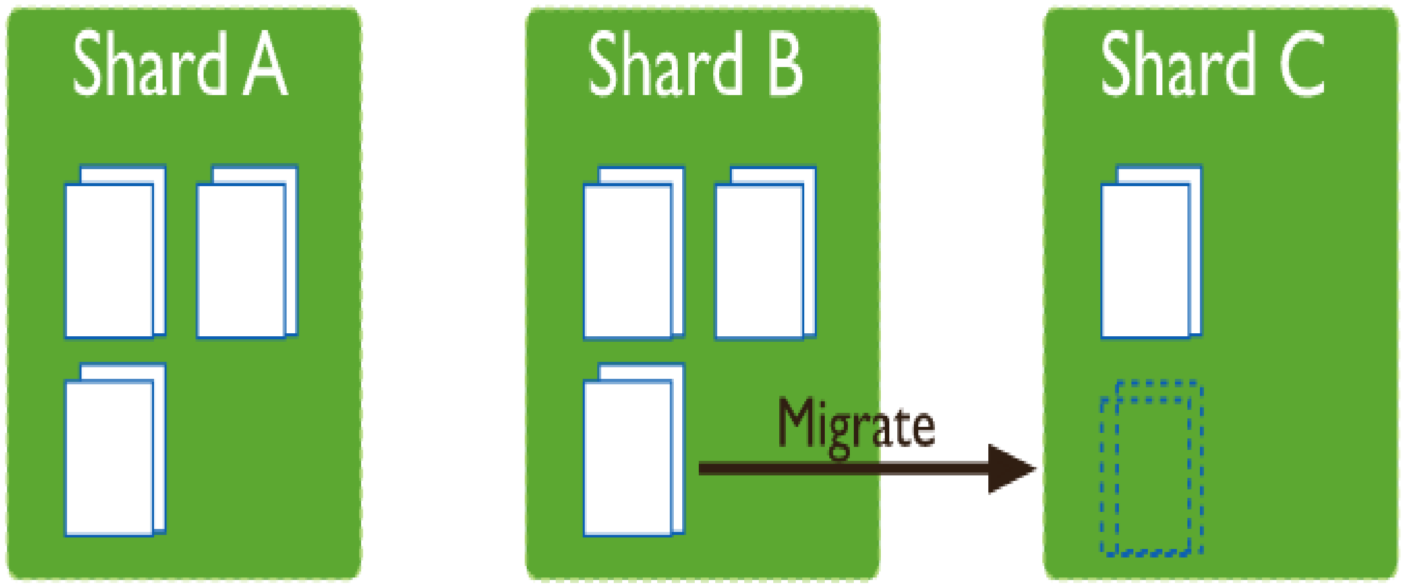
Partitioning with Chunks

- Balancing
- The balancer is a background process that manages chunk migrations.
- If the difference in number of chunks between the largest and smallest shard exceed the migration thresholds, the balancer begins migrating chunks across the cluster to ensure an even distribution of data.

Parul® University

DIGITAL LEARNING CONTENT

Partitioning with Chunks



DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in