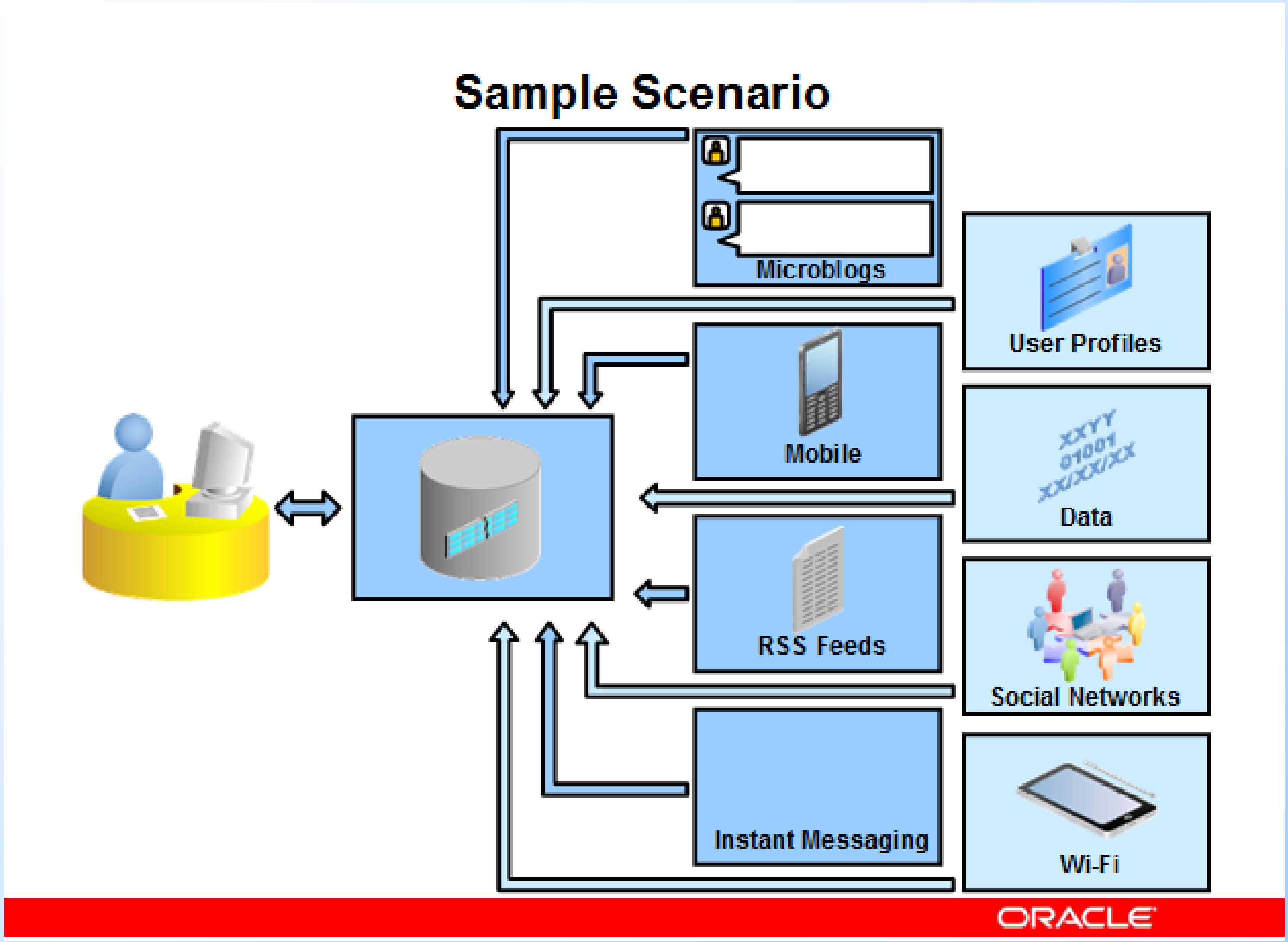
BD nestructurate

Conceptele “Big Data” și NoSQL

Acces la Oracle NoSQL cu Java API

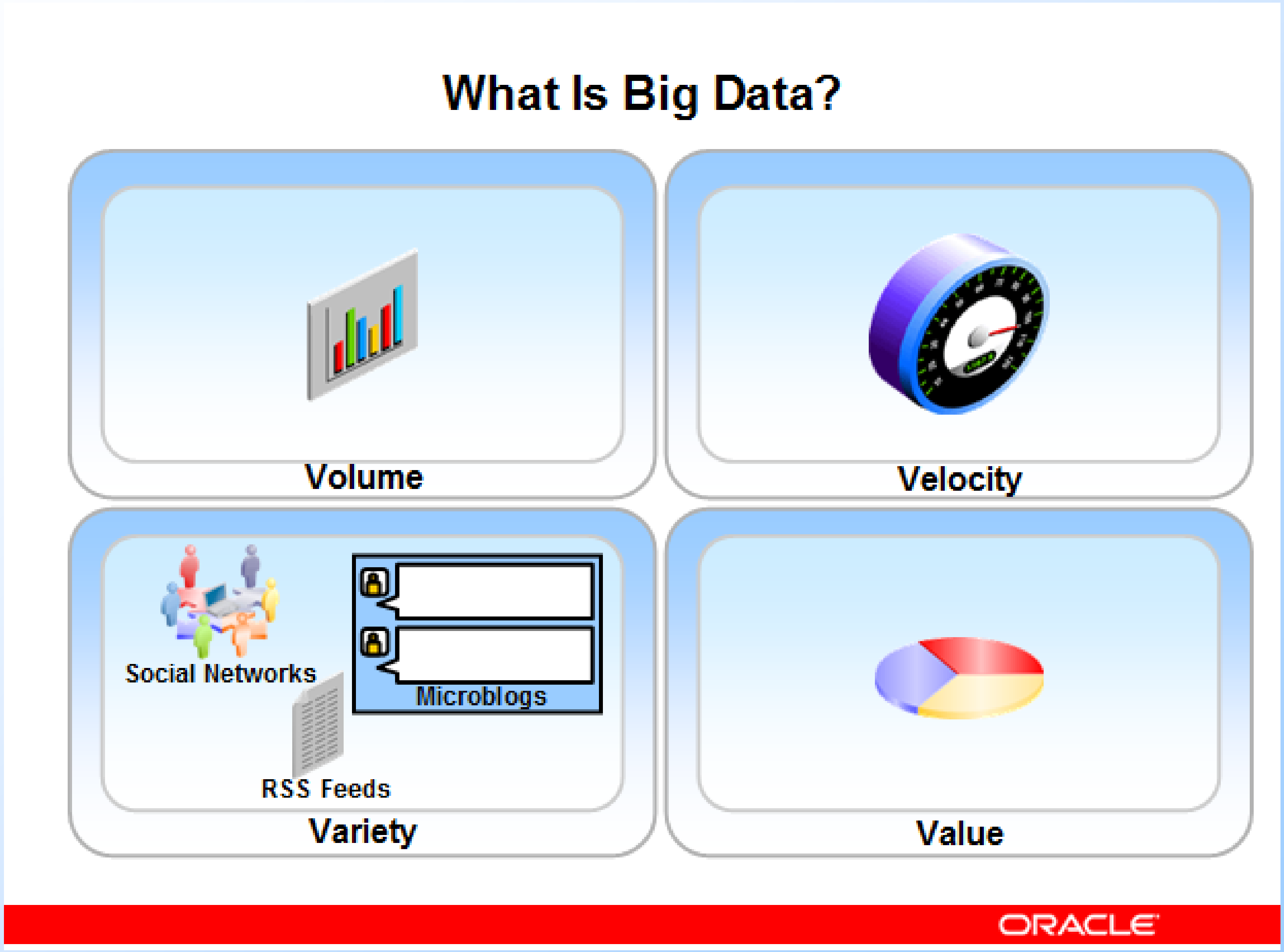
Scenariu de aplicație



Scenariu de aplicație

- Consider the following scenario: Mark developed a database application for his enterprise.
  As the application became widely used, the amount of data generated by the application increased rapidly. Mark and his team decided to scale up their database storage.
- New requirements to capture application users' profile details were passed on to Mark. He worked hard to revise the existing database schema to accommodate the new requirements.
- Now that the application has become more popular, Mark's management has asked him to capture additional details about users, such as their mobile, Facebook, and Twitter footprints on the Internet. Mark agrees that these new requirements will make the application more useful for the business. However, the volume of data to be captured and the velocity at which the data needs to be stored are estimated to be very high. Also, the data to be stored is not of high value unless it is aggregated and evaluated as a whole.
- Trying to fulfill all these requirements with the relational database will be very expensive. Moreover, with so many new requirements, Mark is now overwhelmed with the amount of work that is needed to change the existing database schemas. In addition, management wants the updated application to be rolled out as soon as possible.
- Can you relate to Mark's situation? What should he do in this situation?

Conceptul “Big Data”



What Is Big Data?

Volume

Velocity

Social Networks

Microblogs

RSS Feeds

Variety

Value

ORACLE

Copyright 2012 Oracle and/or its affilliates. All rights reserved.

Conceptul “Big Data”

- What data can be defined as “”big data”? As the term itself suggests, data sets that are very large in size are generally called *big data*. The slide shows the main characteristics of big data (referred to as “the four Vs”):
  - Volume
  - Velocity
  - Variety
  - Value
- You might be familiar with the first two characteristics. They mean that the data grows tremendously in volume with rapid velocity.
- The last two characteristics are unique to big data.
- *Variety* means that the big data datasets can be from different sources. This makes it difficult for these datasets to be stored in traditional relational databases. Because big data does not have a defined structure, it needs to be handled differently.
- *Value* means that, out of all the big data that is generated from these various sources, only a little data is of value to drive business decisions. That is, a piece of information in big data is not valuable on its own, but it becomes valuable in the aggregate.

Conceptul “Big Data”

- More people interacting with data
  - Smart phones
  - Internet

- Greater volumes of data being generated
  - Sensors
  - GPRS

Conceptul “Big Data”

- As more and more people have started using the Internet and new technologies such as smart phones, greater volumes of data are being generated all over the world.
- These data are generated in various formats. Because traditional databases could not handle these volumes of data and process them instantly, there was a need for a different approach to storing data

Conceptul “NoSQL”

- Schema-less storage
- Stores nonrelational data
- Examples of NoSQL databases:
  - Oracle NoSQL
  - Cassandra
  - Voldemort
  - MongoDB

Conceptul “NoSQL”

- A NoSQL database is a nonrelational database that does not store information in the traditional relational format.
- There is no defined schema for the data.
  - The data are stored as key-value pairs.
- The term *NoSQL* is an abbreviation of “Not Only SQL.”

Modele de date NoSQL

- A NoSQL database uses one of the following data models:
  - Key-value
  - Columnar
  - Document
  - Graph

Modele de date NoSQL

- There are four data models for NoSQL databases:
  - **Key-value:** This is the simplest data model for unstructured data. It is highly efficient and highly flexible. The drawback of this model is that the data is not self-describing.
  - **Columnar:** This data model is good for sparse data sets, grouped subcolumns, and aggregated columns.
  - **Document:** This data model is good for XML repositories and self-describing objects. However, storage in this model can be inefficient.
  - **Graph:** This is a relatively new model that is good for relationship traversal. It is not efficient for general searches.
- In this course, you learn about Oracle NoSQL Database, which belongs to the key-value data model category.

Comparație relațional/NoSQL

- RDBMS
  - High-value, high-density, complex data
  - Complex data relationships
  - Joins
  - Schema-centric
  - Designed to scale up, not out
  - Well-defined standards
  - Database-centric
- NoSQL
  - Low-value, low-density, simple data
  - Very simple relationships
  - Avoids joins
  - Schema-free, unstructured or semi-structured data
  - Distributed storage and processing
  - Standards not yet evolved
  - Application-centric and developer-centric

Use Case: Scenariu 1 Healthcare System

- A city needs a centralized healthcare system with the following requirements:
  - It should store the health records of all the people in the city across all the different hospitals.
  - Doctors should be able to use this system to understand the health history of patients.
  - The system should be able to storage different kinds of data.
  - Data that needs to be stored might change over time.

What technology would you recommend to build this application?

Use Case: Scenariu 1 Healthcare System

- This scenario is an example of big data. A NoSQL database should be used to acquire the data.
- What are some of the considerations that helped to select the back-end storage technology for the application?
  - **Data volume:** How big is the data that needs to be stored? In this scenario, supposed the city's population is twenty million. The health information is estimated to reach 5.8 petabytes.
  - **Real-time information:** The doctors should be able to retrieve information about a patient instantly. In many hospitals, doctors get less than five minutes to deal with each patient. So the response time should be immediate.
  - **Data variety:** The application should be able to store any kind of information about the patient (x-ray reports, scans, laboratory results, doctor inputs, medical bills, and so on).
  - **Data change:** Different hospitals will have different policies about the data that needs to be stored. These policies might change from time to time.

Use Case: Scenariu 2
Human Resources System

- A multinational company needs a centralized human resources system with the following requirements:
  - It should store the information of all employees in the organization (both current and former employees).
  - For each employee, it should store information such as date of hire, family details, job history, heath history, benefits received from the company, and date of resignation or retirement.
  - It should be able to store scans of important documents, employee fingerprints, voice samples, and so on.
  - Company benefits and policies applying to an employee might change from time to time.

Use Case: Scenariu 2 Human Resources System

- This scenario is an example of high-value and confidential information. A RDBMS database should be used to acquire the data.
- What are some of the considerations that helped to select the back-end storage technology for the application?
  - **Data value:** The information that this application is required to handle is of very high value. Information about an employee is always considered to be confidential and should be securely stored.
  - **Data structure:** Although this application needs to handle different kinds of data, you can still predict a structure for the information to be stored.

Use Case: Scenariu 3
Retail Marketing System

- In a new marketing strategy, you want to offer discount coupons to your customers when they are near your business.
  You will need to store:
  - Customer profiles
  - Customer purchase history
  - GPRS signals from mobile devices
  - Promotion details

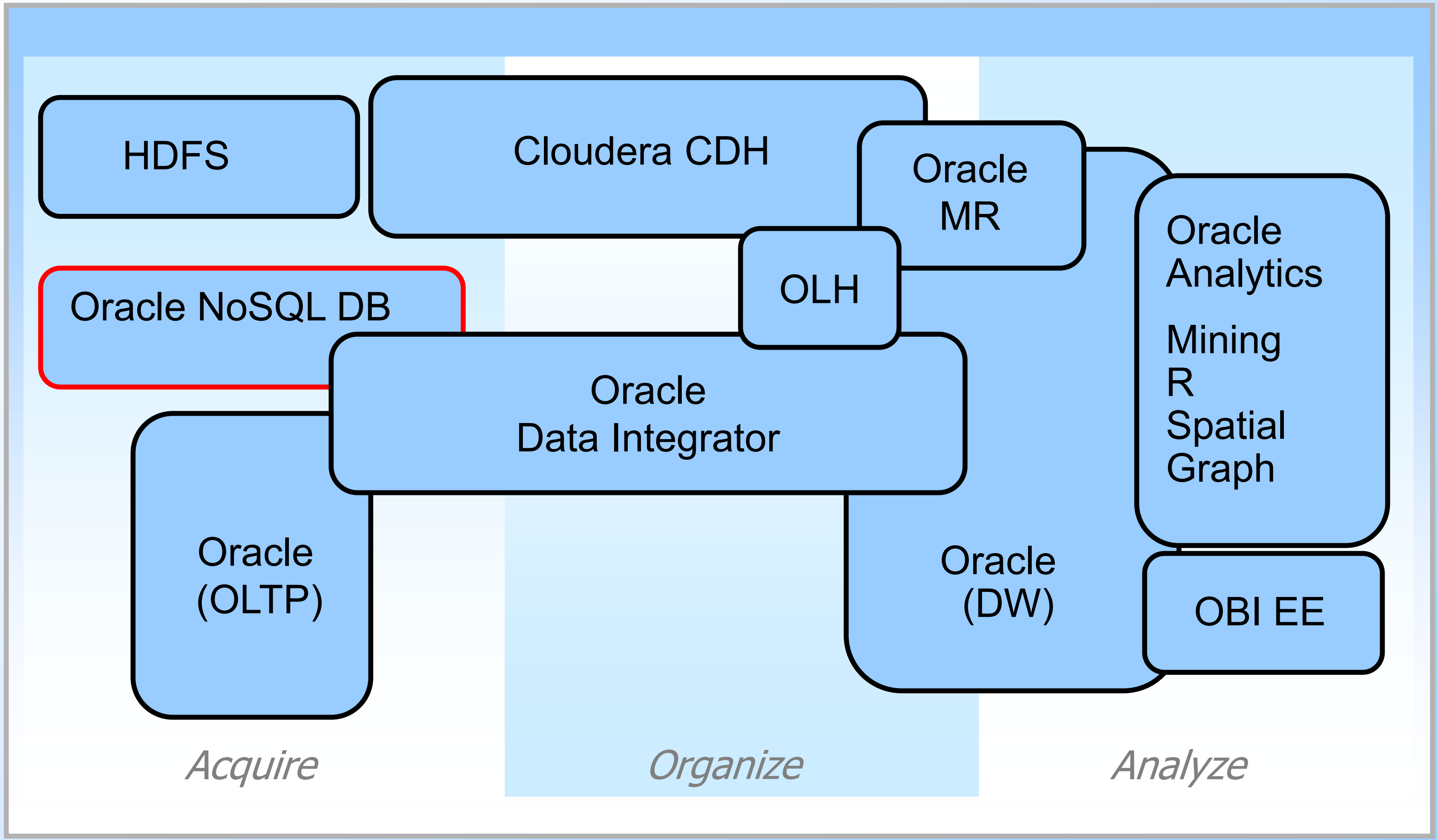
Use Case: Scenariu 3
Retail Marketing System

- In the scenario described in the previous slide, you want to send discount coupons to your customers as they approach your business. You should process all information and send the coupons as soon as the customer is near the business. If you send the coupon when the customer has finished shopping and has already reached the parking lot, it is too late. Only when the customer is approaching the business does the information become valuable.
- You will need to store many details (for example, the GPRS feed). You also need to store the promotions and offers that your business is featuring—which might change on a daily basis.
- With all these requirements in mind, you should choose a NoSQL database as the best storage option for this application.

Criterii pentru a alege NoSQL

- You should make the following analyses when deciding on an applications database technology:
  - Analyze the data to be stored.
    - High volume, low value?
      If answer is “yes,” then NoSQL is a better choice.
  - Analyze the application schema.
    - Dynamic?
      If answer is “yes,” then NoSQL is a better choice.

Oracle Big Data Solution



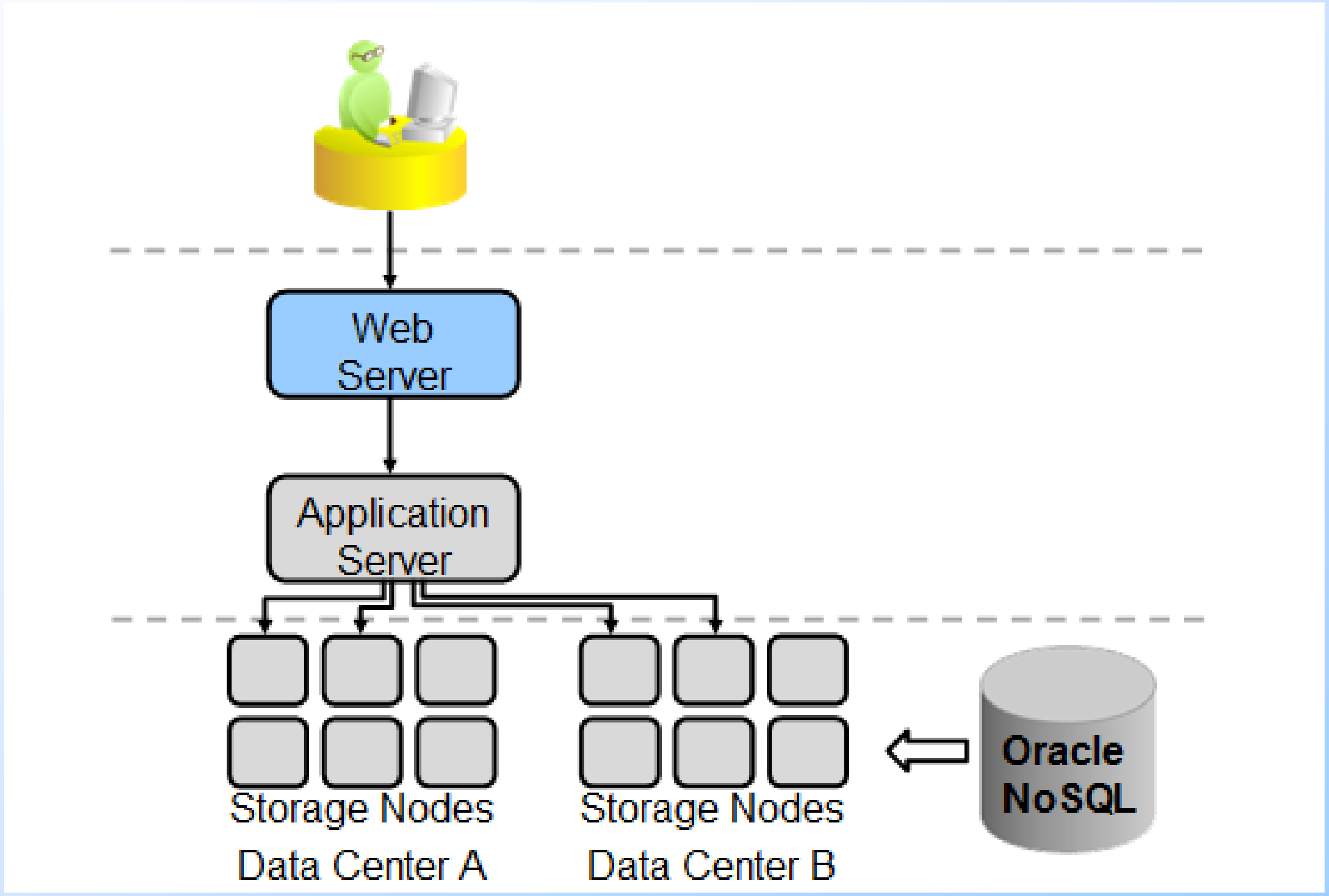
Oracle NoSQL Database

- Oracle NoSQL Database is:
  - A key-value database
  - Written in Java
  - Accessible using Java APIs
  - Built on Oracle Berkeley DB Java Edition
  - The Oracle solution to acquiring big data

Tipuri de date Oracle NoSQL

Text
Numeric
Video
Image
Voice
Spatial
XML
XML
Document

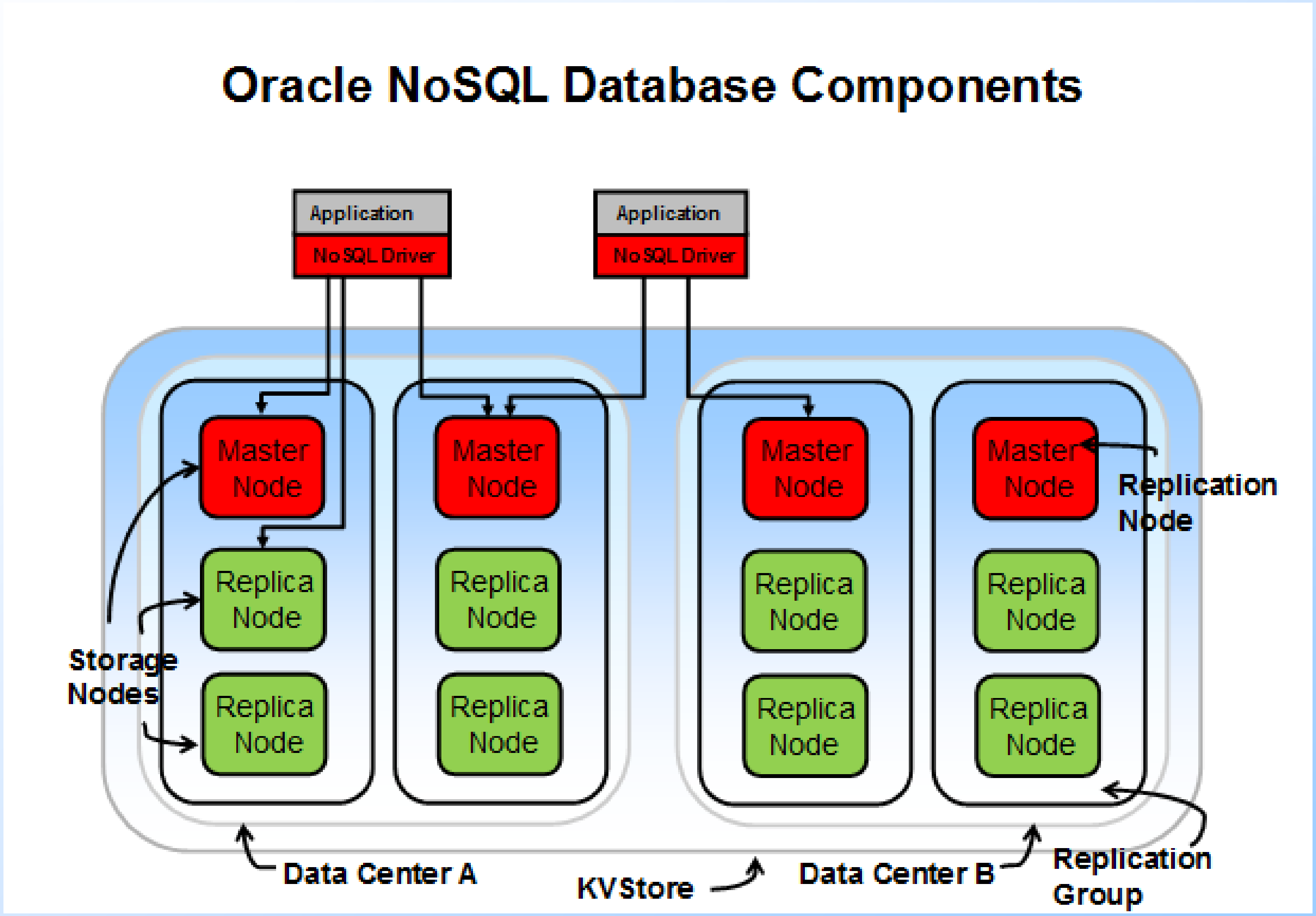
Arhitectura unei aplicații cu Oracle NoSQL



Arhitectura unei aplicații cu Oracle NoSQL

- Consider a typical web application scenario in which the application services requests across the traditional three-tier architecture: web server, application server, and database server.
- In this scenario, Oracle NoSQL Database is installed behind the application server. Oracle NoSQL Database either takes the place of the back-end database server or runs alongside the back-end database.
- Oracle NoSQL Database is installed in a set of storage nodes that may be located in different data centers.

Componente Oracle NoSQL



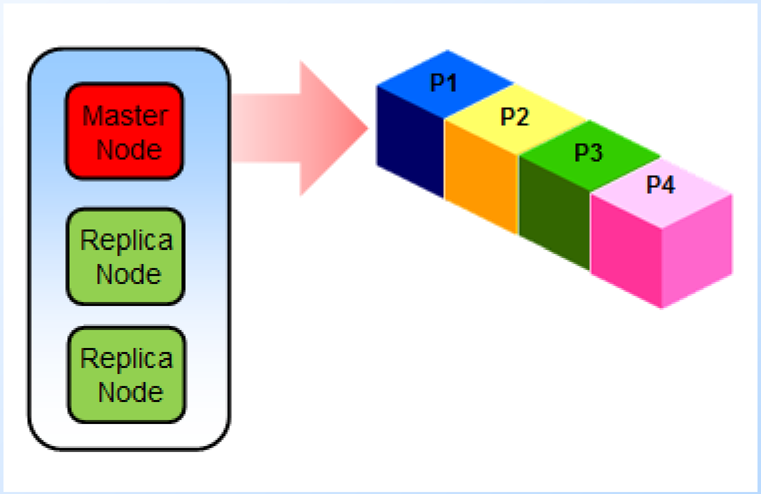
Componente Oracle NoSQL

- The main component of Oracle NoSQL Database is the Key-Value Store (KVStore). The KVStore is a collection of storage nodes that may be distributed across different data centers. A data center could be physically located at a different location than other data centers. The previous slide shows a KVStore with a number of storage nodes distributed across two data centers: A and B.
- A storage node is a physical or virtual machine with its own local storage. It is recommended that all the storage nodes within a KVStore be identical. A storage node hosts one or more replication nodes. For better performance, it is recommended that each storage node host only one replication node. In the slide example, each storage node is considered to contain one replication node.

Componente Oracle NoSQL

- A replication node is the place where the key-value pairs of data are stored. The replication nodes are organized into replication groups (also referred to as *shards*). Each replication group contains a master node and one or more replica nodes. The master node within a replication group handles all the database write operations and keeps the replica nodes updated. The replica nodes handle all the database read operations.
- To enable your application to communicate with the KVStore, you must link an Oracle NoSQL Database driver to your application. This driver is a Java library that you access from your application by using Java APIs.

Componente Oracle NoSQL

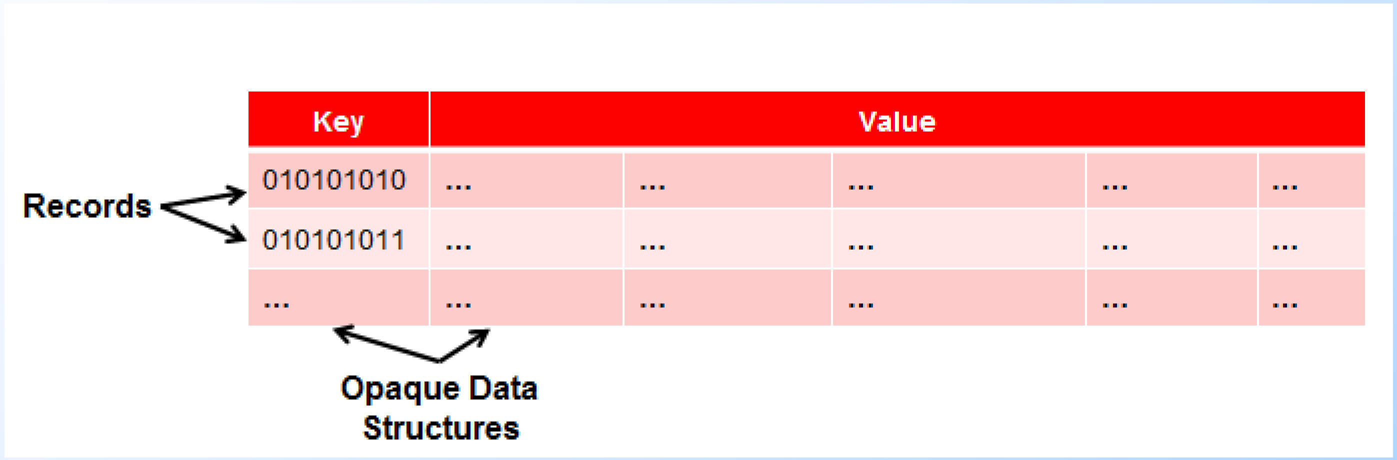


- A replication group is divided into partitions. A partition holds a key or a subset of keys. After a key is placed in a partition, it cannot be moved to a different partition. There are tools (shipped with the product) that enable you to plan how many partitions you need depending on your workload requirements.
- The task of evenly distributing the keys into available partitions is automatically handled by Oracle NoSQL Database. Also, Oracle NoSQL Database ensures that the data is read from and written to the correct partition

Acces la Oracle NoSQL

- You access the KVStore for two different needs.
  - For access to key-value data:
    - Use Java APIs.
  - For administrative actions:
    - Use the command-line interface.
    - Use the graphical web console.

Schema Structure for Oracle NoSQL Database



Key	Value				
010101010	...	...	...	...	...
010101011	...	...	...	...	...
...	...	...	...	...	...

Recall that one of the key features of Oracle NoSQL Database is that it is schema-less. This means that, in Oracle NoSQL Database, the data is not stored in fixed table-like structures.

Schema Structure for Oracle NoSQL Database

- To understand the schema structure for Oracle NoSQL Database, you can relate it to a two-column relational table with key and value columns. However, the structure of the key and value columns can take any form. Each record in Oracle NoSQL Database consists of a key-and-value pair.
- The schema for Oracle NoSQL Database is not self-describing. The application using Oracle NoSQL Database is considered to know the structure of the key and value fields.

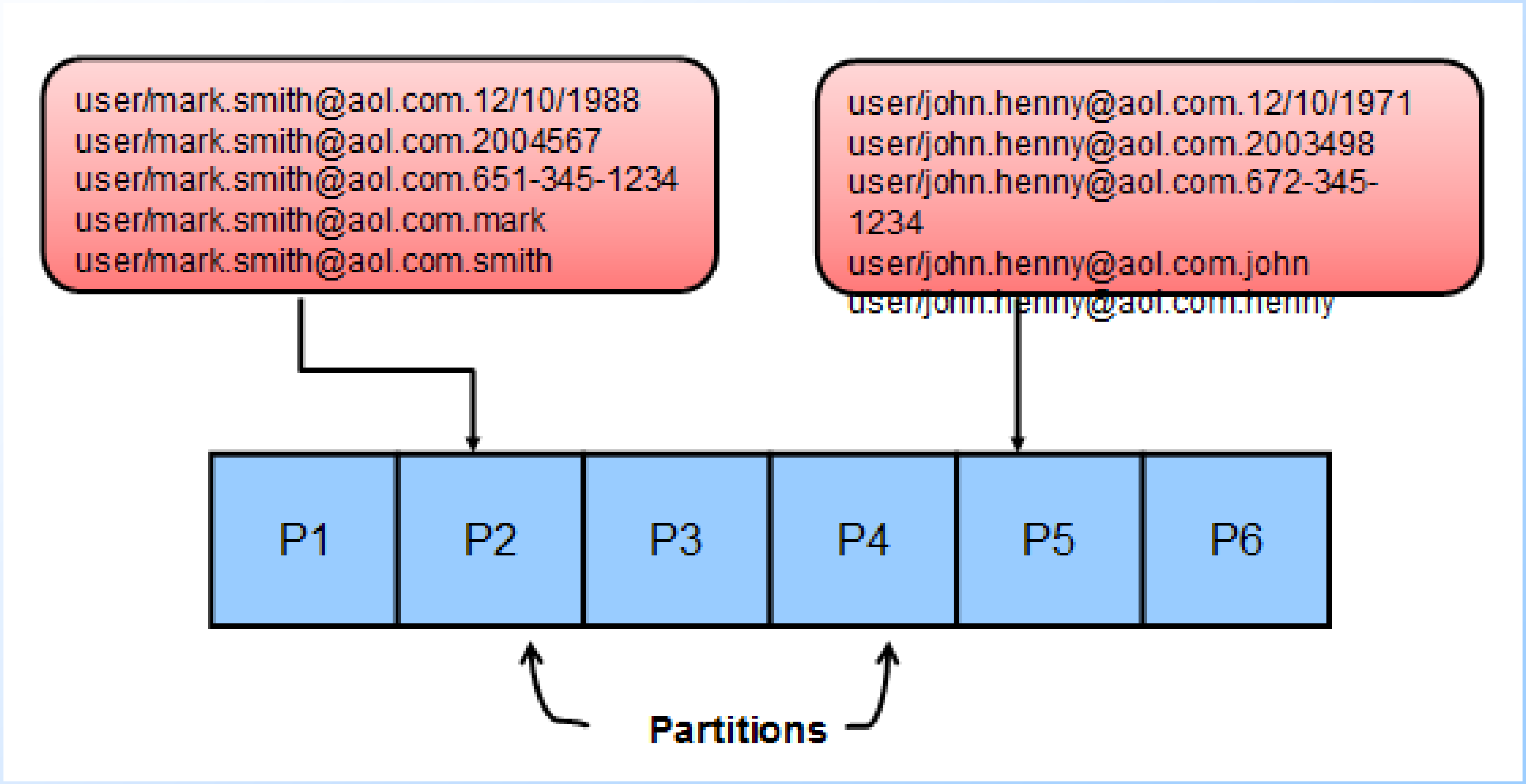
Conceptul “Key”

- The Oracle NoSQL Database key uniquely identifies each record.
  - It is application defined.
  - It is of the `String` data type.
  - It has a value attached.
  - It consists of major and minor components.

Conceptul “Key”

- You use a key in Oracle NoSQL Database to uniquely identify a record from your data. A key is a list of values of the `String` data type. The structure of a key is defined by you, and the application is assumed to know how the key is defined. Each key component is paired with a value component.
- A key can consist of major- and minor-key components. It can have more than one major or minor component. However, all keys must have at least one major component. If a key has minor components, the combination of both the minor- and major-key components uniquely identifies a record.
- How you design a key affects the performance of your application.

Exemplu Key


user/mark.smith@aol.com.12/10/1988
user/mark.smith@aol.com.2004567
user/mark.smith@aol.com.651-345-1234
user/mark.smith@aol.com.mark
user/mark.smith@aol.com.smith
user/john.henny@aol.com.12/10/1971
user/john.henny@aol.com.2003498
user/john.henny@aol.com.672-345-1234
user/john.henny@aol.com.john
user/john.henny@aol.com.henny
P1
P2
P3
P4
P5
P6
Partitions

Exemplu Key

- An application’s keys are evenly spread across the KVStore’s partitions based on the keys’ major components.
- That is, records that share the same combination of major-key components are guaranteed to be in the same partition.
- When a set of records that you want to work with are in the same partition, you can efficiently query them.
- This means that you can perform multiple operations on these records under a single atomic operation.

Exemplu Key

- The previous slide shows two sets of records that have the following record structure:
  - `user/email.birthdate`
  - `user/email.id`
  - `user/email.phone`
  - `user/email.firstname`
  - `user/email.lastname`

Exemplu Key

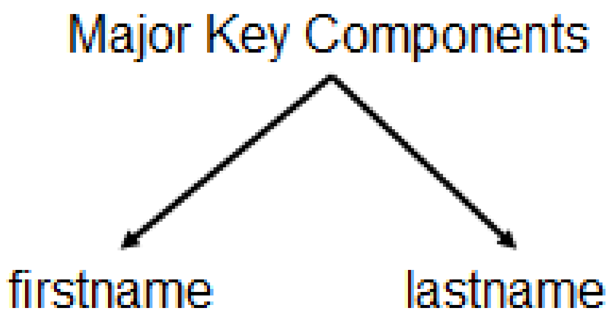
- Here, `user` is a string constant. Along with the email ID of a person, `user` and `email` form the major-key components of a record.
- These records have additional minor keys such as birth date, ID, phone number, first name, and last name.
- In this example and other examples in this lesson, a slash character (`/`) is used to separate the major-key components, and `.` is used to specify minor-key components.
- This convention is used only for the purposes of illustration.

Definirea Key

When defining a key for your application, consider the following:

- Do you want to define both major and minor-key components?
- Do you want to define one or more major-key components?
- Do you want to define one or more minor-key components?

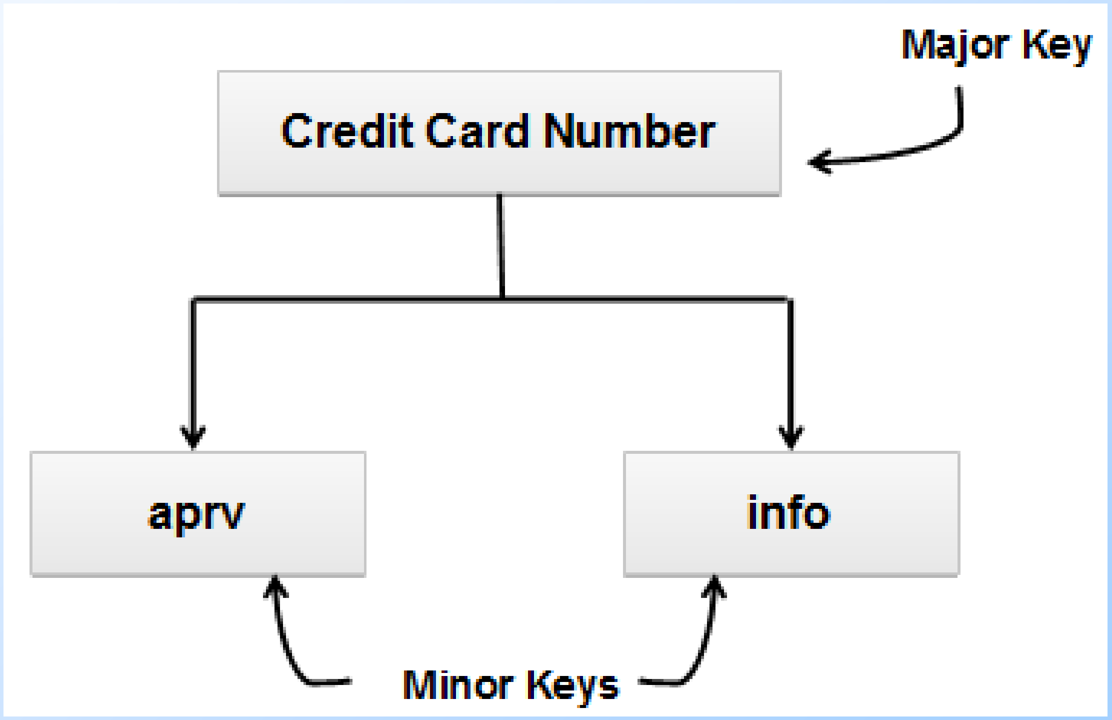
Example:



Definirea Key

- As mentioned previously in this lesson, a key is application defined. Choosing a key for your application data requires you to consider the issues listed in the previous slide.
- In some situations, defining a key with only one major-key component is sufficient. For example, for an application tracking details of all employees within an organization, the employee ID can be defined as the key component. In some situations, you might want to define a person's name as the key component. In this case, the first name and last name of a person can be defined as the two major-key components.
- It is recommended that you do not create all the records under a single major-key component. As the number of records grows in the store, performance problems can result.
- You can define minor-key components to further organize your data. Later in this lesson, you learn when to use minor-key components.

Exemplu: Credit Card Approval Application:Major Key

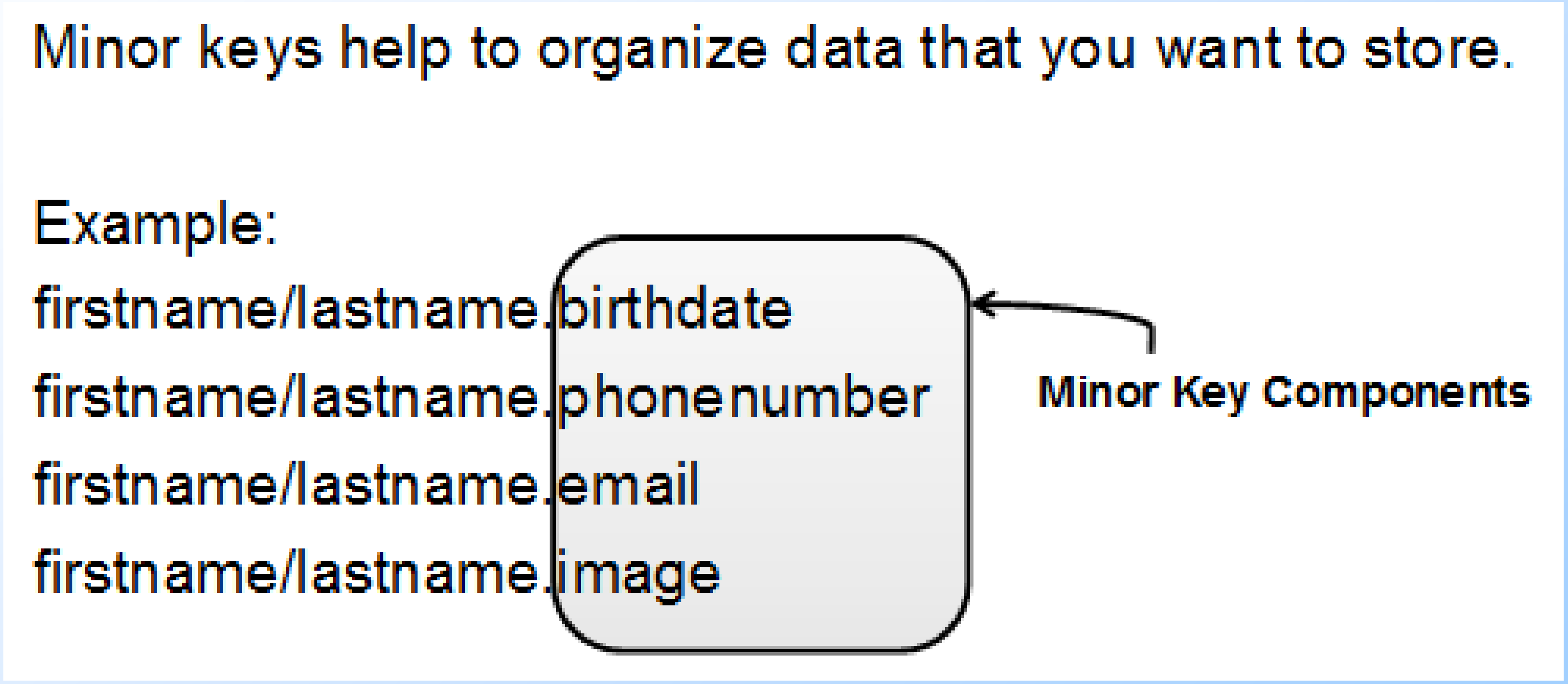


- Structure:
  - /<creditcardnumber>/-/aprv
  - /<creditcardnumber>/-/info

Exemplu: Credit Card Approval Application:Major Key

- In the Credit Card Approval application sample, there is only one major-key component: the credit card number.
- There is also a minor-key component, which takes two values: `aprv` and `info`.
- The structure of the records formed using these key components is shown in the slide.

Minor Key



Minor keys help to organize data that you want to store.

Example:

firstname/lastname.birthdate

firstname/lastname.phonenumber

firstname/lastname.email

firstname/lastname.image

Minor Key

- You can define minor-key components to further organize your data.
- For example, when you want to store a person's information, you can organize the data under minor-key components such as birth date, phone number, image, email ID, and so on.
- When designing a key, remember that there is an overheard involved with each key that the store maintains.
- Defining too many minor keys might not be the best solution for the store's performance.

Proiectarea componentelor Key

- Consider the following before finalizing your key structure:
  - What is the data (value) that you want to store?
  - What data always must be accessed together?
  - What data can be independently accessed?
  - How big is the data (value) component?
  - How frequently is the data accessed?
  - How many partitions are there in the store?

Proiectarea componentelor Key

- How you design your key components will greatly affect your application’s performance. Consider the issues listed in the previous slide while designing the key components.
- From these considerations, you will see that you must have a clear picture of the type of data you are dealing with before finalizing the key structure. You need to know what data you need to store, how large or small that data is, how frequently it will be accessed, and so on.
- It is also recommended that you have as many different major-key components as you have partitions in the store.

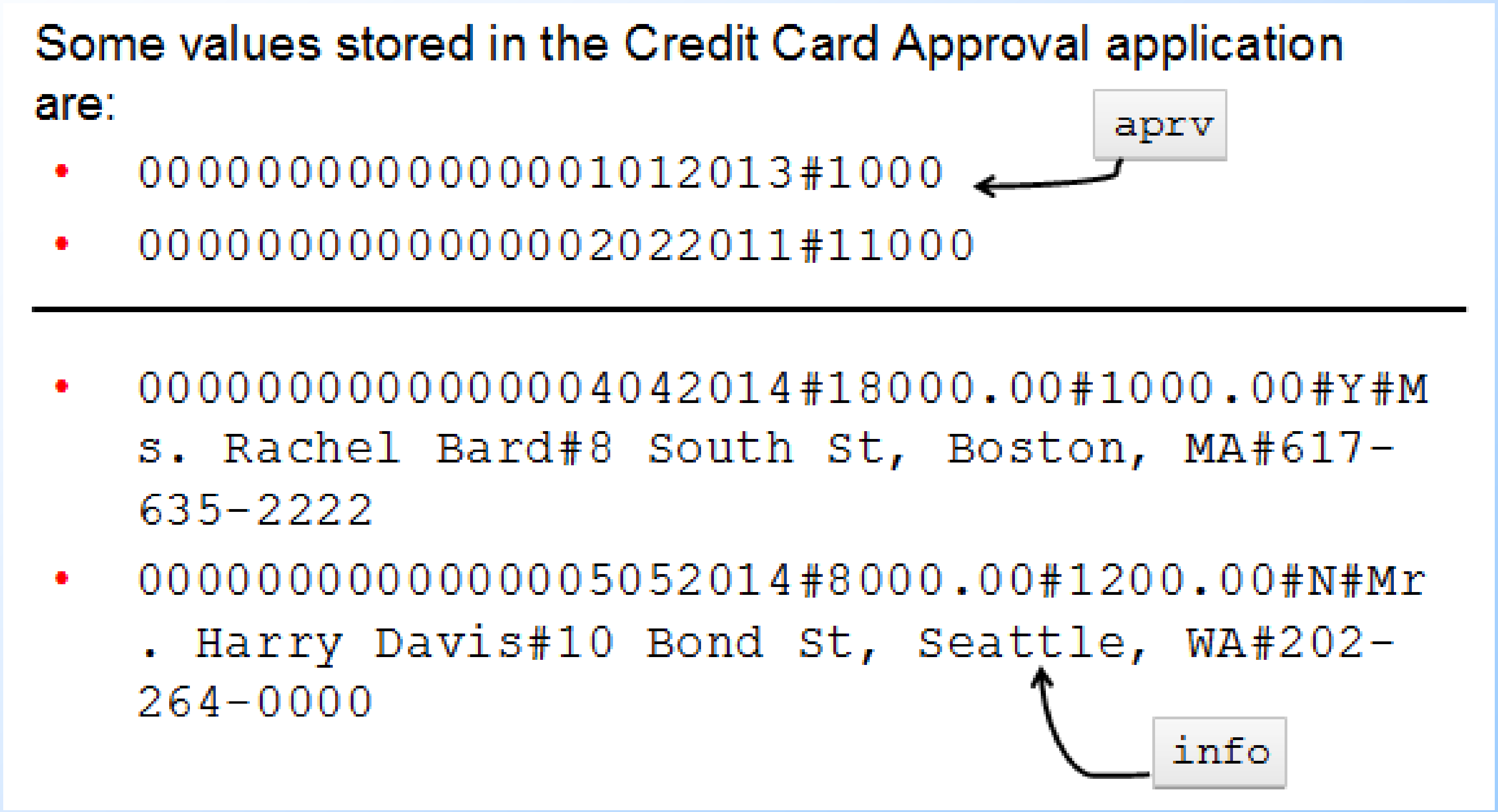
Conceptul "Value"

- In a key-value pair, the value component:
  - Is the data that you want to store and manage
  - Is a byte array
  - Is application-defined

Conceptul “Value”

- The value part of a key-value record is the data that you want to store and manage using Oracle NoSQL Database. It is stored in Oracle NoSQL Database as a byte array.
- The value component can be as large or as small as you want it to be. However, larger records take a longer time to read and write than do shorter records.

Exemplu: Credit Card Approval Application:Value



Some values stored in the Credit Card Approval application are:

- `00000000000000001012013#1000`
- `00000000000000002022011#11000`

- `00000000000000004042014#18000.00#1000.00#Y#Ms. Rachel Bard#8 South St, Boston, MA#617-635-2222`
- `00000000000000005052014#8000.00#1200.00#N#Mr. Harry Davis#10 Bond St, Seattle, WA#202-264-0000`

Proiectarea componentelor Value

- Consider the following before finalizing your value structure:
  - How large will you allow the individual records to grow?
  - How frequently will the records be accessed?
  - Strike a balance between too many small records or a small number of very large records.
  - The maximum size of value component is four gigabytes (4GB).

Aplicație cu NoSQL

- Consistency guarantees
- Durability guarantees
- Versioning

Consistency

- Consistency guarantee:
  - The policy that describes whether a record in the replica nodes can be different from the same record in the master node
- Two ranges:
  - High consistency guarantee
  - Low consistency guarantee

Consistency

- Recall that a KVStore consists of replication groups. Each replication group has a master node and one or more replica nodes. At any specific point in time, there is a possibility that a record in the replica nodes is different from the same record in the master node. This is known as a *consistency guarantee*.
- When there is a high probability that a record in the replica node is identical to the same record in the master node, the application is said to have *high consistency guarantees*.

Consistency

- Similarly, if there is a low probability that a record in the replica node is identical to the same record in the master node, the application is said to have *low consistency guarantees*.
- You can control how high you want an application's consistency guarantee to be. However, a high consistency guarantee results in slow write performance. Similarly, a low consistency guarantee results in fast write performance.

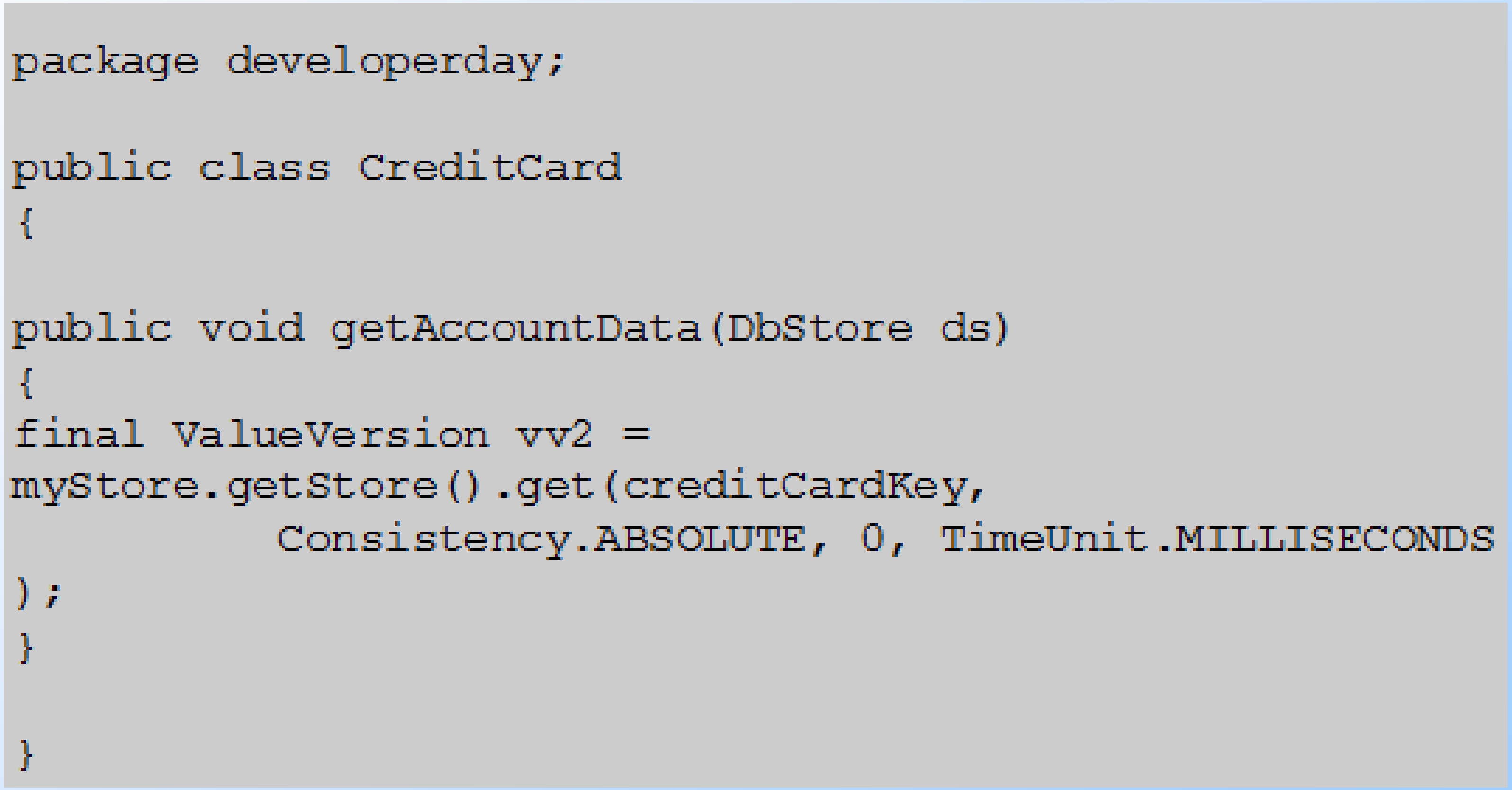
Implementarea Consistency

- You implement consistency in the following ways:
  - Set a default consistency for the entire store.
  - Set consistency for a particular operation.
  - Override the default consistency for a particular operation

Implementarea Consistency

- Oracle NoSQL Database enables you to choose how to implement consistency guarantees for your application. You can set a default consistency for the entire store. This policy will be applied to all the operations you perform on the store. In this case, you can also override the default policy by setting a different policy for an operation.
- There are different types of consistency policies that you can implement and use:
  - Predefined consistency guarantees
  - Time-based consistency guarantees
  - Version-based consistency guarantees

Exemplu Consistency



```
package developerday;

public class CreditCard
{

public void getAccountData(DbStore ds)
{
final ValueVersion vv2 =
myStore.getStore().get(creditCardKey,
          Consistency.ABSOLUTE, 0, TimeUnit.MILLISECONDS
);
}


}
```

Exemplu Consistency

- The previous slide illustrates an example from the Credit Card Approval application.
- In the `CreditCard` class, the consistency of a particular operation is specified.
- In this example, the `Consistency.ABSOLUTE` parameter is used to specify that the record should always be consistent with the master node.

Durability

- Durability guarantee:
  - The policy that describes the persistence of data in a store in case of failure in the store
- Two ranges:
  - High durability guarantee
  - Low durability guarantee

Durability

- Write operations in a KVStore are performed on master nodes. These write operations might be the creation of new records, updates of existing records, or the deletion of records.
- The master node is responsible for ensuring that the write operation has made it to stable storage.
- It is also responsible for transmitting the write operation to the replica nodes.
- You can set the rules or policies for when a write operation on the master node is said to be "complete." This is called a *durability guarantee*.

Durability

- A *high durability guarantee* means that in a store failure such as a power outage or disk crash, the write operation is still retained.
- Likewise, a *low durability guarantee* means that in a failure, the write operation will be lost.
- With higher durability guarantees, the write performance of the store becomes slower.

Implementarea Durability

- You implement durability by using one of the following:
  - Synchronization-based durability policies
  - Acknowledgement-based durability policies

Implementarea Durability

- A write operation consists of the following operations:
  1. Data in the in-memory cache is modified.
  2. Data is written to the file system’s data buffers.
  3. Buffer data is synchronized to stable storage.

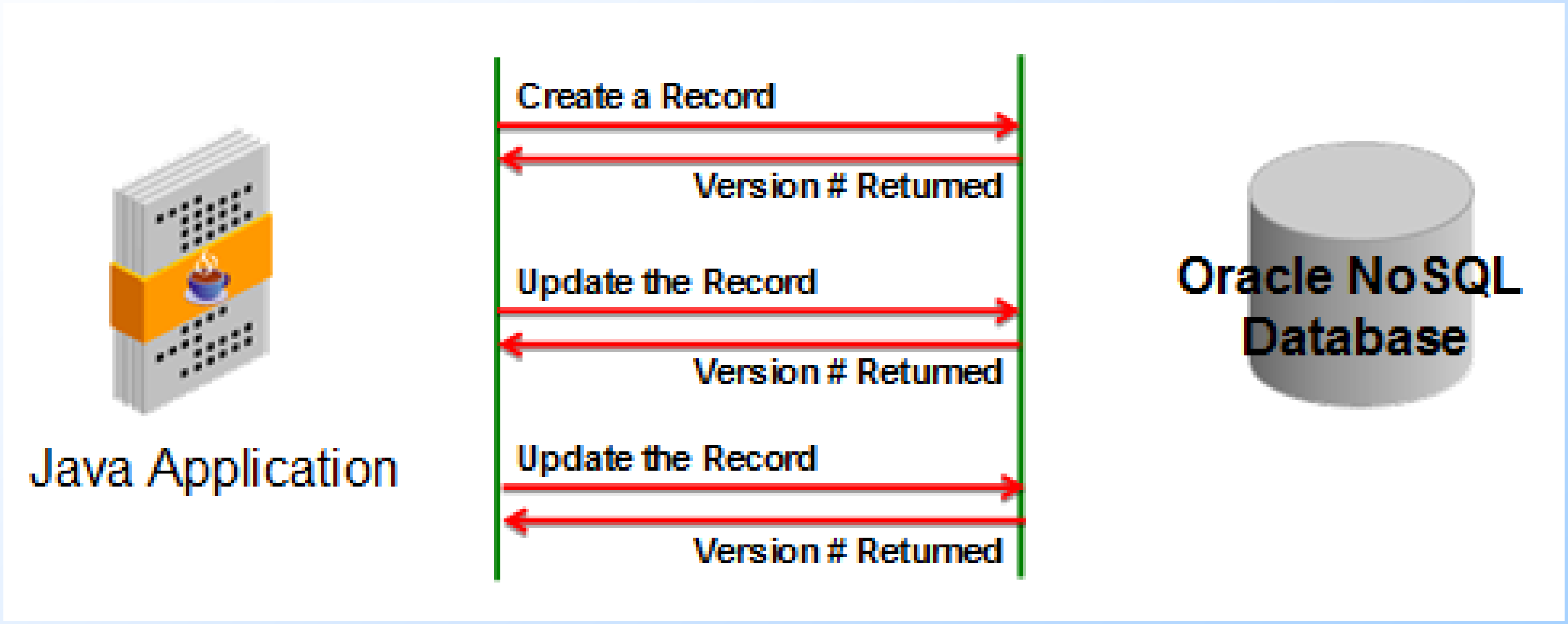
Implementarea Durability

- When using synchronization-based durability policies, you can control which of these operations the master node will wait to complete before it considers the write process to be completed.
- The master node's performance will become slower if it must wait to complete more operations.

Implementarea Durability

- When you use the acknowledgment-based durability policy, you are specifying that the master node must wait for acknowledgements from the replica nodes before considering the write operation to be completed.
- You can set the master node to wait for acknowledgements from all replica nodes, no replica nodes, or a majority of replica nodes.
- If the master node requires more acknowledgements, its write performance will become slower.

Versioning



- With this Oracle NoSQL Database feature, a version token is maintained each time a record is updated.

Versioning

- Oracle NoSQL Database automatically maintains a version number when a record is initially inserted and each time it is updated. This version information is important for two reasons:
  - When performing an update or a delete, you might want to perform the operation only if the record's value has not changed.
  - When performing a read operation, you might want to ensure that you read the value that was previously written.

Exemplu Versioning

```
public class CreditCard
{
public boolean approveCharge(double amount, DbStore dbs)
{
boolean updateDone = false;
while (!updateDone) {
final ReturnValueVersion oldValueVersion =
new ReturnValueVersion
(ReturnValueVersion.Choice.ALL);

final Version newVersion = dbs.getStore().putIfVersion
   (creditCardKey, newValue, getVersion(),
oldValueVersion, null, 0, null );
}
 }
}
```

Exemplu Versioning

- The previous slide illustrates an example from the Credit Card Approval application. It shows how to use the version attributes in the `CreditCard` class.
- Here, the credit card balance is updated if the version has not changed. If the version has changed, the existing `Value` and `Version` details are returned and used to retry the update.

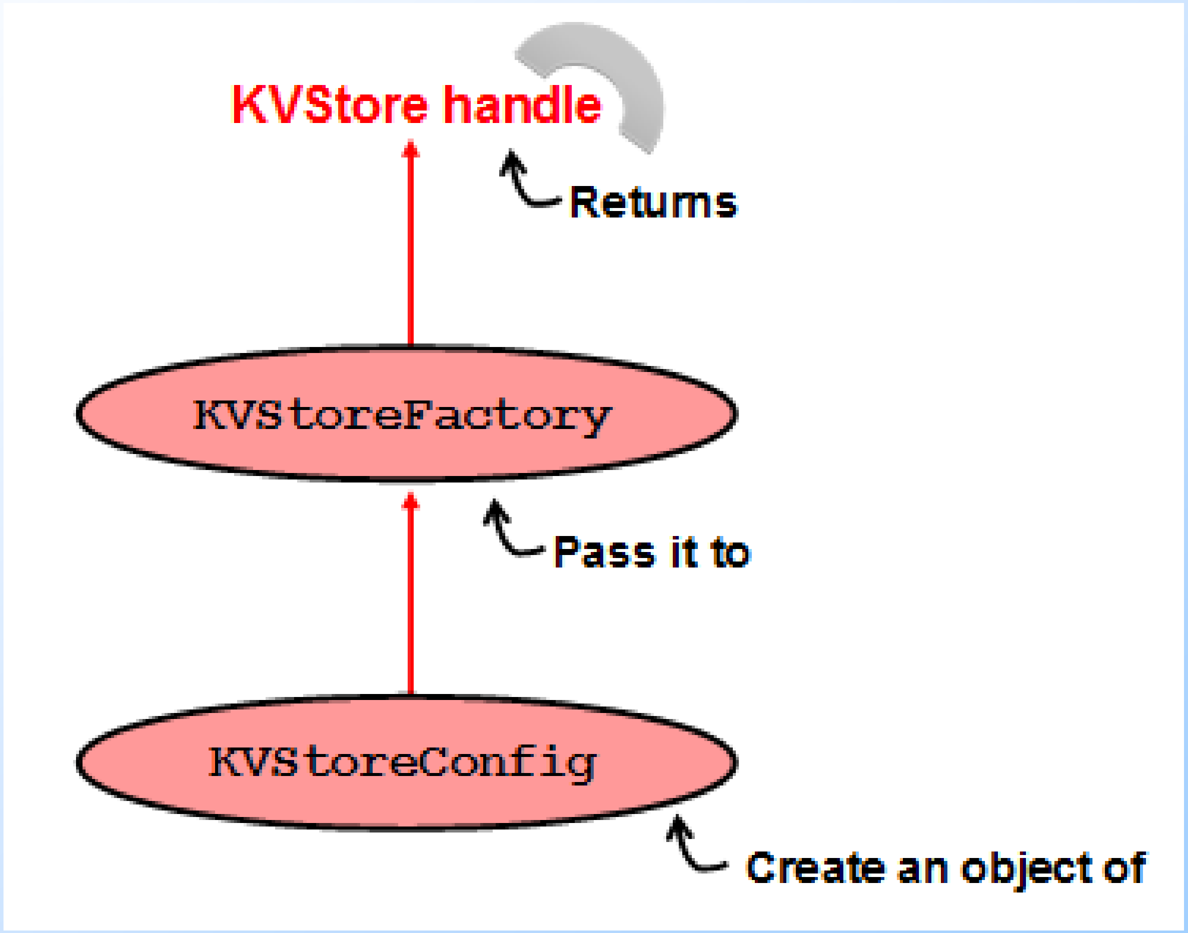
Accesarea KVStore

- KVStore Handle
  - Is a resource that controls access to Oracle NoSQL Database
  - Is used to open and close an already running store
  - Is required to perform any operation on the store

Accesarea KVStore

- Any kind of access to Oracle NoSQL Database is performed by first obtaining a KVStore handle. The KVStore handle is a resource that is required to access the KVStore and is also used to open and close an already-running store.
- You need to access the KVStore whenever you perform the following operations in Oracle NoSQL Database:
  - Create
  - Read
  - Update
  - Delete

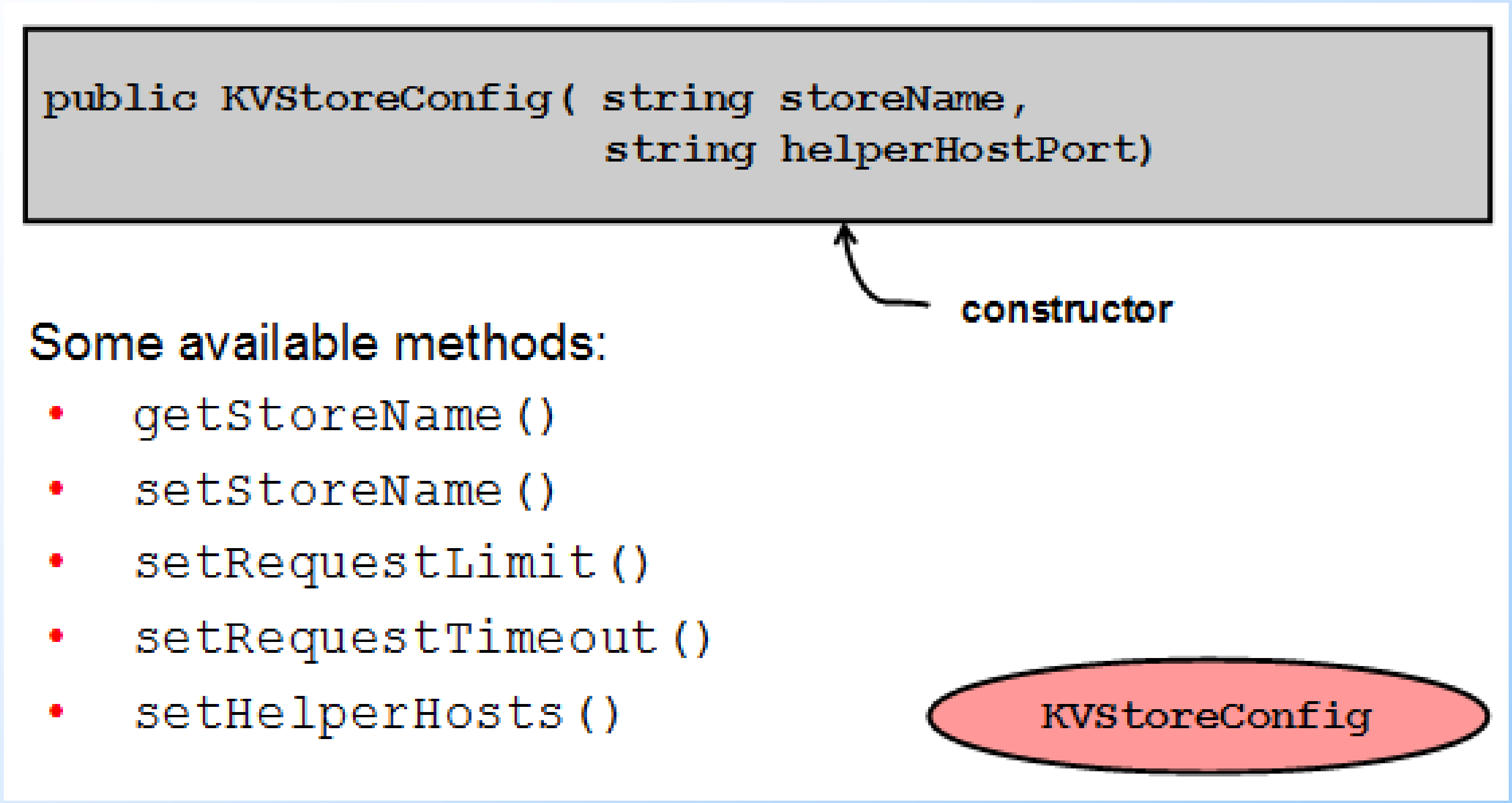
Crearea KVStore handle



Crearea KVStore handle

- To create a KVStore handle, you use the `KVStoreFactory` and `KVStoreConfig` classes. You first create an object of the class `KVStoreConfig`. You then pass this `KVStoreConfig` object to the `KVStoreFactory` class, which returns the KVStore handle.
- When a KVStore handle is obtained, the store is automatically opened.

Utilizarea KVStoreConfig



```
public KVStoreConfig( string storeName,
                      string helperHostPort)
```

Some available methods:

- `getStoreName()`
- `setStoreName()`
- `setRequestLimit()`
- `setRequestTimeout()`
- `setHelperHosts()`

Utilizarea KVStoreConfig

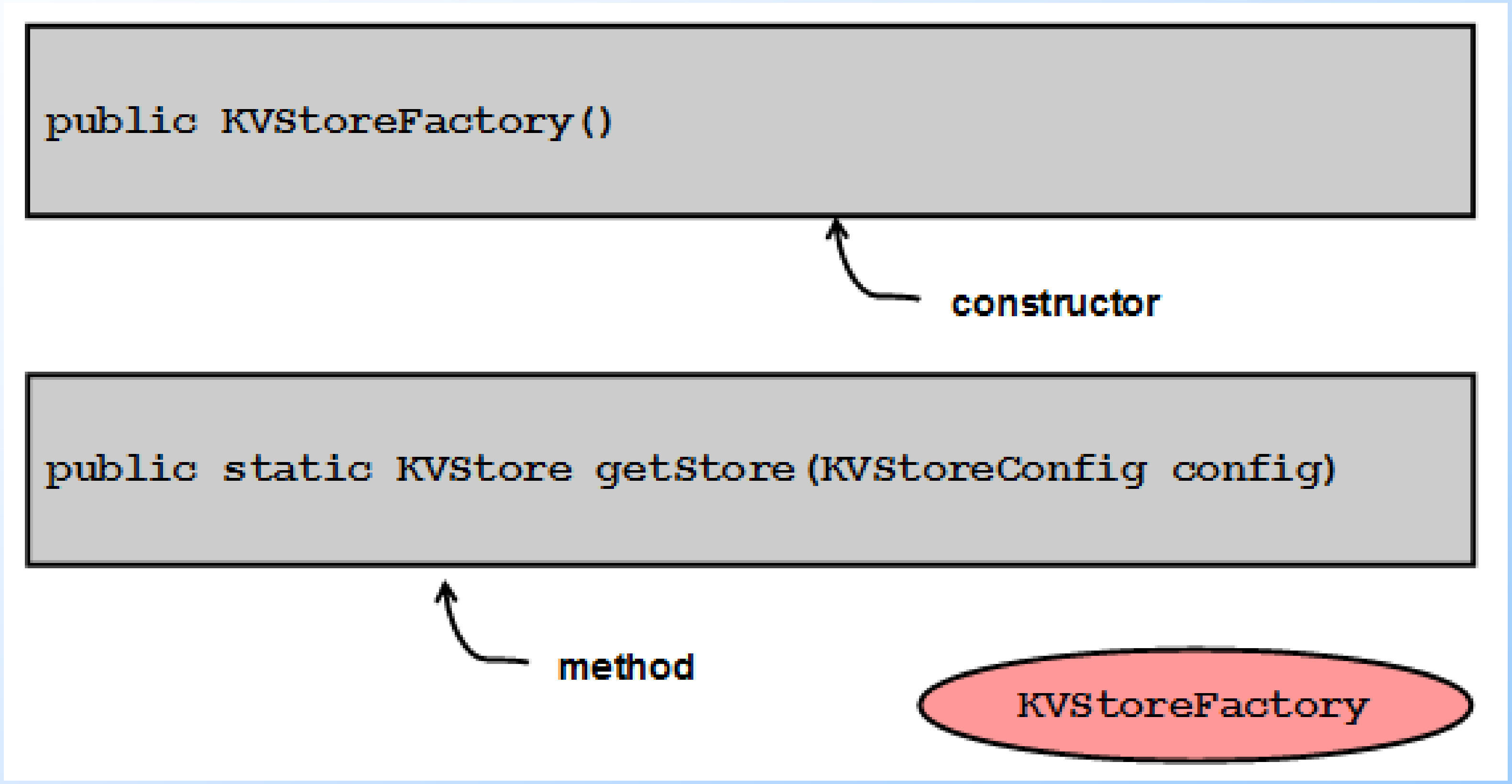
- The `KVStoreConfig` class describes a KVStore handle. The previous slide shows the `KVStoreConfig` constructor, which is used to create an instance of the `KVStoreConfig` class.
- The `storeName` parameter specifies the name of the KVStore. It must be entirely uppercase or lowercase letters and digits.
- The `helperHostPort` parameter is a set of string values that specify the host name and port of an active node in the KVStore. Each string value must be in the format `hostname:port`. You must specify at least one helper host name and port.
- The `KVStoreConfig` class has a list of methods that you can use to get and set Store Name, Durability, Consistency, Request Timeout, and other parameters. For a complete list of these methods and their definitions, view the Javadoc from the installation documents.

Exemplu: Utilizarea KVStoreConfig

```
KVStoreConfig kconfig = new KVStoreConfig("teachStore",
                                    "localhost:5000");
```

- The slide shows an example of using the `KVStoreConfig` class. Here, an object of type `KVStoreConfig` is created and initialized.
  - **`kconfig`** is the object name.
  - **`teachStore`** is the name of the KVStore, which must already be running.
  - **`localhost:5000`** is the host name and port number of the active node in the KVStore.

Exemplu: Utilizarea KVStoreFactory



```
public KVStoreFactory()
```

```
public static KVStore getStore(KVStoreConfig config)
```

Exemplu: Utilizarea KVStoreFactory

- The `KVStorefactory` class is a static class. It has one method, which is called `getStore`.
- The `getStore` method takes a `KVStoreConfig` object as the input parameter and returns the KVStore handle to the store, which is specified in the `KVStoreConfig` object.

Exemplu: Utilizarea KVStoreFactory

```
KVStore kvstore = KVStoreFactory.getStore(kconfig);
```

- The slide shows an example of using the `KVStoreFactory` class. Here, an object of type `KVStore` is created and initialized.
  - **`kvstore`** is the object name. The value for this object is obtained by calling the `getStore` method of the `KVStoreFactory` class and passing an object of type `KVStoreConfig`.
  - **`kconfig`** is the object that is passed to the `getStore` method.

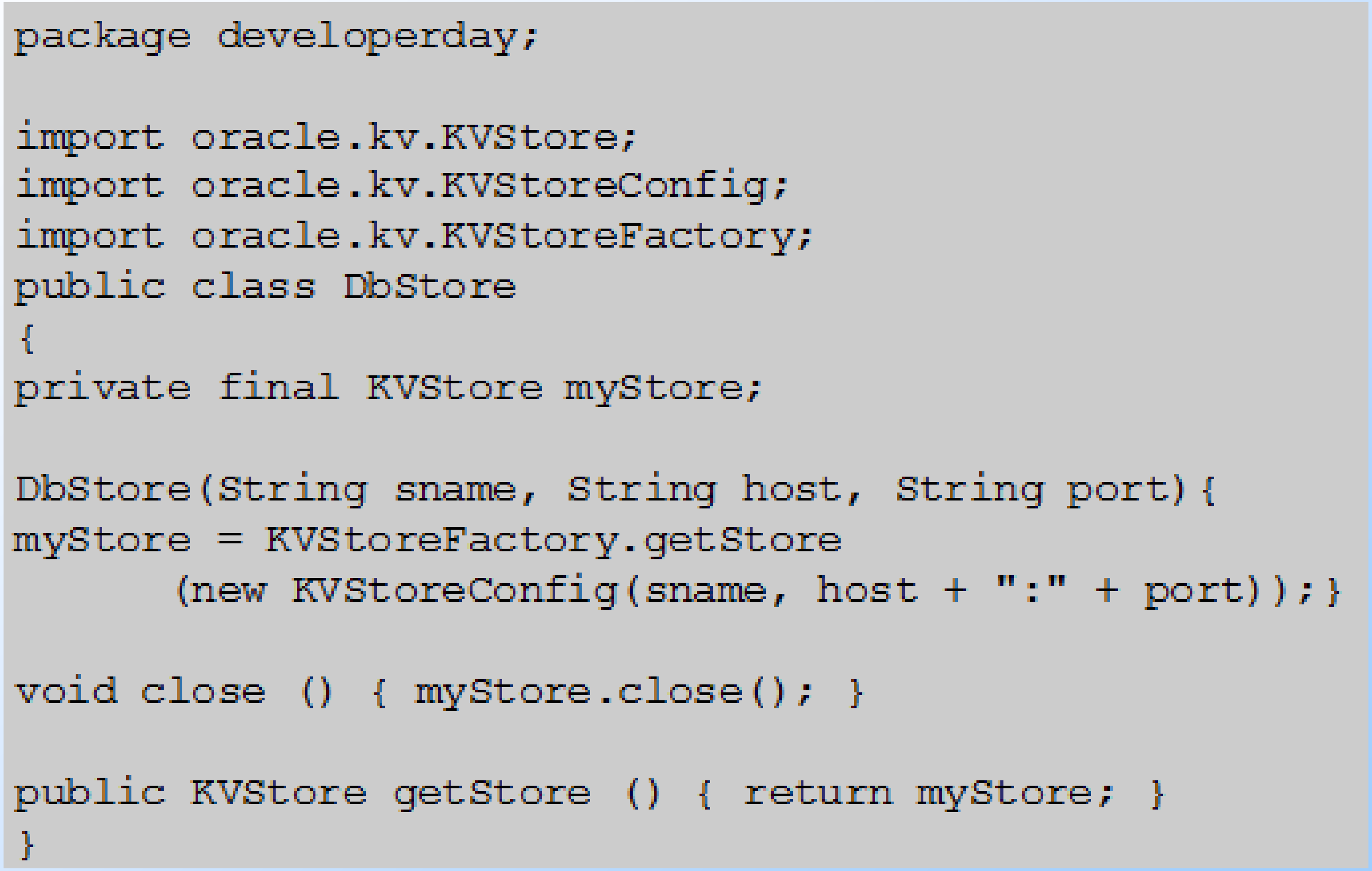
Java Packages for Creating a KVStore Handle

- The following packages should be imported to a Java class before creating a KVStore handle:
  - oracle.kv.KVStore
  - oracle.kv.KVStoreConfig
  - oracle.kv.KVStoreFactory

Exemplu: Crearea KVStore Handle

```
package teach;

import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;

public class createStoreHandle
{
KVStore myStore;
public static void main(String args[])
{
KVStoreConfig kconfig = new
    KVStoreConfig("teachStore","localhost:5000");
KVStore myStore = KVStoreFactory.getStore(kconfig);
}
}
```

Credit Card Approval Application

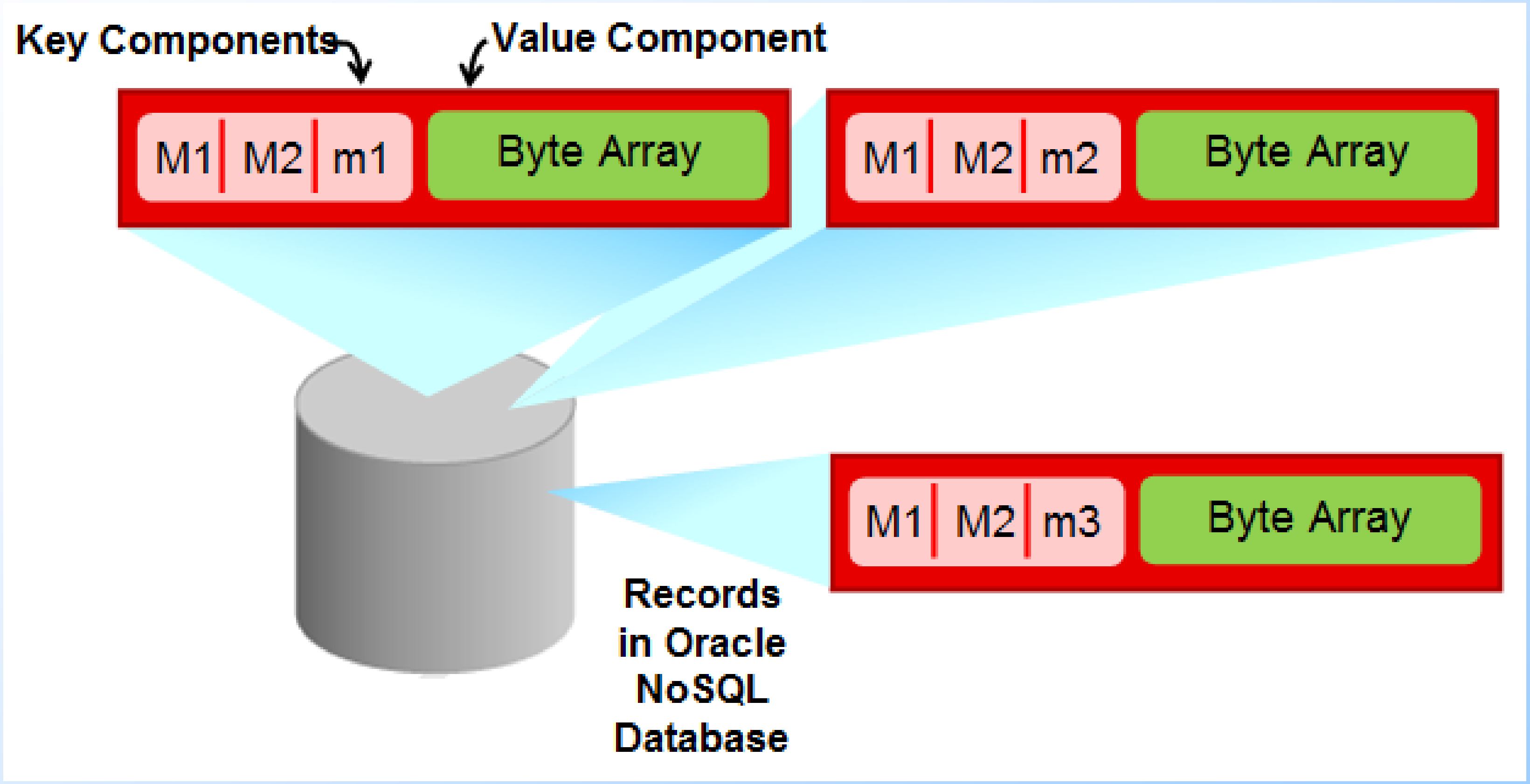


```
package developerday;

import oracle.kv.KVStore;
import oracle.kv.KVStoreConfig;
import oracle.kv.KVStoreFactory;
public class DbStore
{
private final KVStore myStore;

DbStore(String sname, String host, String port){
myStore = KVStoreFactory.getStore
      (new KVStoreConfig(sname, host + ":" + port));}

void close () { myStore.close(); }

public KVStore getStore () { return myStore; }
}
```

Credit Card Approval Application

- The `DbStore` class of the Credit Card Approval application contains the code to create a KVStore handle.
- `myStore` is the variable of `KVStore` type. The value for this variable is set using a constructor. The constructor takes the name of the store whose handle is to be obtained along with the host name and port number as input parameters. In the constructor, the `KVStoreFactory` and `KVStoreConfig` classes are used to obtained the KVStore handle.
- The `DbStore` class also has two methods:
  - **`getStore`**: Returns the `myStore` handle to the calling method
  - **`Close`**: Closes the store and automatically releases all resources

Componentele unei înregistrări



Componentele unei înregistrări

- Recall that a record consists of a key component and a value component. Also, a key component can consist of major and minor components. If a record consists of major and minor components, each unique combination of major and minor components results in a single record.
- For example, the previous slide illustrates a case where there are two major components and one minor component with three different values. In this lesson, you learn how to construct a record with the structure as shown in the slide. The major components are a person’s first and last names. The three values of the minor component are `info`, `image`, and `voice`.

Creare componente Major-Key

- To create major-key components in a Java program:
  1. Create an ArrayList of the `String` data type, named `majorPath`.
  2. Add the major-key component values to this ArrayList.

```
List<String> majorPath = new ArrayList<String>();
...
majorPath.add("Smith");
majorPath.add("Bob");
```

Creare componente Major-Key

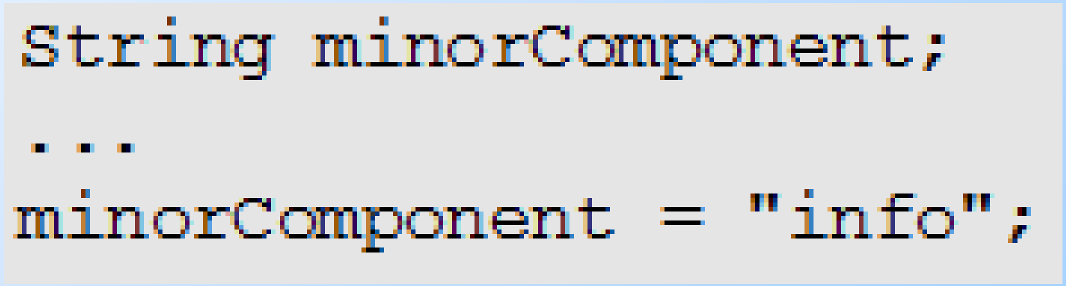
- The simplest way to create major components is to define an array of the data type `String`.
  It is mandatory that you define the data type as `String`, because the key component in Oracle NoSQL Database is restricted to the `String` data type.
- **Note:** You use an ArrayList here because the major key has two components. In a case where the major key has only one component, you can consider using a simple `String` data type.

Creare componente Major-Key

- As a naming convention best practice, you can use `majorPath` as the name of the array to store the major-key components. After the array is defined, you can add the major-key component values to the array.
- The slide example adds `Smith` and `Bob` to the the `majorPath` array.

Creare componente Minor-Key

- To create minor-key components in a Java program:
  1. Create a variable of the `String` data type, named `minorComponent`.
  2. Set the minor-key component value to this variable.



```
String minorComponent;
...
minorComponent = "info";
```

Creare componente Minor-Key

- Creating the minor components is similar to creating the major components. You should define a variable of the `String` data type.
- **Note:** If the minor key has more than one component, you can consider storing the key values in an ArrayList of `String` data type.
- Again, as a best practice, you can name the variable `minorComponent`.
- The slide example defines a `String` variable called `majorComponent` and assigns the value `info` to this variable.

Creare Key pentru înregistrare

```
Key myKey = Key.createKey(majorPath,
                          minorComponent);
```

- After you have defined the major- and minor-key components, use the `createKey` method to create the key. The slide shows an example of using the `createKey` method.
  - **`myKey`** is a variable of type `Key`.
  - **`majorPath`** contains the major-key component values.
  - **`minorComponent`** contains the minor-key component values.

Creare Value pentru înregistrare

```
List<String> data = new ArrayList<String>();

data.add("35");
data.add("male");
data.add("developer");

Value myValue = Value.createValue(data.getBytes());
```

- The value component of a record is created by using a `createValue` method. Before using this method, create the data content of the record. Either you create a variable of the `String` data type and store the data in it, or you can create an ArrayList and add all the data values to the array.
- The slide example creates an ArrayList of the `String` data type called `data`. The data values are added to this ArrayList.
- The `createValue` method accepts values of the `byte` data type and creates a value that can be stored in a KVStore.

Required Java Packages for Constructing a Record

- The following packages should be imported to a Java class before constructing a record:
  - java.util.ArrayList
  - oracle.kv.Key
  - oracle.kv.Value

Exemplu: Construire înregistrare

```
public class createRecord
{
public static void main(String args[])
{
List<String> majorPath = new ArrayList<String>();
String minorComponent;
String data;
majorPath.add("Smith");
majorPath.add("Bob");
minorComponent = "info";
Key myKey = Key.createKey(majorPath,
                          minorComponent);
data="Male, 35, Developer";
Value myValue = Value.createValue(data.getBytes());
}
}
```

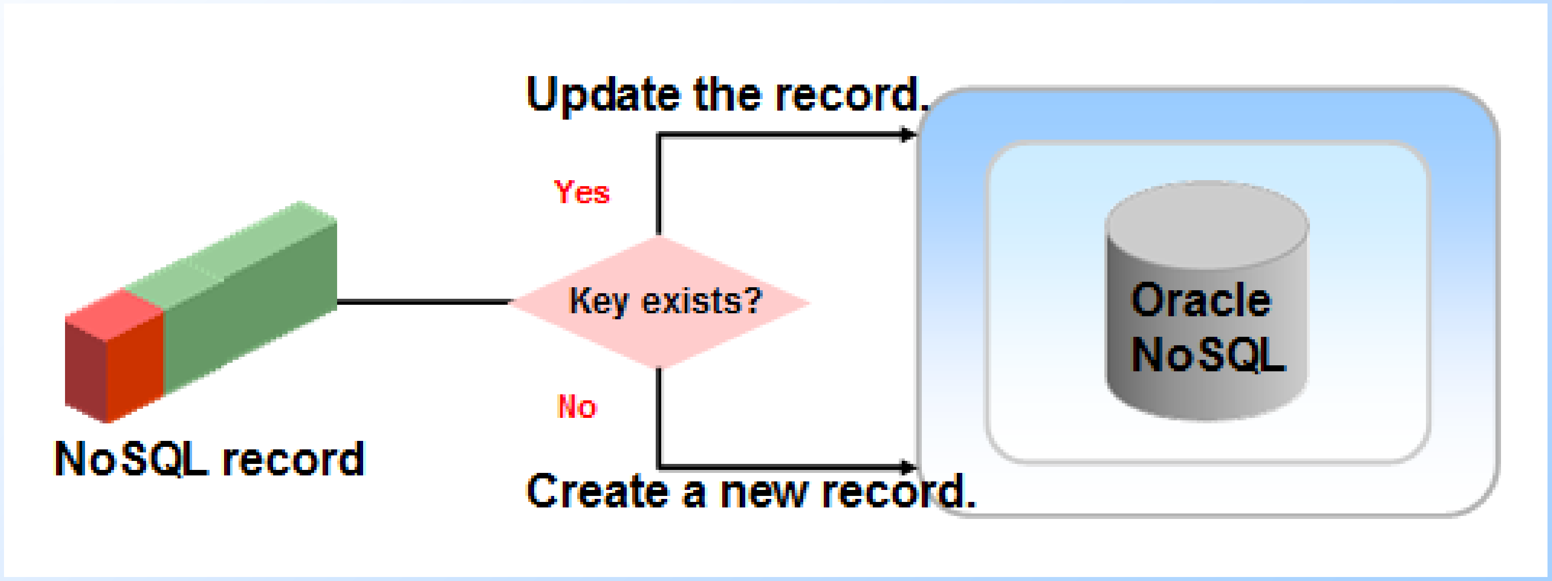
Reviewing Sample Code: Credit Card Approval Application

```
public class KeyDefinition
{
    static final String APRV_PROPERTY_NAME = "aprv";
    static final String INFO_PROPERTY_NAME = "info";

    static Key makeUserAprvKey(String ccNumber)
    {
          return Key.createKey(ccNumber,
APRV_PROPERTY_NAME);
    }
    static Key makeUserInfoKey(String ccNumber)
    {
          return Key.createKey(ccNumber,
INFO_PROPERTY_NAME);
    }
}
```

Reviewing Sample Code: Credit Card Approval Application

- The `KeyDefinition` class of the Credit Card Approval application contains the code to create the key component. This class contain two methods:
  - **`makeUserAprvKey:`** Takes the credit card number and returns a key with minor key value `aprv`
  - **`makeUserInfoKey:`** Takes the credit card number and returns a key with minor key value `info`

Procesul Write



Procesul Write

- Records are written to a KVStore based on their key. If a record with a specified key does not exist in the store, the record is created in the KVStore.
- If there is an existing record in the store with the specified key, the record is updated with the data that is being passed.
- However, there are methods available that enable you to write a record if it either exists or does not exist.

Procesul Write

- To write a record to the store:
  1. Construct a key.
  2. Construct a value.
  3. Use an API to write the record to the store.
- You have already learned how to construct a key and a value. Next, you learn how to write a constructed record to the store.

API Java pentru procesul Write

- Use the following APIs to write records to a KVStore:
  - `put()`
  - `putIfAbsent()`
  - `putIfPresent()`
  - `putIfVersion()`

API Java pentru procesul Write

- You can use any one of the four methods listed in the slide to write records to a KVStore.
  - **`put()`:** Writes a record to the KVStore by either creating a new record or overwriting an existing record as appropriate
  - **`putIfAbsent()`:** Writes a record to the KVStore only if there is no existing record with the specified key
  - **`putIfPresent()`:** Writes a record to the KVStore only if there is already an existing record with the specified key
  - **`putIfVersion()`:** Writes a record to the KVStore only if the existing record’s version matches the `Version` argument specified with this method

Handling Write and Delete Exceptions

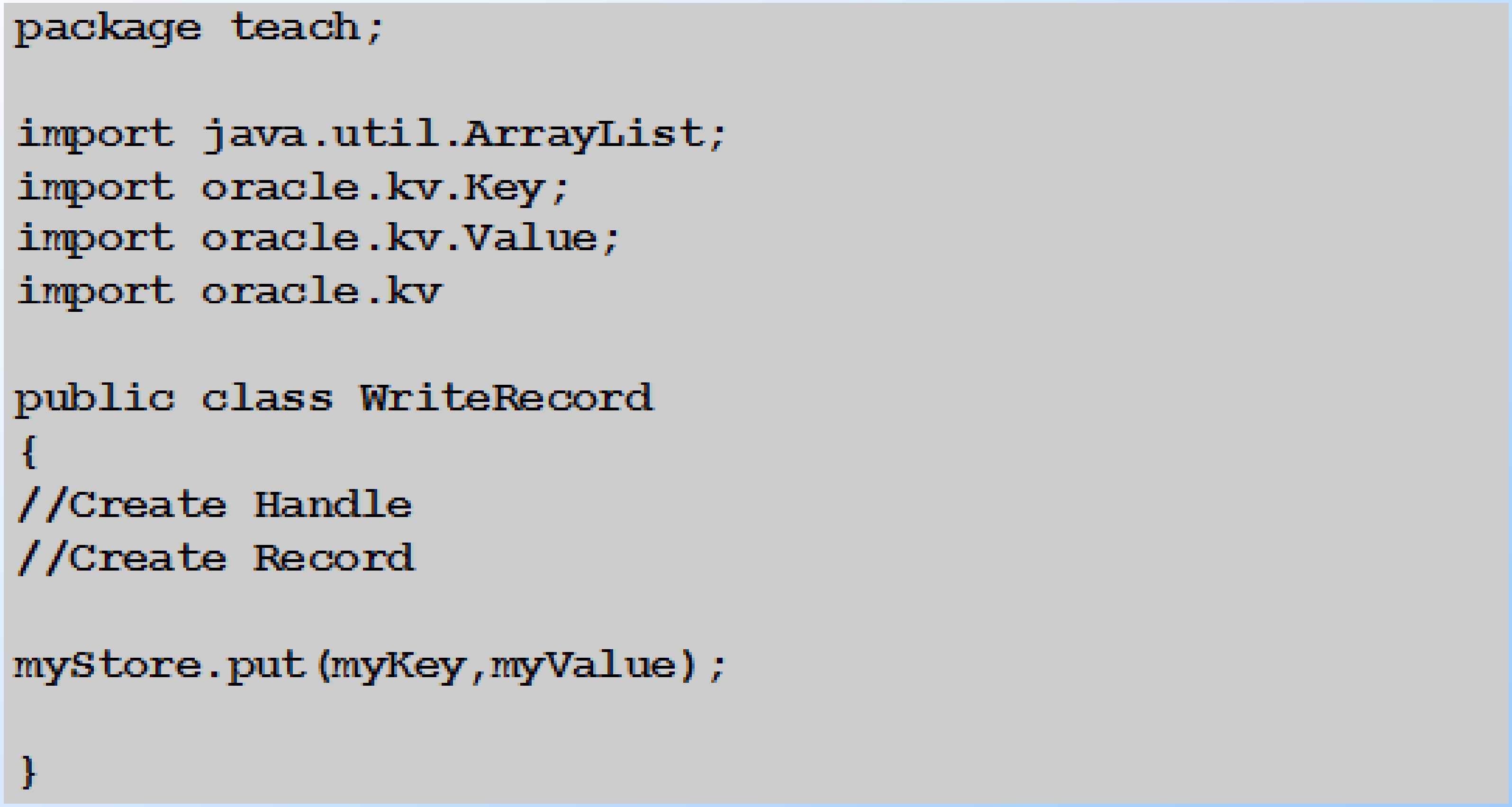
- Handle the following exceptions when writing to or deleting from a KVStore:
  - `DurabilityException`
  - `RequestTimeoutException`
  - `FaultException`

Procesul Write cu put()

```
Version put(Key key, Value value)
throws DurabilityException,
RequestTimeoutException,
FaultException
```

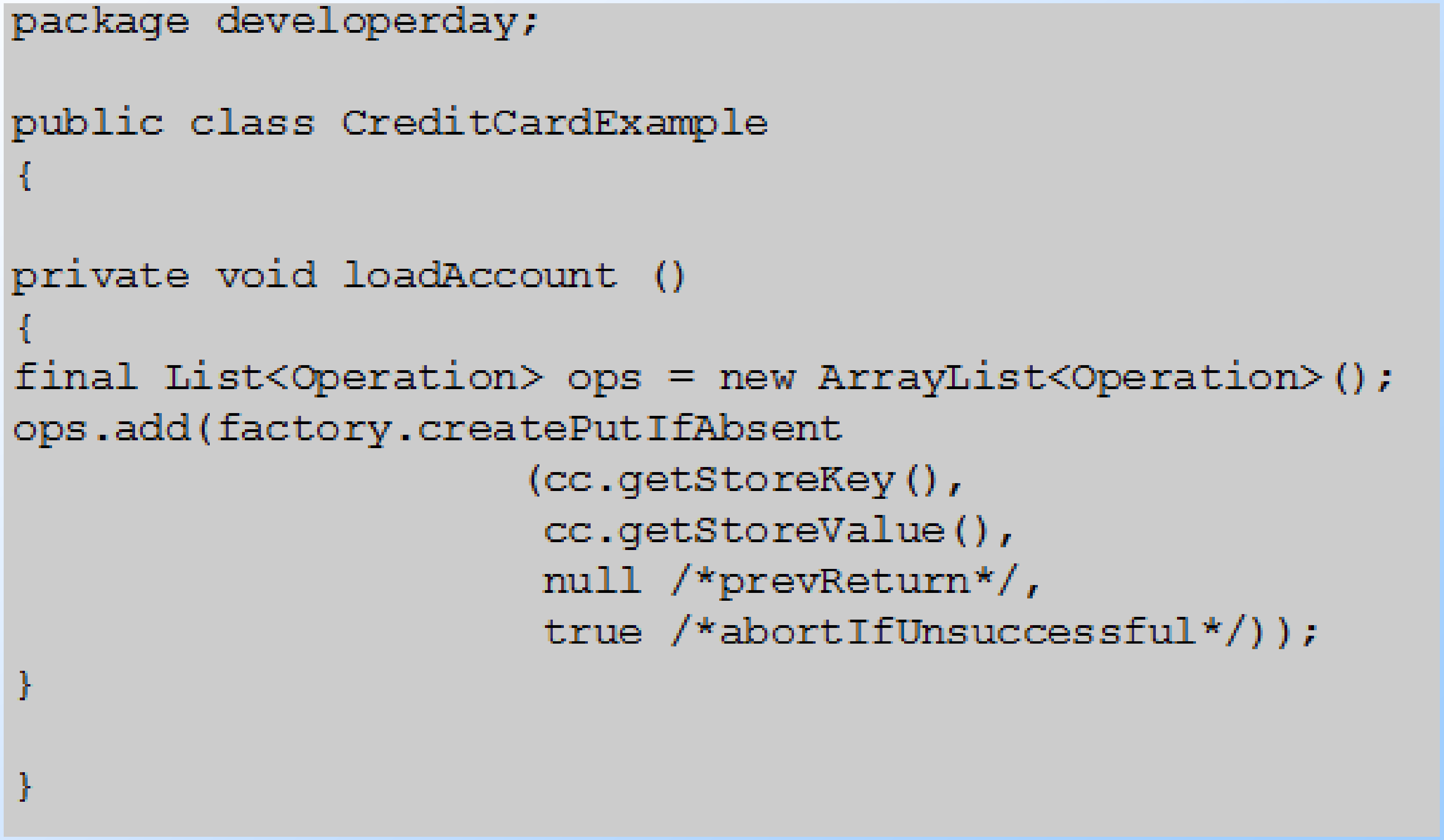
- The slide shows the simplest definition of the `put()` method. It takes two arguments: the key part and the value part of the record to be inserted. It returns the version number of the inserted record.
- In this method, the default durability and timeout values are used.

Exemplu: Procesul Write



```
package teach;

import java.util.ArrayList;
import oracle.kv.Key;
import oracle.kv.Value;
import oracle.kv

public class WriteRecord
{
//Create Handle
//Create Record

myStore.put(myKey,myValue);

}
```

Reviewing Sample Code: Credit Card Approval Application



```
package developerday;

public class CreditCardExample
{

private void loadAccount ()
{
final List<Operation> ops = new ArrayList<Operation>();
ops.add(factory.createPutIfAbsent
                    (cc.getStoreKey(),
                     cc.getStoreValue(),
                     null /*prevReturn*/,
                     true /*abortIfUnsuccessful*/));
}


}
```