Advanced Programming 2 - Accessing Databases

DR. ELIAHU KHALASTCHI

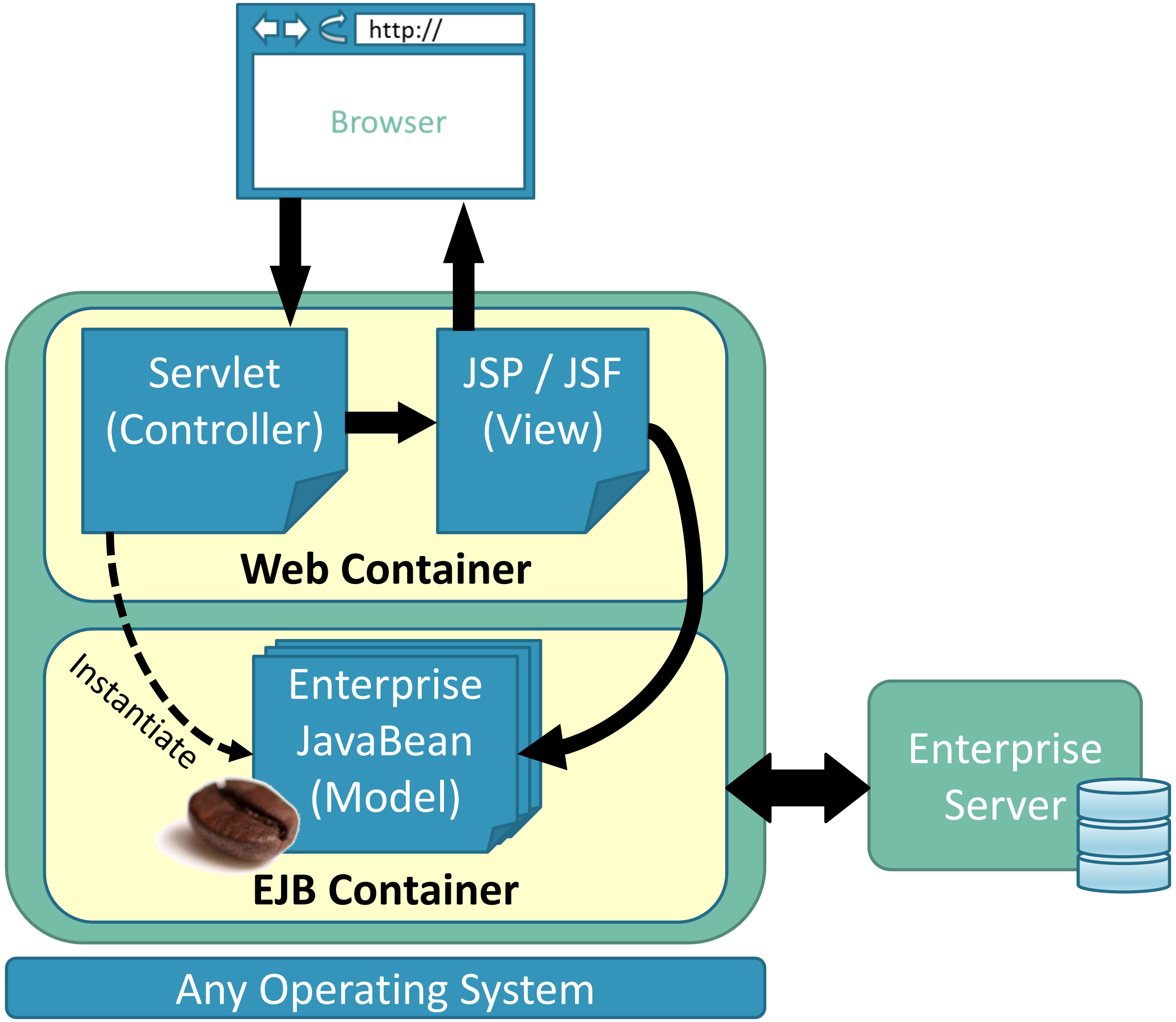
2016

In the last lessons…

Client Side:
(client tier)

Java2 EE
Application Server
(middle tier)

Enterprise
Information
System (EIS)
tier



Today’s Lesson

ACCESSING DATABASES

ADO.Net

Agenda

- What is a data base
- Querying a data base
- Introduction to ADO.Net
- Data Providers
- Connecting to data bases
- Querying and reading a data base (fully connected)
- Querying and reading a data base (disconnected)
- A small tip about security

What is a relational data base? (briefly)

- It is a collection of data, typically organized in relational data models (tables)
- For example:

Employees Table

Attributes:

A record:

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
…	…	…

A relation

Department Table

Number	Name	manager
101	control	12345
413	R&D	96534
…	…	…

Querying a Data Base

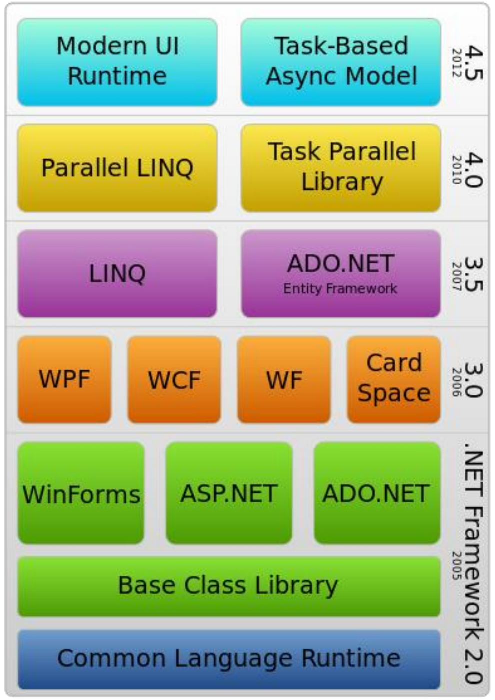
- Data bases of different types can be queried by different querying languages
- For example, an **SQL** data base can be accessed with the **S**tandard **Q**uerying **L**anguage
- Crating Records:
  - Insert into Employees (ID, Name, DepartmentId, Salary, Location)
    values (98765, “Charles Montgomery Burns”, 0, 1000000, Springfield)
- Read Records:
  - Select ID, name from Employees //Get ID and name columns for all records
  - Select * from Employees where ID = 12345 //Get all columns for a single record
- Update Records:
  - Update Employees Set salary = 18000 where ID = 12345

How do we use it in our program?

ADO.NET

Introduction

- ADO – Active Data Objects
- ADO.NET is an **object-oriented** set of libraries that allows you to interact with data sources
- Commonly, the data source is a **database**, but it could also be
  - a text file, an Excel spreadsheet, or an XML file
- ADO.NET provides a **relatively common way** to interact with data sources
  - but comes in different sets of libraries for each way you can talk to a data source
- These libraries are called **Data Providers**



The .NET Framework Stack

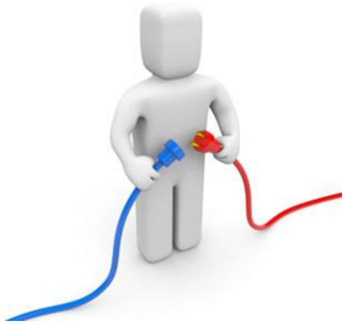
Data Providers

Provider Name	API prefix	Data Source Description
ODBC Data Provider	Odbc	Data Sources with an ODBC interface. Normally older data bases.
OleDb Data Provider	OleDb	Data Sources that expose an OleDb interface, i.e. Access or Excel.
Oracle Data Provider	Oracle	For Oracle Databases.
SQL Data Provider	Sql	For interacting with Microsoft SQL Server.
Borland Data Provider	Bdp	Generic access to many databases such as Interbase, SQL Server, IBM DB2, and Oracle.

What is the API prefix?

- Consider a “connection” object that allows you to connect to your data base
- An **OleDb**Connection object allows you to connect to OleDb interfaced data bases (access, excel)
- An **Odbc**Connection object allows you to connect to Odbc interfaced data bases
- An **Sql**Connection object allows you to connect to SQL interfaced data bases
- etc.

Connecting to a data base



- First, we need to create a connection object
- The connection tells the rest of the ADO.NET code which database it is talking to
- It manages all of the low level logic associated with the specific database protocols

- A connection is a valuable resource
  - You need to understand connections in order to make the right decisions when coding your data access routines
  - Think about an enterprise application where hundreds of users are accessing the same database
  - Think about a website that is being visited by hundreds of thousands of users…

- To establish a connection we simply need to instantiate a connection object

Connecting to a data base

- The CTOR: **SqlConnection**(string connectionString)

- The connection string sets the following parameters:
  - Data source - Identifies the server. Could be **local machine**, machine **domain name,** or **IP Address**
  - Initial Catalog – the database name
  - Integrated Security - set to SSPI to make connection with user's Windows login
  - User ID - Name of user configured in SQL Server
  - Password - Password matching SQL Server User ID

- Example:

```
SqlConnection conn = new SqlConnection(
      "Data Source=(local); Initial Catalog=SpringfieldDB; User ID=YourUserID; Password=YourPassword“
);
```

Reading a Data Base

```
using System;
using System.Data;
using System.Data.SqlClient;
 class SqlConnectionDemo    {
        static void Main(string[] args)  {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(
                "Data Source=(local);Initial Catalog=SpringfieldDB;
                 Integrated Security=SSPI");

            SqlDataReader rdr = null;

            try   {
                // 2. Open the connection
                conn.Open();

                // 3. Pass the connection to a command object
                SqlCommand cmd = new SqlCommand("select * from Employees", conn);

              // 4. Use the connection
              // get query results
                rdr = cmd.ExecuteReader();

                // print the EmployeeID of each record
                while (rdr.Read())   {
                    Console.WriteLine(rdr[0]);
                }
            }
            finally {
                // close the reader
                if (rdr != null)  {
                    rdr.Close();
                }

                // 5. Close the connection
                if (conn != null)  {
                    conn.Close();
                }
            }
        }
    }
```

```
using System.Data;
using System.Data.SqlClient;
```

Reading a Data Base

```
using System;
using System.Data;
using System.Data.SqlClient;
 class SqlConnectionDemo    {
        static void Main(string[] args)  {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(
                "Data Source=(local);Initial Catalog=SpringfieldDB;
                 Integrated Security=SSPI");

            SqlDataReader rdr = null;

            try   {
                // 2. Open the connection
                conn.Open();

                // 3. Pass the connection to a command object
                SqlCommand cmd = new SqlCommand("select * from Employees", conn);

              // 4. Use the connection
              // get query results
                rdr = cmd.ExecuteReader();

                // print the EmployeeID of each record
                while (rdr.Read())   {
                    Console.WriteLine(rdr[0]);
                }
            }
            finally {
                // close the reader
                if (rdr != null)  {
                    rdr.Close();
                }

                // 5. Close the connection
                if (conn != null)  {
                    conn.Close();
                }
            }
        }
    }
```

```
// 1. Instantiate the connection
SqlConnection conn = new SqlConnection(
                    "Data Source=(local);
                     Initial Catalog=SpringfieldDB;
                     Integrated Security=SSPI");
```

Reading a Data Base

```
using System;
using System.Data;
using System.Data.SqlClient;
 class SqlConnectionDemo    {
        static void Main(string[] args)  {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(
                "Data Source=(local);Initial Catalog=SpringfieldDB;
                 Integrated Security=SSPI");

            SqlDataReader rdr = null;

            try   {
                // 2. Open the connection
                conn.Open();

                // 3. Pass the connection to a command object
                SqlCommand cmd = new SqlCommand("select * from Employees", conn);

              // 4. Use the connection
              // get query results
                rdr = cmd.ExecuteReader();

                // print the EmployeeID of each record
                while (rdr.Read())   {
                    Console.WriteLine(rdr[0]);
                }
            }
            finally {
                // close the reader
                if (rdr != null)  {
                    rdr.Close();
                }

                // 5. Close the connection
                if (conn != null)  {
                    conn.Close();
                }
            }
        }
    }
```

```
SqlDataReader rdr = null;

try   {
 // 2. Open the connection
 conn.Open();

 // 3. Pass the connection to a command object
 SqlCommand cmd = new SqlCommand("select * from Employees",conn);
```

Employees Table

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
…	…	…

Reading a Data Base

```
using System;
using System.Data;
using System.Data.SqlClient;
 class SqlConnectionDemo    {
        static void Main(string[] args)  {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(
                "Data Source=(local);Initial Catalog=SpringfieldDB;
                 Integrated Security=SSPI");

            SqlDataReader rdr = null;

            try   {
                // 2. Open the connection
                conn.Open();

                // 3. Pass the connection to a command object
                SqlCommand cmd = new SqlCommand("select * from Employees", conn);

              // 4. Use the connection
              // get query results
                rdr = cmd.ExecuteReader();

                // print the EmployeeID of each record
                while (rdr.Read())   {
                    Console.WriteLine(rdr[0]);
                }
            }
            finally {
                // close the reader
                if (rdr != null)  {
                    rdr.Close();
                }

                // 5. Close the connection
                if (conn != null)  {
                    conn.Close();
                }
            }
        }
    }
```

```
  // 4. Use the connection
  // get query results
  rdr = cmd.ExecuteReader();

  // print the EmployeeID of each record
  while (rdr.Read())   {
     Console.WriteLine(rdr[0]);
  }
}
```

Employees Table

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
…	…	…

Reading a Data Base

```
using System;
using System.Data;
using System.Data.SqlClient;
 class SqlConnectionDemo    {
        static void Main(string[] args)  {
            // 1. Instantiate the connection
            SqlConnection conn = new SqlConnection(
                "Data Source=(local);Initial Catalog=SpringfieldDB;
                 Integrated Security=SSPI");

            SqlDataReader rdr = null;

            try   {
                // 2. Open the connection
                conn.Open();

                // 3. Pass the connection to a command object
                SqlCommand cmd = new SqlCommand("select * from Employees", conn);

              // 4. Use the connection
              // get query results
                rdr = cmd.ExecuteReader();

                // print the EmployeeID of each record
                while (rdr.Read())   {
                    Console.WriteLine(rdr[0]);
                }
            }
            finally {
                // close the reader
                if (rdr != null)  {
                    rdr.Close();
                }

                // 5. Close the connection
                if (conn != null)  {
                    conn.Close();
                }
            }
        }
    }
```

```
finally {
    // close the reader
    if (rdr != null)  {
        rdr.Close();
    }

    // 5. Close the connection
    if (conn != null)  {
        conn.Close();
    }
}
```

The SqlCommand Object

- Querying data:
  - SqlCommand cmd = new SqlCommand("**select** ID **from** Employees", conn);
  - SqlDataReader rdr = cmd.ExecuteReader();   // Call Execute reader to get query results

- Inserting data:
  - string insertString = @”Insert into Employees (ID, Name, DepartmentID, Salary, Location) values (98765, “Charles Montgomery Burns”, 0, 1000000, Springfield)”;
  - SqlCommand cmd = new SqlCommand(insertString, conn);
  - cmd.ExecuteNonQuery(); // Call ExecuteNonQuery to send command, expecting ‘void’ in return

The SqlCommand Object

- Updating data: (this time using a different constructor)
  - string updateString = @”Update Employees Set salary = 18000 where ID = 12345”;
  - SqlCommand cmd = new SqlCommand(updateString);
  - cmd.Connection = conn;
  - cmd.ExecuteNonQuery();

- Deleting data: (this time using the default constructor)
  - string deleteString = @”delete from Employees where ID = 12345”;
  - SqlCommand cmd = new SqlCommand();
  - cmd.CommandText = deleteString;
  - cmd.Connection = conn;
  - cmd.ExecuteNonQuery();

The SqlDataReader

- A SqlDataReader is a type that is good for reading data in the most efficient manner possible

- You can read from SqlDataReader objects in a forward-only sequential manner
  - You will not be able to go back and read it again, you will need a new query for that…
  - The forward only design of the SqlDataReader is what enables it to be fast – less overhead

- The object is instantiated with an ExecuteReader call
  - SqlDataReader rdr = cmd.ExecuteReader();

- You need a loop to read the data - row by row

The SqlDataReader

- The returned row contains values in its columns
- These values can be accessed with the ’[]’ operator (like a map or array)
  - We can insert an index, but it is preferable to use a string index since it is more readable

```
while (rdr.Read())  // read a row, returns a Boolean
{
    // get the results of each column
    int id = (int)rdr[0];   // not very readable
    string name = (string)rdr["Name"];
    int depId = (string)rdr["DepartmentID"];
    // now we can do anything we want with
    // these C# objects :)
}
```

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
…	…	…

Disconnected Data

Introduction

- Up until now we used a fully connected mode
  - We used the *SqlCommand* object to query the data base
  - We used the *SqlDataReader* to read it quickly
  - But the session with the server was open the entire time!
  - Very resource consuming, particularly if used by many clients…

- What about something in between?
  - Introducing the *DataSet* & *DataAdapter* objects!

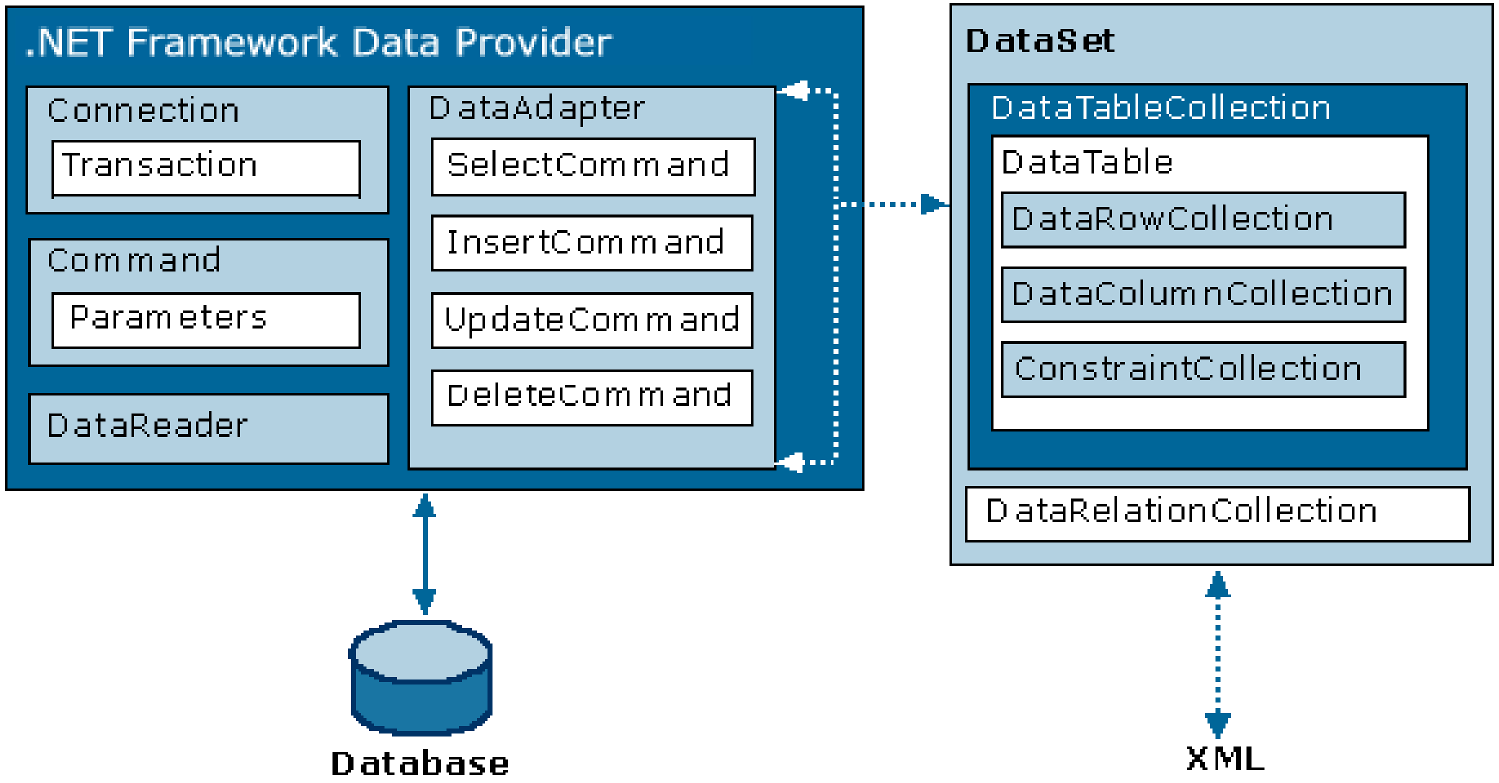
The DataSet & SqlDataAdapter Objects

- A **DataSet** is an **in-memory** data store that can hold numerous tables
- DataSets only hold data and do not interact with a data source

- It is the **SqlDataAdapter** that manages connections
- It opens a connection only when required and closes it as soon as it has performed its task
  - Automatically!

- An initialized DataSet can be read numerous times without invoking the data base

The ADO.Net Architecture



Let’s get to work!

```
DataSet ds = new DataSet();
DataTable departments = new DataTable("Departments");
DataTable employees = new DataTable("Employees");

// filling the Employee table from the data base
SqlCommand cmd = new SqlCommand("select * from Employees", conn);
SqlDataAdapter tableAdapter = new SqlDataAdapter(cmd); // opens conn
// use Fill method to fill the data table
tableAdapter.Fill(employees); // connection is now closed

// filling the Department table from the data base
cmd = new SqlCommand("select * from Departments", conn);
tableAdapter = new SqlDataAdapter(cmd); // connection is opened
tableAdapter.Fill(departments);// connection is now closed again

// adding the tables to the data set
ds.Tables.Add(departments);
ds.Tables.Add(employees);
```

Departments Table

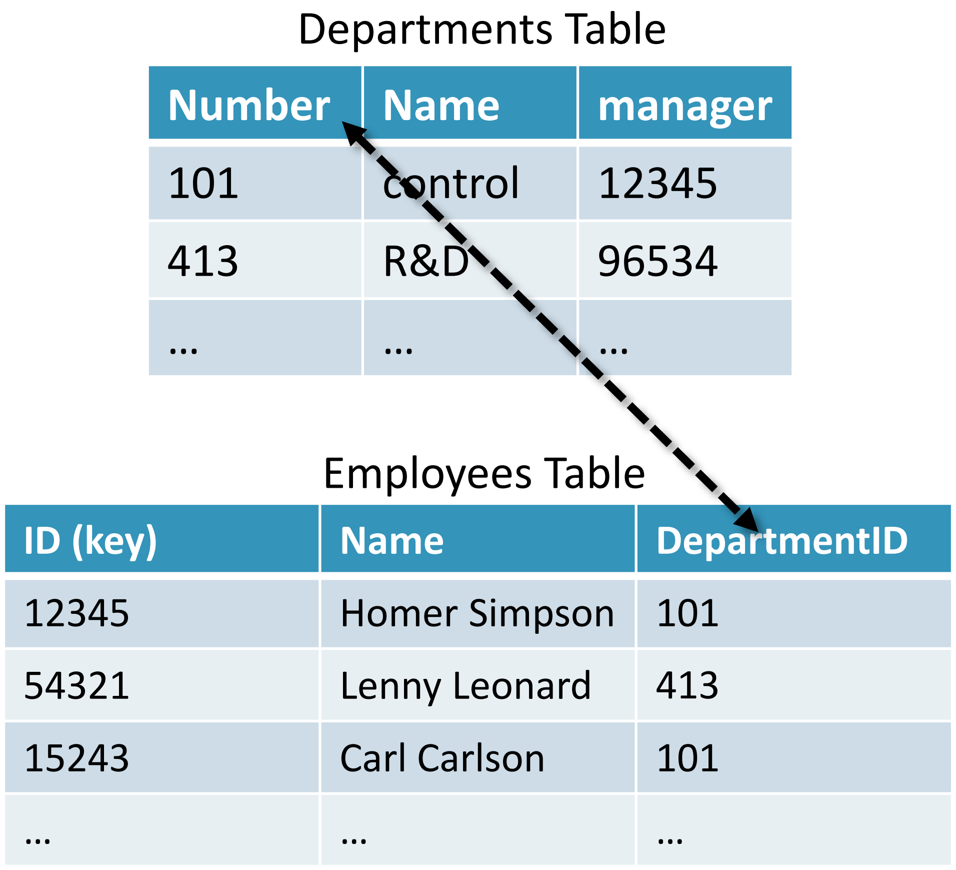
Number	Name	manager
101	control	12345
413	R&D	96534
…	…	…

Employees Table

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
…	…	…

Let’s get to work!

```
// adding a relation
DataRelation dr=new DataRelation("EmployeeDepartment",
                    ds.Tables[0].Columns["Number"],
                    ds.Tables[1].Columns["DepartmentID"]
                    );
ds.Relations.Add(dr);

// adding a new row
// only in our memory, not the data base yet
DataRow nr = ds.Tables["Employees"].NewRow();
nr["ID"] = "34825";
nr["Name"] = "Bart Simpson";
nr["DepartmentID"] = "101";

ds.Tables["Employees"].Rows.Add(nr);
```



Departments Table

Number	Name	manager
101	control	12345
413	R&D	96534
…	…	…

Employees Table

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
…	…	…

Let’s get to work!

```
// acessing the data set
    foreach(DataTable aTable in ds.Tables)
        foreach (DataRow aRow in aTable.Rows)
            foreach (DataColumn aCol in aTable.Columns)
                Console.WriteLine(aRow[aCol]);

// updating the data base when needed
tableAdapter.Update(ds,"Employees");
```

Departments Table

Number	Name	manager
101	control	12345
413	R&D	96534
…	…	…

Employees Table

ID (key)	Name	DepartmentID
12345	Homer Simpson	101
54321	Lenny Leonard	413
15243	Carl Carlson	101
34825	**Bart Simpson**	**101**

ORM

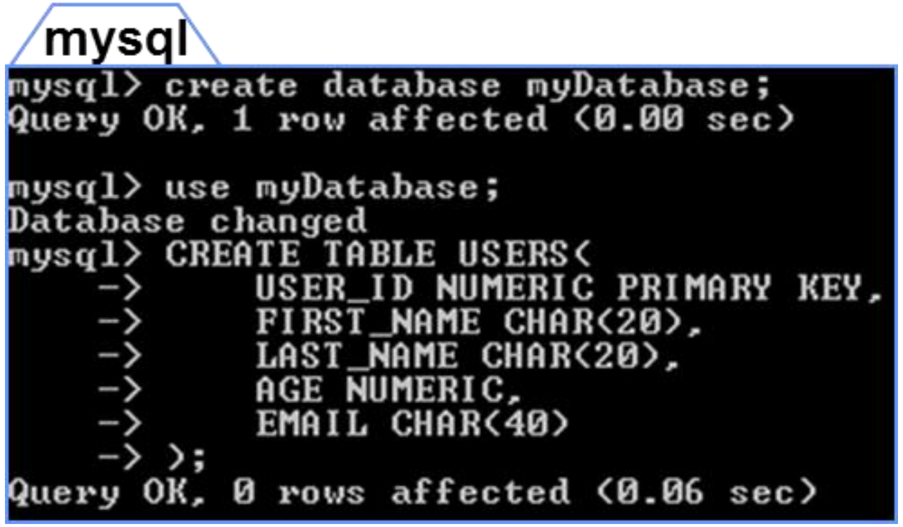
OBJECT RELATIONAL MAPPING

Object Relational Mapping

- Automatically convert relational data ⇔ objects
- You can implement your own code, or use existing tools
- Hibernate: HIBERNATE

```java
public class User {
 private long userId = 0 ;
 private String firstName = "";
 private String lastName = "";
 private int age = 0;
 private String email = "";
}
```

⇔



```
mysql> create database myDatabase;
Query OK, 1 row affected (0.00 sec)

mysql> use myDatabase;
Database changed
mysql> CREATE TABLE USERS(
    ->         USER_ID NUMERIC PRIMARY KEY,
    ->         FIRST_NAME CHAR(20),
    ->         LAST_NAME CHAR(20),
    ->         AGE NUMERIC,
    ->         EMAIL CHAR(40)
    -> );
Query OK, 0 rows affected (0.06 sec)
```

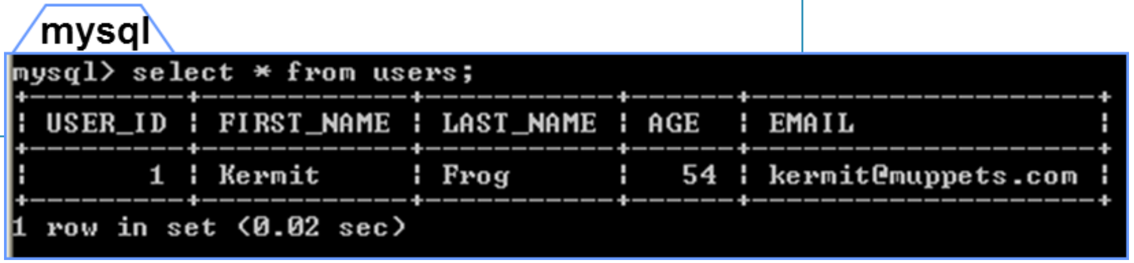
Hibernate Mapping via XML

```xml
<?xml version="1.0"?>
  <!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >

  <hibernate-mapping>
      <class name="User" table="USERS" >
          <id name="userId" type="java.lang.Long" column="user_id" >
                <generator class="increment" />
          </id>
          <property name="firstName" type="java.lang.String" column="first_name" length="20" />
          <property name="lastName" type="java.lang.String" column="last_name" length="20" />
          <property name="age" type="java.lang.Integer" column="age" length="-1" />
          <property name="email" type="java.lang.String" column="email" length="40" />
      </class>
  </hibernate-mapping>
```

Hibernate Use Example

```java
import org.hibernate.Session;

public class UserManager {
 private Session session = null;
 public UserManager(Session session) {
    this.session = session;
 }
 public void saveUser(User user){
  session.save(user);
 }
 public void updateUser(User user){
  session.update(user);
 }
 public void deleteUser(User user) {
  session.delete(user);
 }
}
```

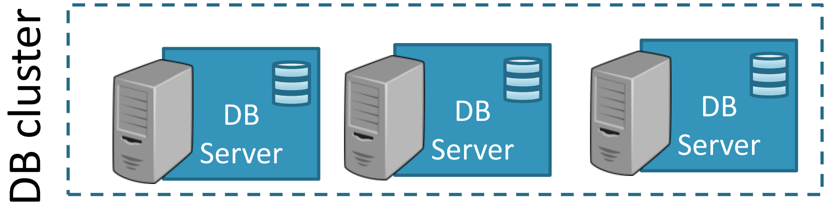
```java
public static void main(String[] args) {
 User user = new User();
 user.setFirstName("Kermit");
 user.setLastName("Frog");
 user.setAge(54);
 user.setEmail("kermit@muppets.com");

 SessionFactory sessionFactory = new
 Configuration().configure().buildSessionFactory();
 Session session = sessionFactory.openSession();
 UserManager manager = new UserManager(session);

 manager.saveUser(user);
 session.flush();
}
```



```
mysql> select * from users;
+---------+------------+-----------+------+--------------------+
| USER_ID | FIRST_NAME | LAST_NAME | AGE  | EMAIL              |
+---------+------------+-----------+------+--------------------+
|       1 | Kermit     | Frog      |   54 | kermit@muppets.com |
+---------+------------+-----------+------+--------------------+
1 row in set (0.02 sec)
```

ACID

ACID

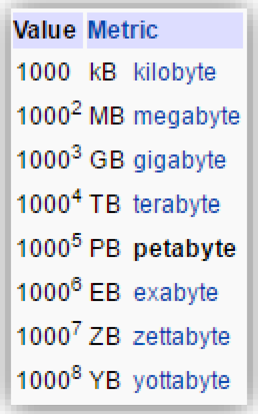


- We want to access databases concurrently
- And do it right…
- ACID properties:
  - **Atomicity** – requires that each transaction be "all or nothing“
  - **Consistency** – any transaction should bring the database from one valid state to another
  - **Isolation** – concurrent execution of transactions should result in a system state that would be obtained if transactions were executed serially (one after the other)
  - **Durability** – once a transaction has been committed, it will remain so, even in the event of power loss, crashes, or errors
- Hard to achieve…
  - Especially when trying to scale-out

BIG
DATA
USE
ONE
SINCE
SOFTWARE
REQUIRING
UBIQUITOUS
RADIO-FREQUENCY
COMPLEXITY
SAN
SOLID
WIRELESS
TOLERABLE
BUSINESS
MOVING
ANALYTICS
PARALLEL
SIZE
NEEDED
QUALITIES
BURIED
ORGANIZATIONS
PETABYTES
PROCESSING
LOGS
INTERNET
TECHNOLOGIES
USED
DISTRIBUTED
CAPTURE
MAY
MANAGEMENT
MANAGE
GROW
SOCIAL
LARGE
DEFINING
STORE
EVERY
CASE
LARGER
USING
TYPES
WORKING
GARTNER
MASSIVELY
PERFORMANCE
RELATED
ALSO
BIOLOGICAL
HUNDREDS
CREATED
COMPUTING
TOOLS
SET
GENOMICS
ZETTABYTES
DISK
RELATIONAL
SHARED
TIME
WITHIN
COMBAT
SIGNIFICANT
PROCESS
INCLUDE
SYSTEMS
DEFINITION
BIOGEOCHEMICAL
NETWORKS
RECONSIDER
SEARCH
OPPORTUNITIES
INFORMATION
CONNECTOMICS
RECORDS
DESKTOP
CURRENTLY
FC
TENS
CAPACITY
FORMS
PRACTITIONERS
NOW
PRESENTATIONS
WORLD'S
COST
CONTINUES
TERABYTES
MPP
RESEARCH
STORAGE
CITATION
COMPLEX
DATABASES
INDEXING
DIFFICULTY
TARGET
ABILITY
SETS
EXAMPLES
ELAPSED
CURRENT
THOUGHT
APPLIED
SENSOR
ARCHIVES
AMOUNT
DESCRIBING

Bid Data

- Data sets that are so large or complex
  - that traditional data processing applications **are inadequate**
- petabytes of data…



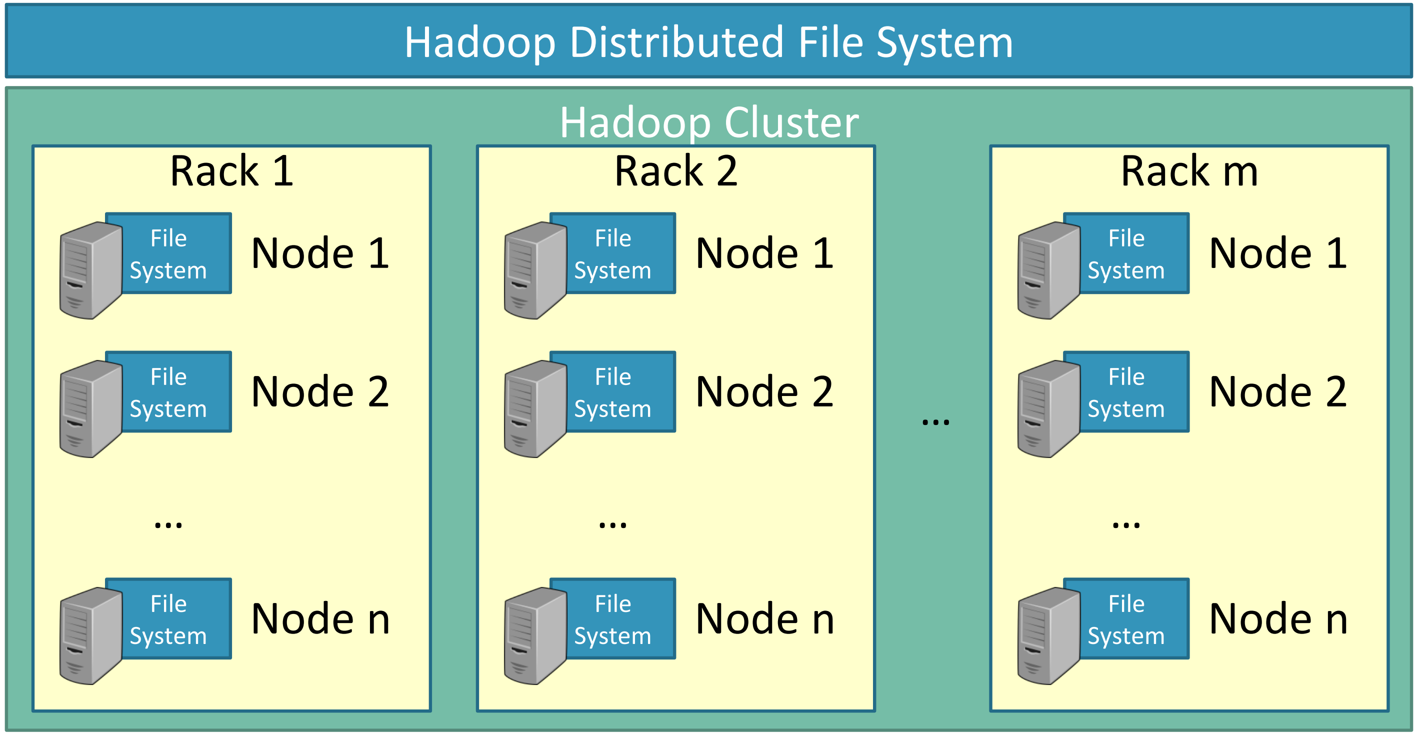
Value	Metric	
1000	kB	kilobyte
1000^2	MB	megabyte
1000^3	GB	gigabyte
1000^4	TB	terabyte
1000^5	PB	**petabyte**
1000^6	EB	exabyte
1000^7	ZB	zettabyte
1000^8	YB	yottabyte

- Characteristics – the 5 V's:
  - **Volume** – The quantity of generated and stored data
  - **Variety** – The different forms of data we collect
  - **Velocity** – The speed at which the data is generated
  - **Variability / Veracity** – Inconsistency of the data / trustworthiness of the data
  - **Value** – can we extract useful information from it / why we fight!

YAHOO!
facebook
twitter
Linked in
ebaY
IBM
amazon
AOL
Adobe
Bai du 图标
last·fm
hulu
hadoop

Apache Hadoop

- Open source
- Distributed **storage** and **processing** of very large data sets on computer clusters
- Hadoop Distributed File System (HDFS):
  - Scalable (6k nodes and 120PB)



Hadoop Distributed File System

Hadoop Cluster

Rack 1

File System Node 1

File System Node 2

…

File System Node n

Rack 2

File System Node 1

File System Node 2

…

File System Node n

…

Rack m

File System Node 1

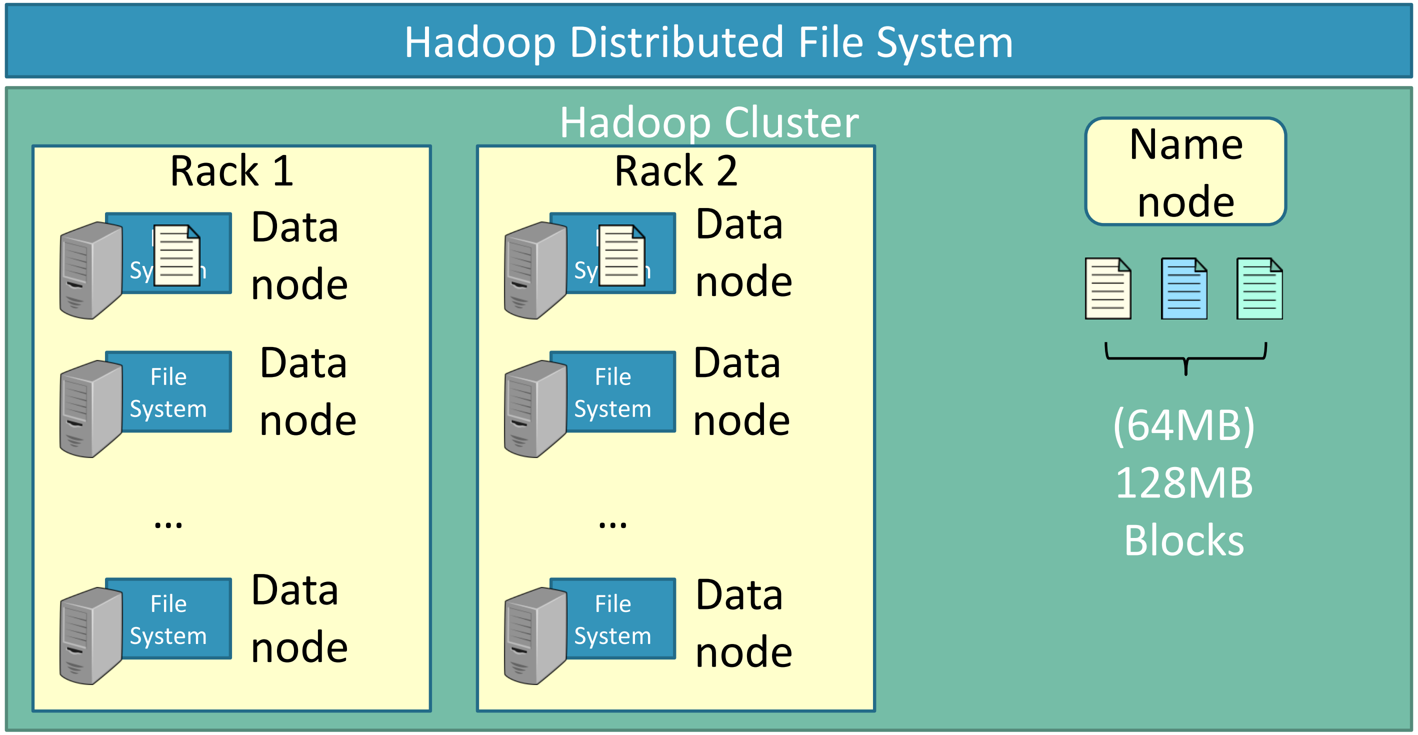
File System Node 2

…

File System Node n

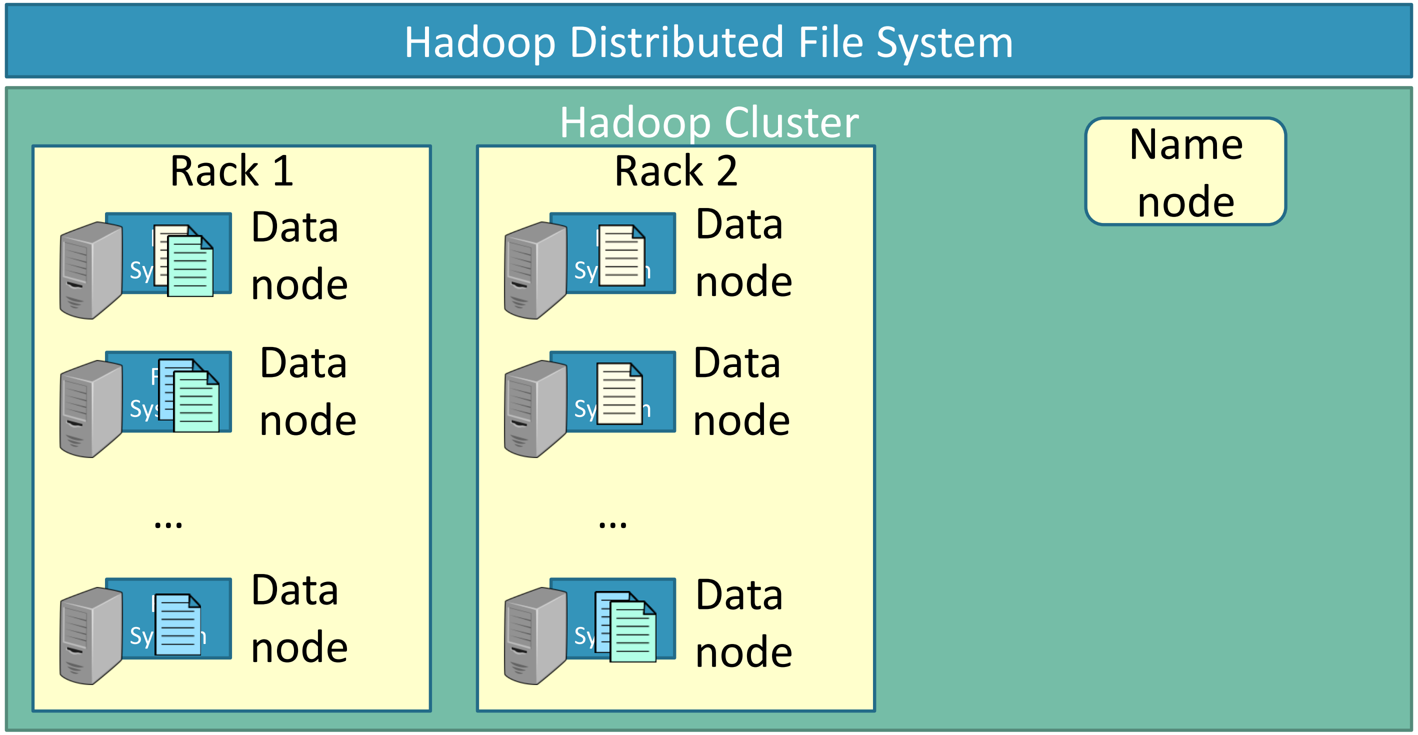
Apache Hadoop

- Open source
- Distributed **storage** and **processing** of very large data sets on computer clusters
- Hadoop Distributed File System (HDFS):
  - Scalable (6k nodes and 120PB)



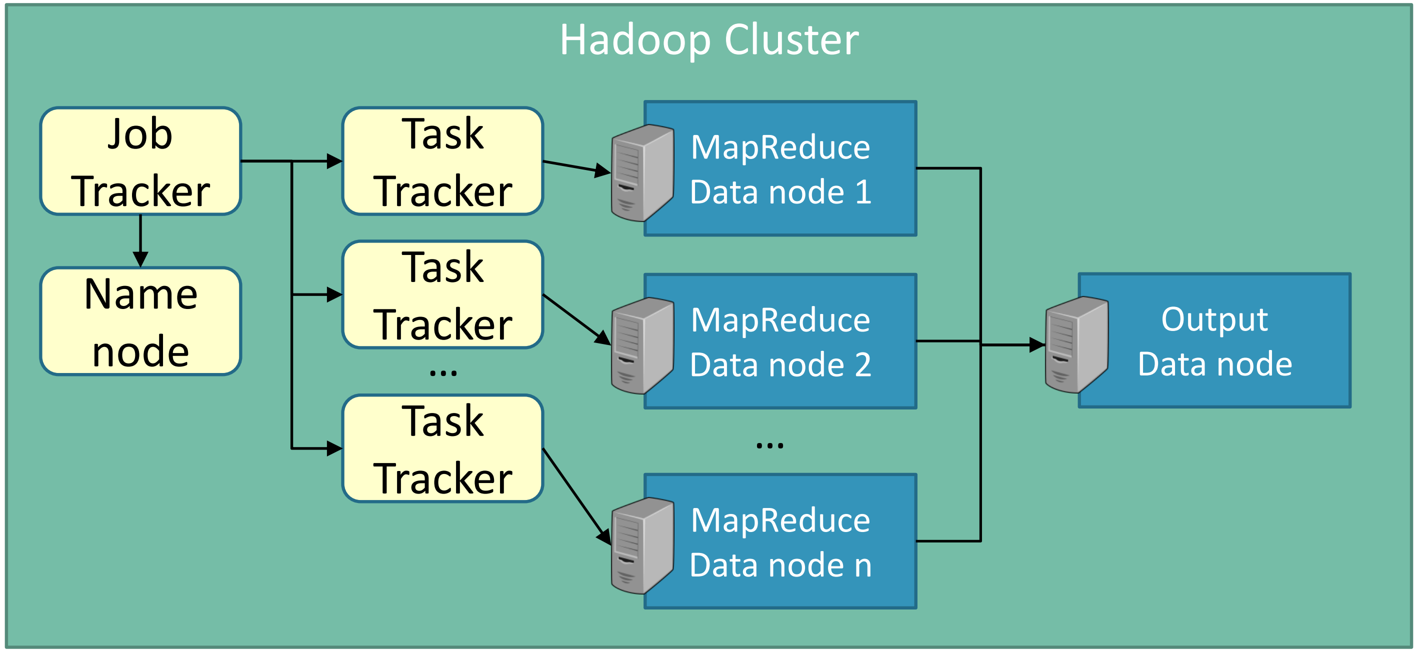
Apache Hadoop

- Open source
- Distributed **storage** and **processing** of very large data sets on computer clusters
- Hadoop Distributed File System (HDFS):
  - Scalable (6k nodes and 120PB)
  - Reliable
  - Manageable
  - Portable (Java)



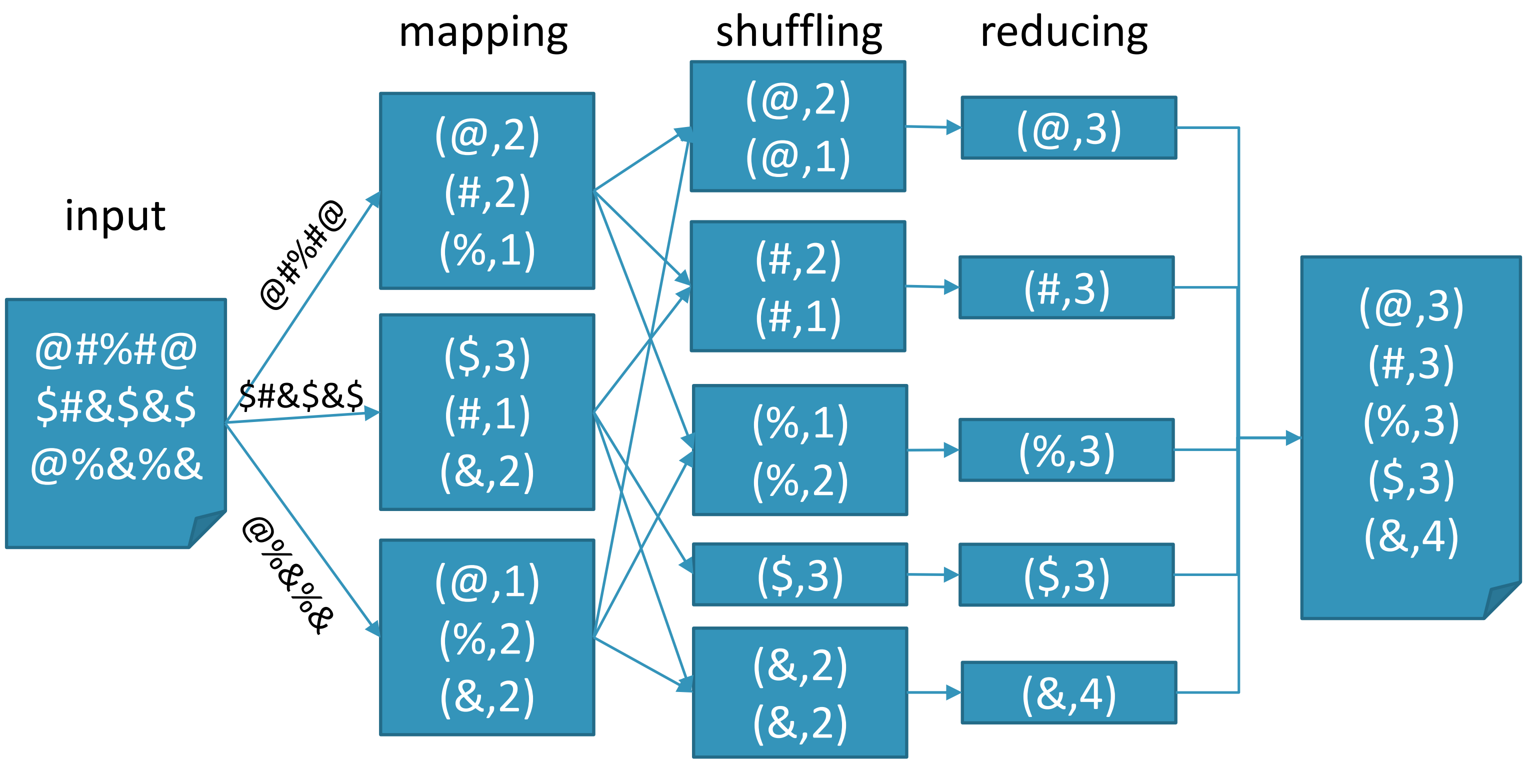
Apache Hadoop

- Open source
- Distributed **storage** and **processing** of very large data sets on computer clusters
- Hadoop Distributed File System (HDFS):
  - Scalable (6k nodes and 120PB)
  - Reliable
  - Manageable
  - Portable (Java)
- MapReduce
  - Parallel, distributed algorithm



Apache Hadoop

- Open source
- Distributed **storage** and **processing** of very large data sets on computer clusters
- Hadoop Distributed File System (HDFS):
  - Scalable (6k nodes and 120PB)
  - Reliable
  - Manageable
  - Portable (Java)
- MapReduce
  - Parallel, distributed algorithm
  - Java API
  - Example: word count



input

@#%#@
$#&$&$
@%&%&

@#%#@

$#&$&$

@%&%&

mapping

(@,2)
(#,2)
(%,1)

($,3)
(#,1)
(&,2)

(@,1)
(%,2)
(&,2)

shuffling

(@,2)
(@,1)

(#,2)
(#,1)

(%,1)
(%,2)

($,3)

(&,2)
(&,2)

reducing

(@,3)

(#,3)

(%,3)

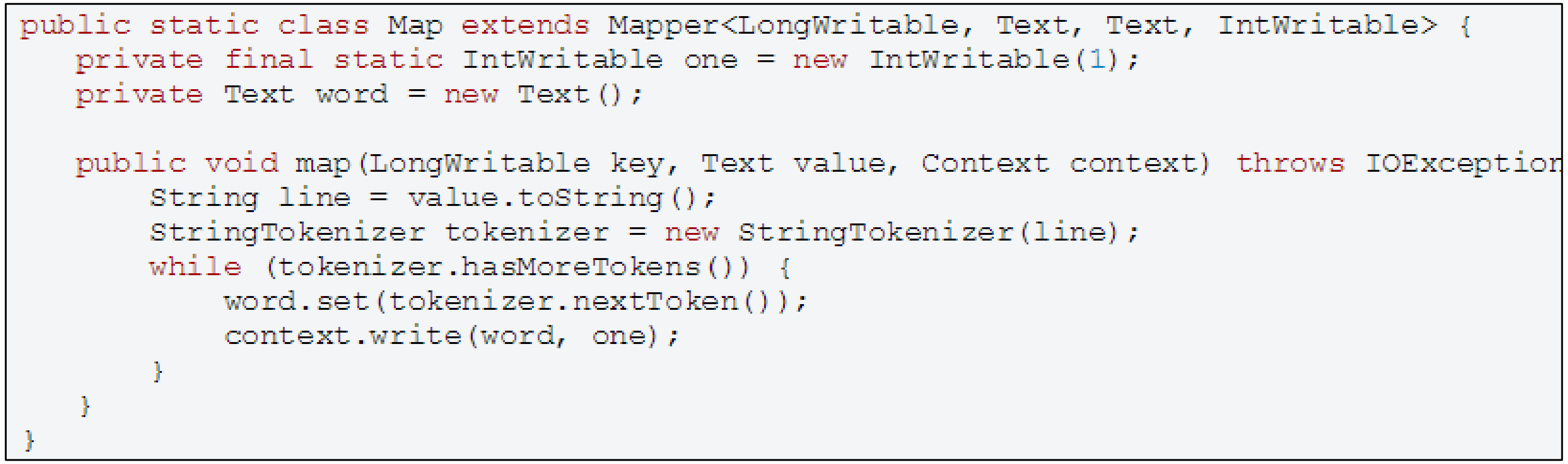
($,3)

(&,4)

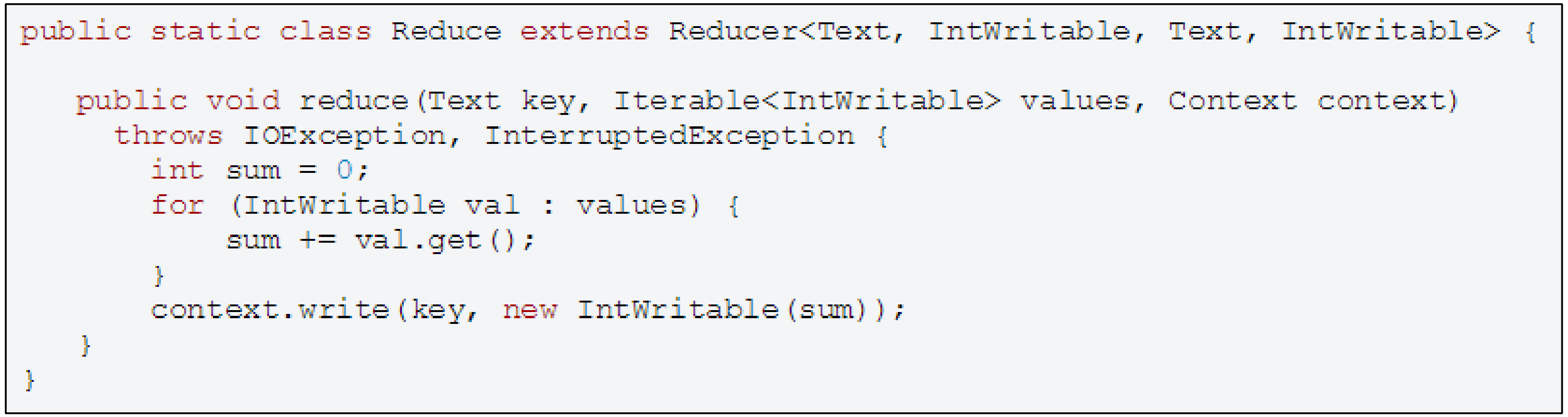
(@,3)
(#,3)
(%,3)
($,3)
(&,4)

Simple MapReduce in Java

- Word count Example:



```
public static class Map extends Mapper<LongWritable, Text, Text, IntWritable> {
   private final static IntWritable one = new IntWritable(1);
   private Text word = new Text();

   public void map(LongWritable key, Text value, Context context) throws IOException
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
           word.set(tokenizer.nextToken());
           context.write(word, one);
       }
   }
}
```



```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterable<IntWritable> values, Context context)
     throws IOException, InterruptedException {
       int sum = 0;
       for (IntWritable val : values) {
           sum += val.get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```