AMIS

Friends of Oracle and Java

Instrumenting, Analyzing, & Tuning the Performance of Oracle ADF Applications

Frank Houweling
UKOUG 2014

Frank Houweling

- Frank Houweling
- Senior Java / ADF specialist
- Focus on performance diagnosis and performance management
- Lead developer of the ADF Performance Monitor
- AMIS (Netherlands)
- Oracle Partner

Agenda

- What is instrumentation
- Why instrumentation is important
- Cost of tracking
- Analyzing and tuning
- Oracle ODL Analyzer
- Build your own performance monitor
  - What you can instrument
    - Not ADF specific
    - ADF specific
  - Five examples how you can instrumentation key spots in ADF applications
    - HTTP Request
    - Errors / Exceptions
    - ADF Business Components
    - JVM memory consumption
    - ApplicationModule pooling (activation and passivation)

Instrumentation

- Instrumentation; we can not live without it

Instrumentation of Code

- Instrumentation refers to an ability to monitor or measure the level of a product's performance, to diagnose errors and to write trace information
- Instrumenting gives visibility and insight of what is happening inside the application
  - What methods and queries are executed, when and how often
- Programmers implement instrumentation in the form of code instructions that monitor specific components in a system

Why instrumentation is important (1)

- Many applications are like a smoke screen
- It is very unclear what is happening in the background
- Where should we look for bottlenecks ?

Why instrumentation is important (2)

- End-users do not accept slow applications anymore
- Is your application running as it should?
- Response Times within SLA agreements?
- Are there errors? What type/severity of errors?

Cost of Tracking

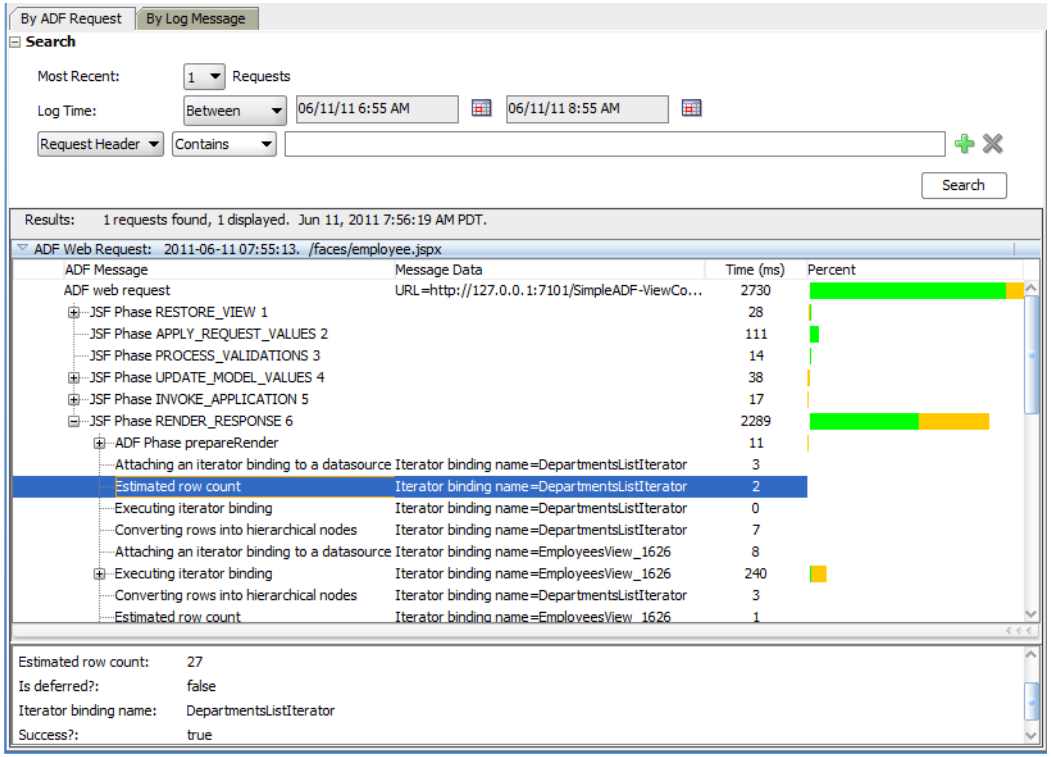
- Tracking everything is impossible – it is like a fire hose; there is far too much data and there will be a significant performance overhead
- We should find the balance; to track important/relevant events but also not tracking too much
- Tracking should be able to be turned on and off at runtime

Analyzing and Tuning

- What is the SLA?
- If users are complaining and if they are right, you have to know where time is spent
- Instrumentation should give clear performance information (metrics) when methods/operations begin and when they end, and calculate how long it took to execute
  - What methods and queries are executed, when and how often
- When we have insight into performance problems specific problems may be addressed

Instrumentation Provided by Oracle: ODL Analyzer

- In JDeveloper support for performance analysis of ADF apps is provided through the ODL Analyzer (By ADF Request).
- **Problem**: can't be used in test en product environments because of the amount of logging generated and the significant performance overhead

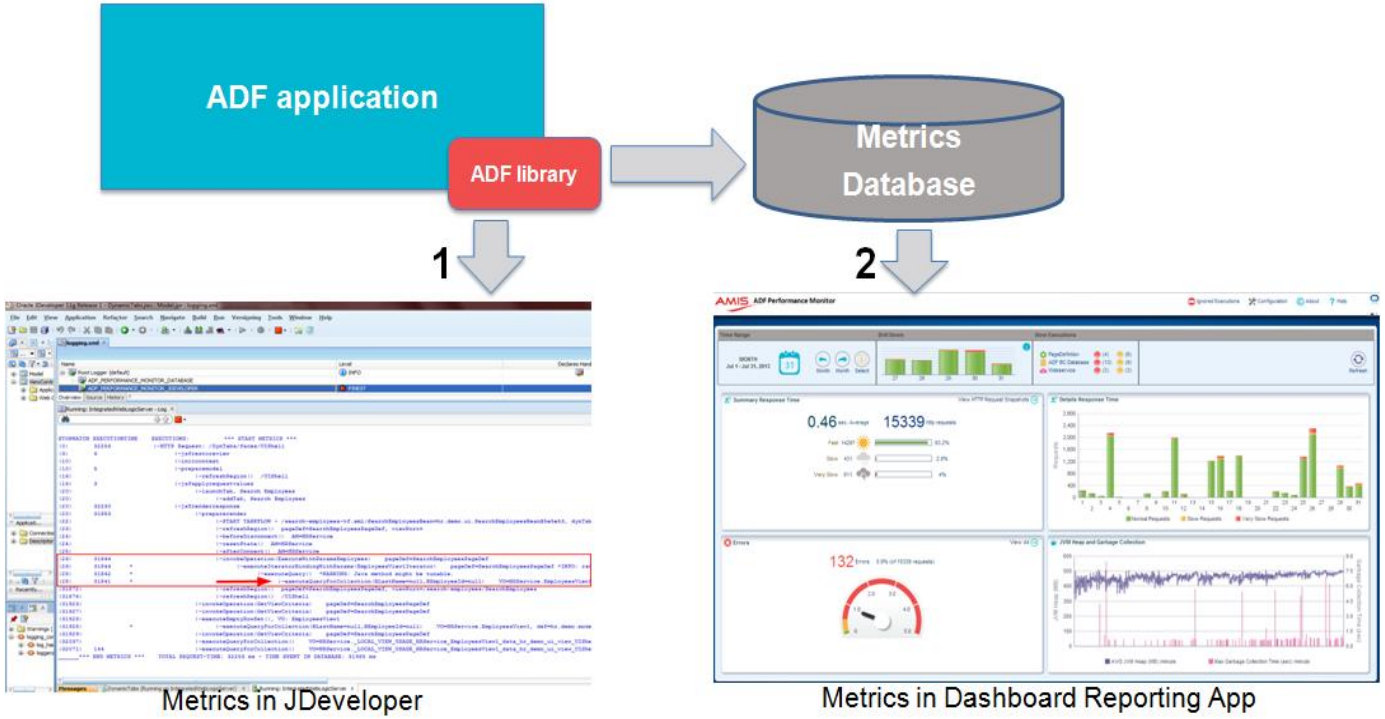


Built Your Own Monitor; Instrument your ADF code

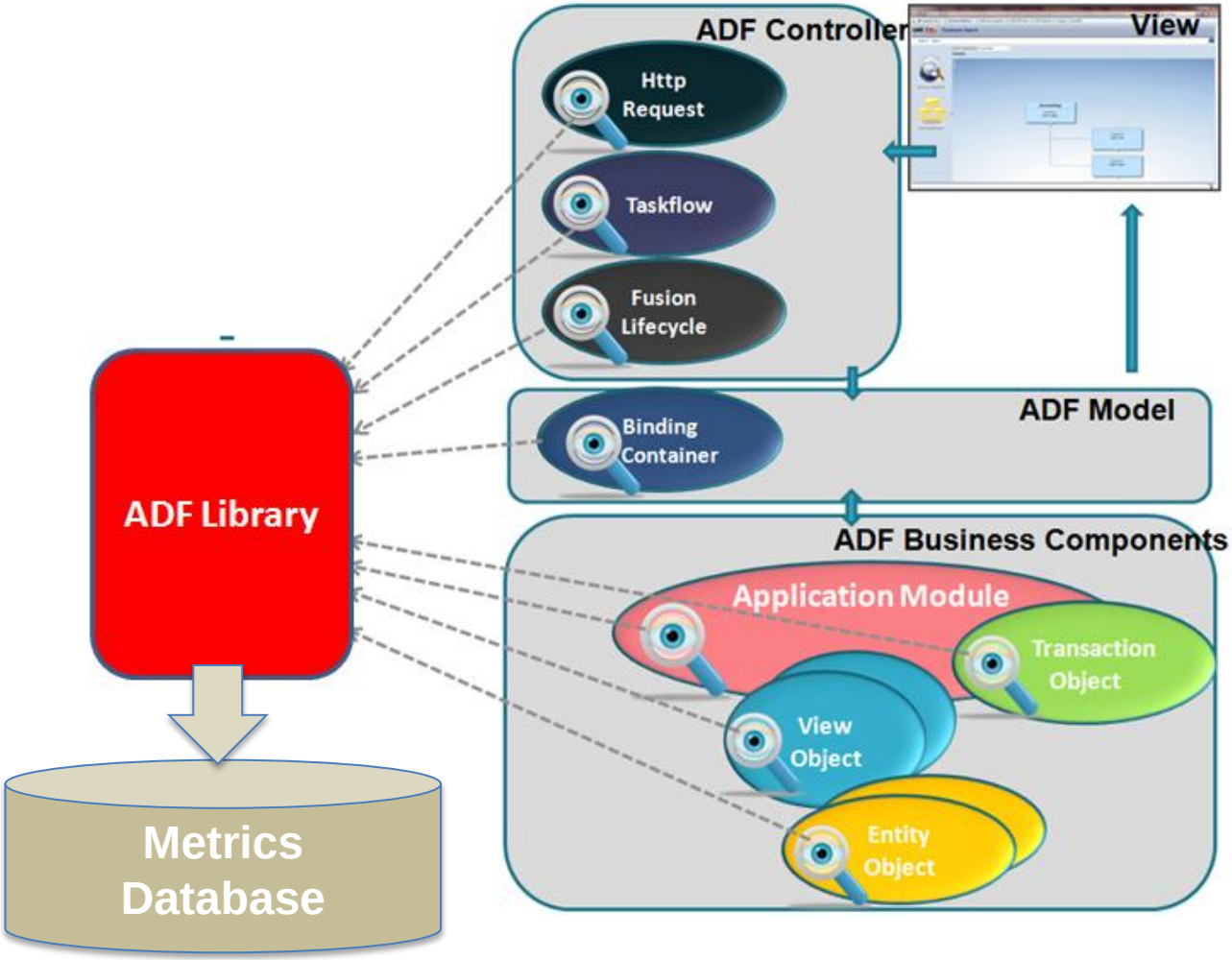
- You can instrument your own ADF code and create your own performance monitor tracking tool
- Create several ‘stopwatch’ metrics classes that can be plugged in into the ADF framework
  - ADF BC framework extension classes
  - Fusion lifecycle phaseListener
  - ServletFilter
  - HTTP eventListener
  - Error eventListener
- Create a “metrics” library that contains your metrics code

Save Metrics in Database

- Add this library to your ADF app
- The metrics library saves the metrics to the database
  - That uses its own JDBC Datasource
- You can create a Dashboard Reporting App based on the metrics

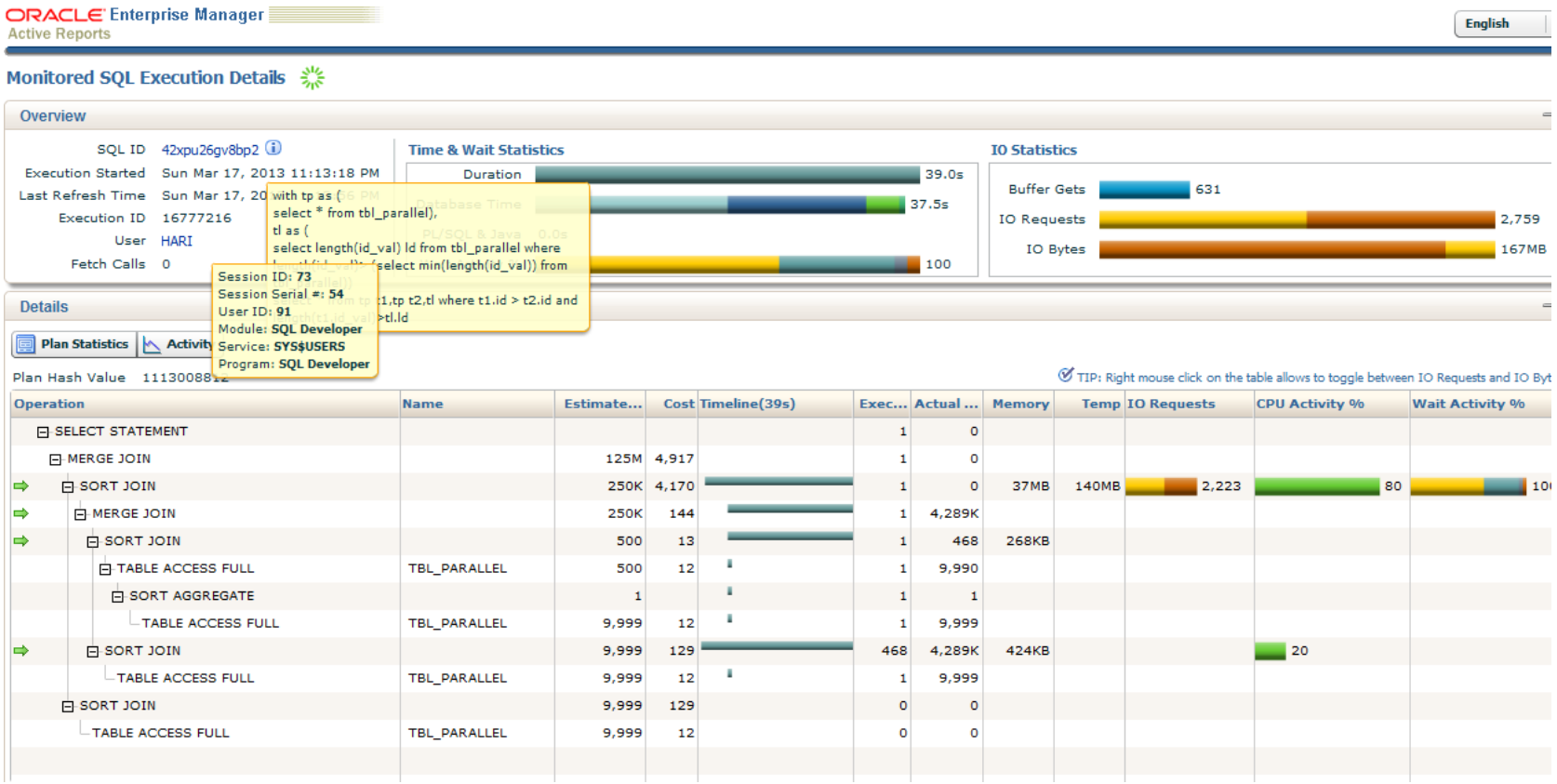


Instrument Multiple Layers



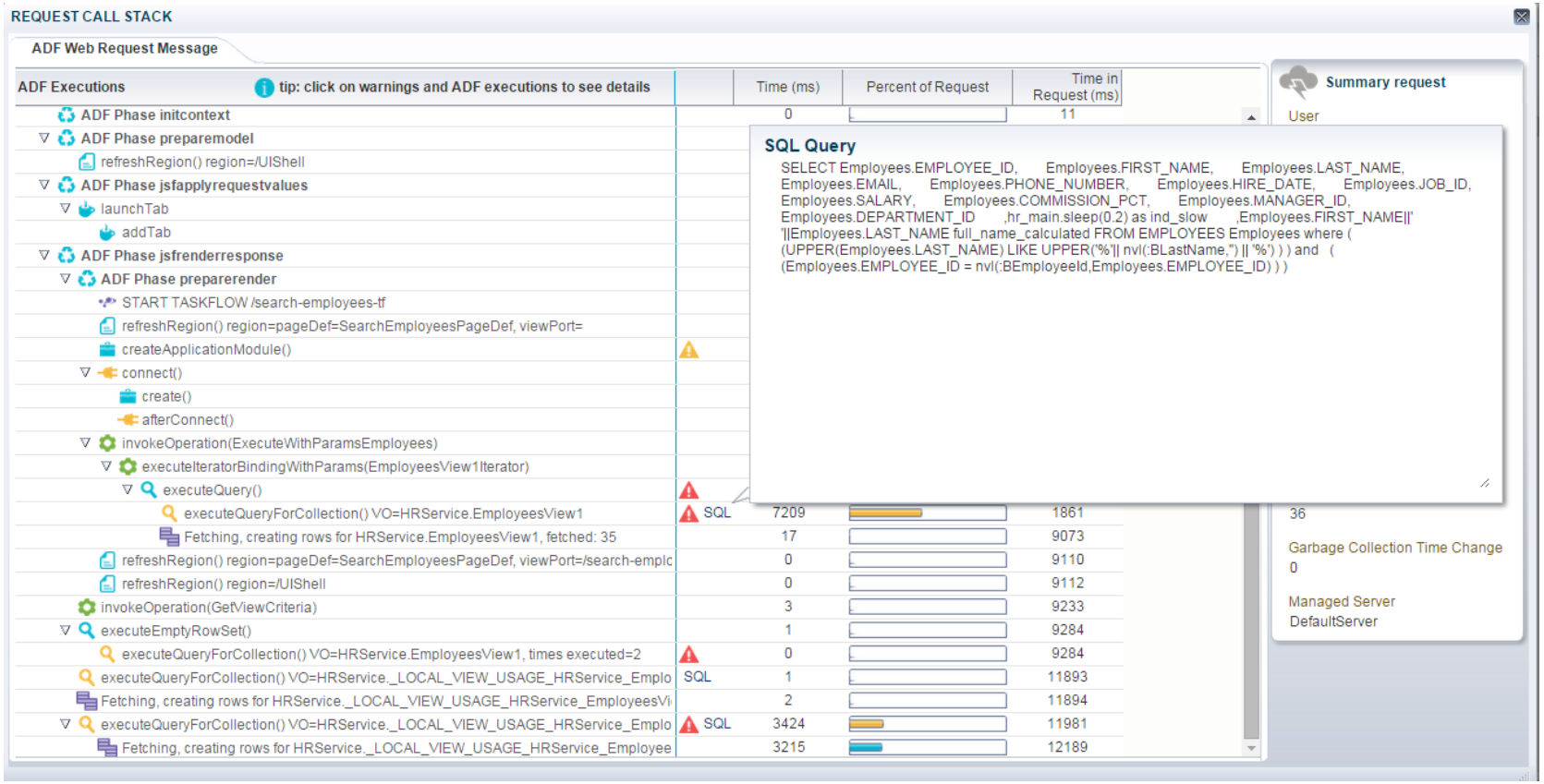
Instrumentation in Enterprise Manager

- SQL Execution Details shows metrics that give visibility and insight
  - After analysis SQL experts can take action to solve slow SQL executions



ADF Call Stack

- Like the database EM we can also create call stacks in ADF
  - Gives visibility into which ADF method caused other methods to execute
  - Organized by the sequence of their execution
  - Elapsed times and a view of what happened when.



What you could Instrument (1)
(Not ADF specific)

- HTTP Request
  - Response Time
  - User ID,
  - User Organization
  - Button/Link clicked on (ADF component ID)
  - View ID
  - Session ID
  - Date and Time
  - Browser
  - Client IP
  - WLS Managed Server Name
  - Server URI
  - ECID
- Errors / Exceptions
  - Error Code, Error Message, Error Class
  - Stack Trace
- JVM
  - Heap Space
  - Garbage Collection Time
- ……..

What you could Instrument (2)
(ADF specific)

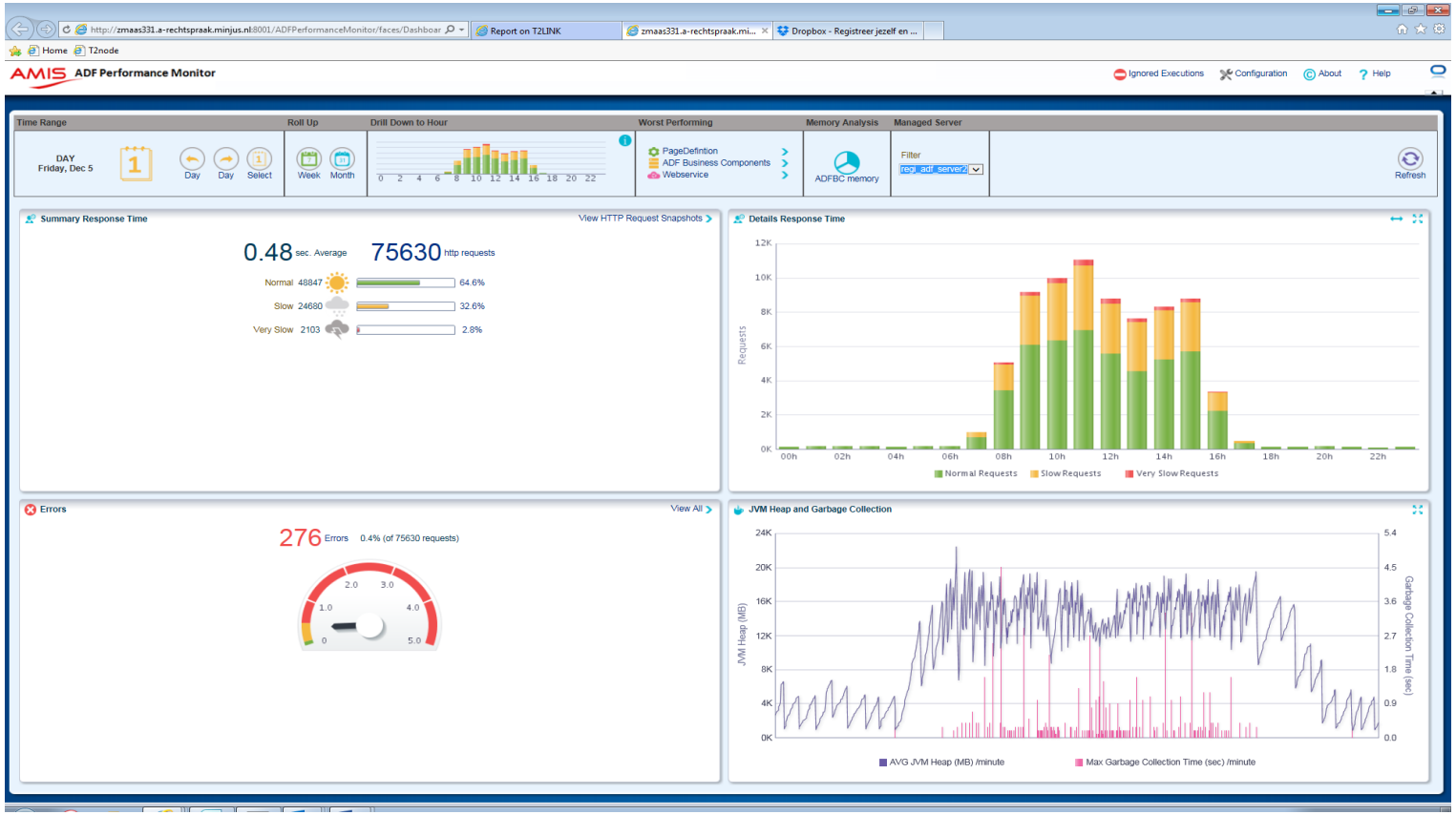
- ViewObject SQL queries
  - BindVariable values
  - Rows fetched (loaded) from database into ADF app
  - Applied ViewCriteria
  - SQL query statement
  - ViewAccessors
- EntityObject
  - DML methods
  - Locks
- ApplicationModule
  - AM Pooling; passivations and activations
  - Database transactions
  - Creation Time
  - Connection time to database
- Fusion Lifecycle phases
  - Start & end of phases
- Taskflow paramaters
  - Initializers and start parameters
  - finalyzers
- BindingContainer (PageDef)
  - Iterator executions
  - Operations and Custom Methods
- ……

Case 1
HTTP Request Instrumentation

- Use a ServletFilter to instrument HTTP Request response time
  - Instrument *doFilter()*
  - Alternatively *HttpServletRequestListener*
- User ID
  - *ADFContext adfCtx = ADFContext.getCurrent();*
    SecurityContext secCntx = adfCtx.getSecurityContext();
    String userId = secCntx.getUserName();
- Managed Server
  - *System.getProperties().getProperty("weblogic.Name");*
- Session ID
  - *httpServletRequest.getSession().getId()*
- Server URI
  - *httpServletRequest.getRequestURI();*

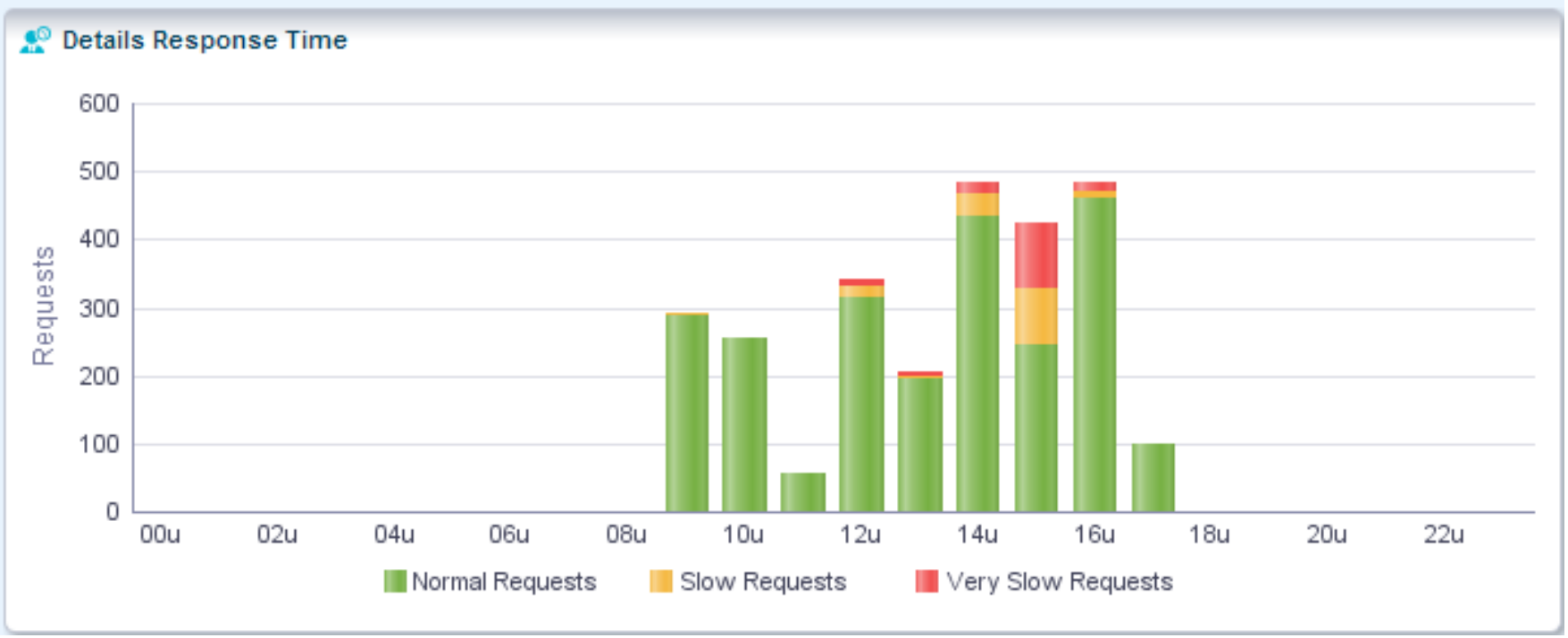
Monitor Service Level Agreement (SLA)

- Real-time performance information
- Are response times within or outside SLA boundaries?
- Is immediate action required?



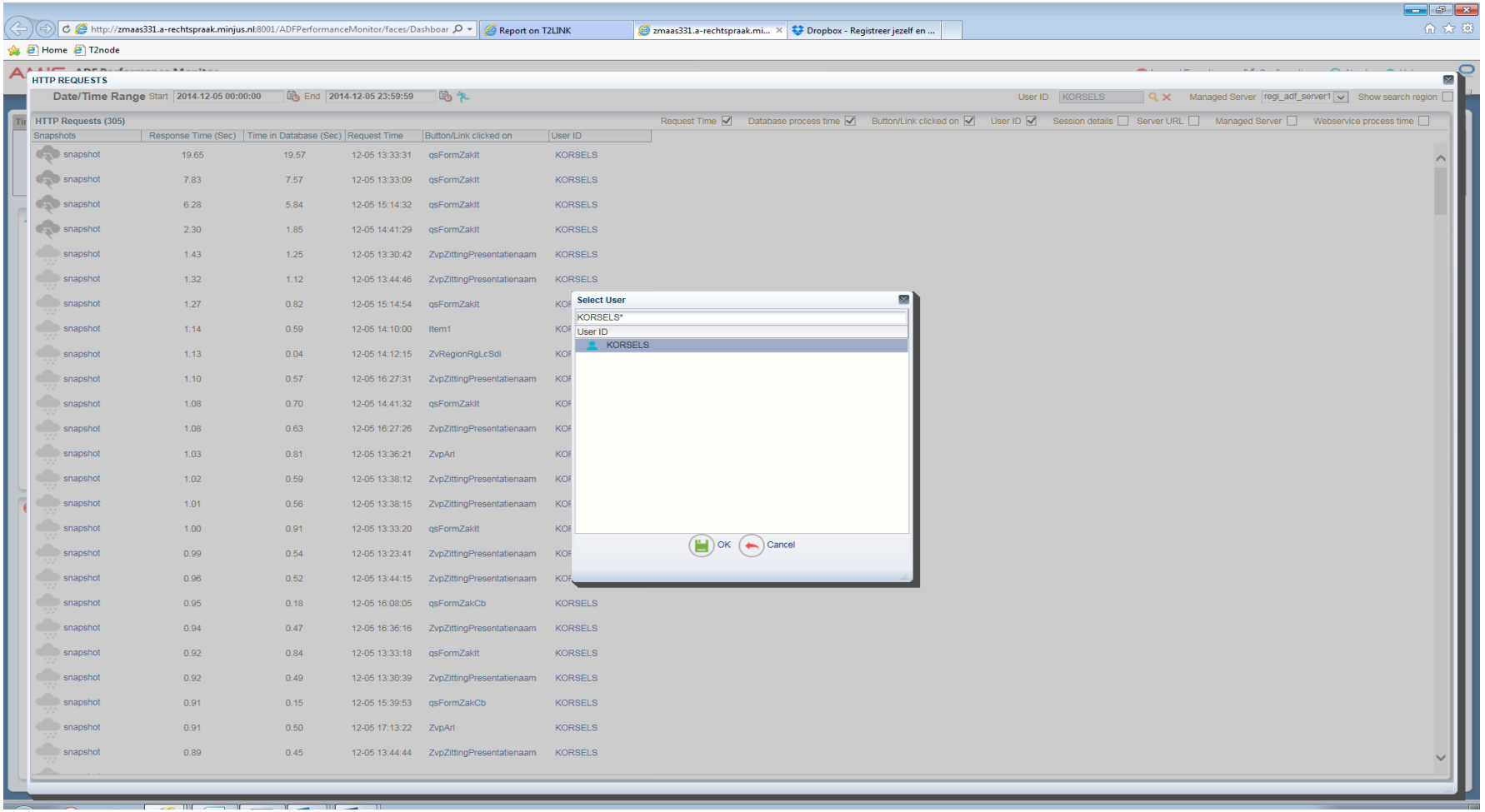
Monitor SLA - HTTP Request Response Time

- Example: from 15:00 to 16:00 the response times have increased significantly (more red and yellow colored parts in the 15u bar)
  - This should be a trigger to investigate the root cause



Monitor a specific End-users

- Problems of specific end-users can be analyzed



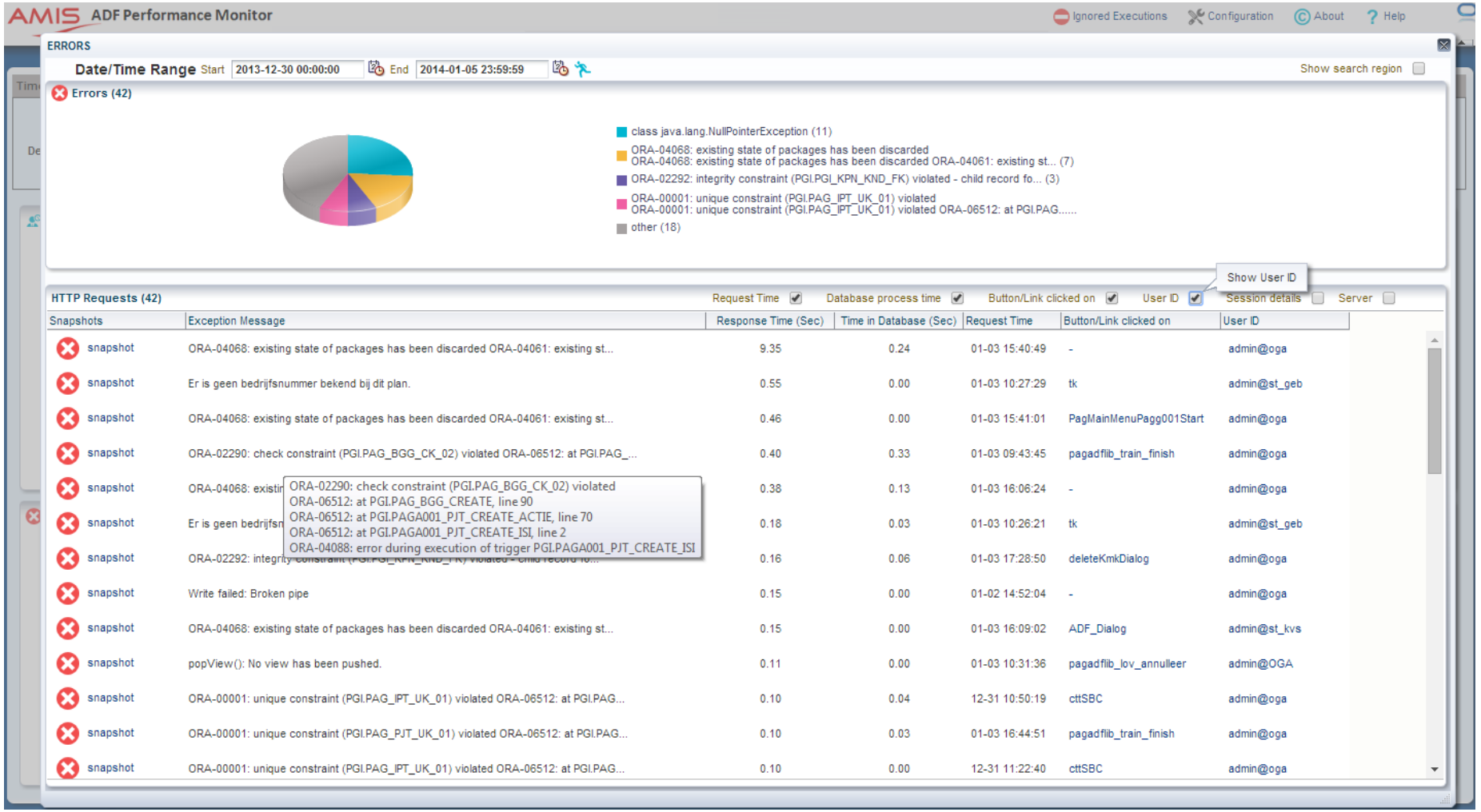
Case 2
Instrumenting Errors / Exceptions

- Configure a custom *MetricsDCErrorHandler* that extends from *DCErrorHandlerImpl*:
  - http://technology.amis.nl/2014/04/27/adf-performance-monitor-troubleshooting-test-and-production-errors/

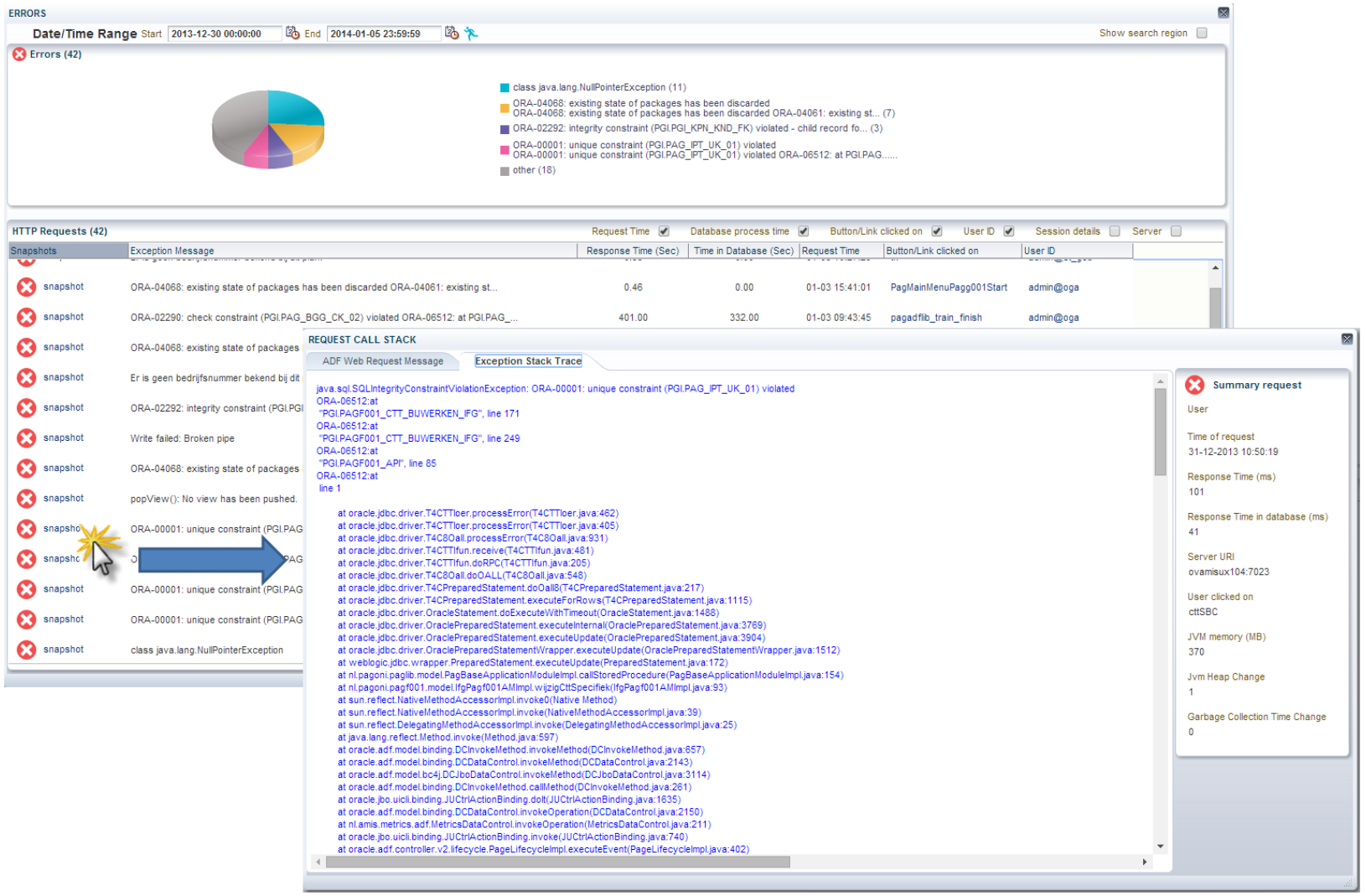
MetricsDCErrorHandlerImpl.java

```
1  package nl.amis.metrics.adf;
2
3  import ...;
5
6
7  public class MetricsDCErrorHandlerImpl extends DCErrorHandlerImpl {
8
9      public MetricsDCErrorHandlerImpl(boolean b) {
10         super(b);
11     }
12
13     public MetricsDCErrorHandlerImpl() {
14         super(true);
15     }
16
17     @Override
18     public void reportException(DCBindingContainer dCBindingContainer, Exception exception) {
19         boolean active = MetricsUtils.isActive();
20         if (active) {
21             String message = exception.getMessage();
22             if (message != null && !(message.contains("ORA-20"))) {
23                 MetricsUtils.logException(exception, false);
24             }
25         }
26         super.reportException(dCBindingContainer, exception);
27     }
28 }
```

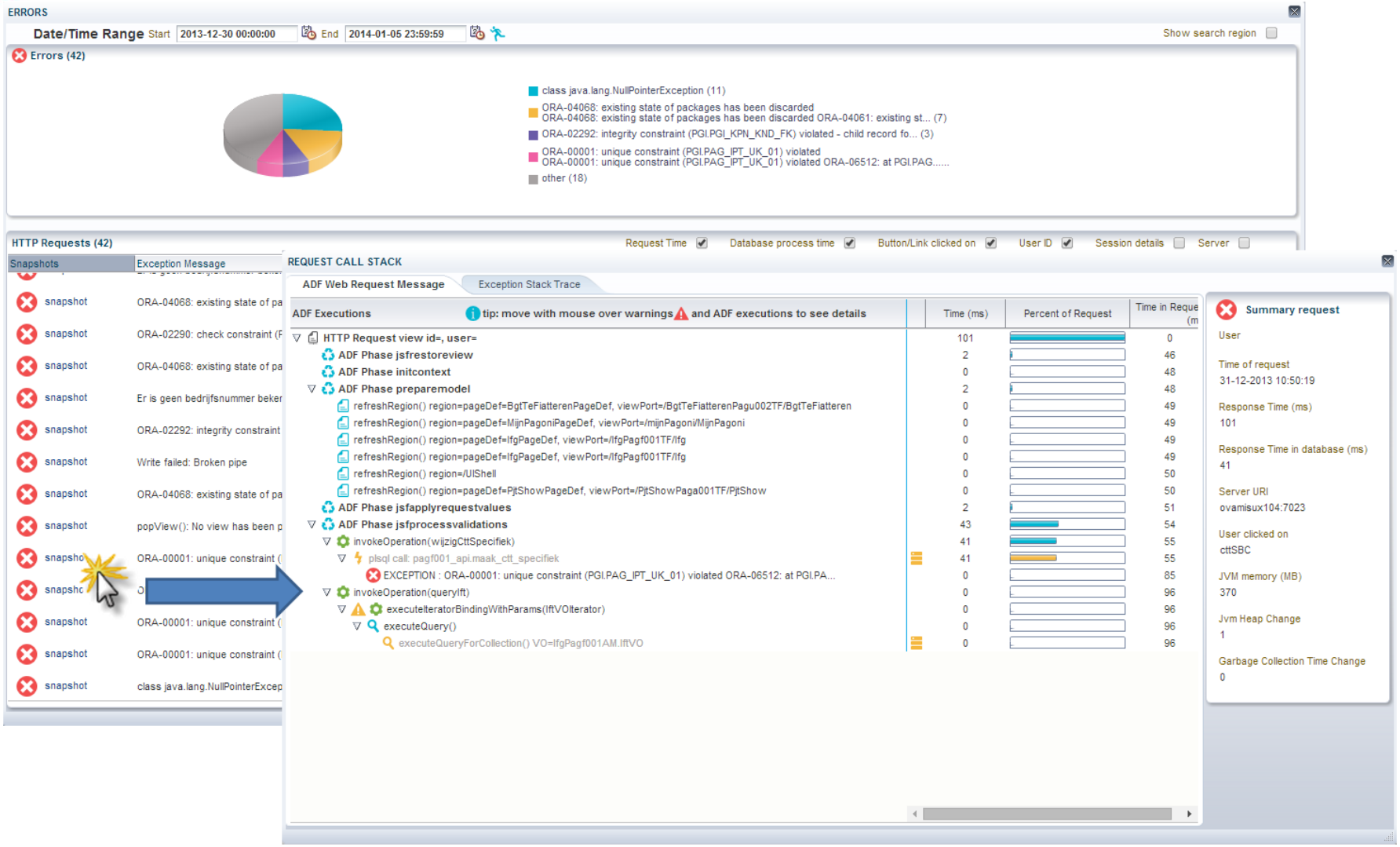
Errors (1) - Overview



Errors (2) - Stacktrace

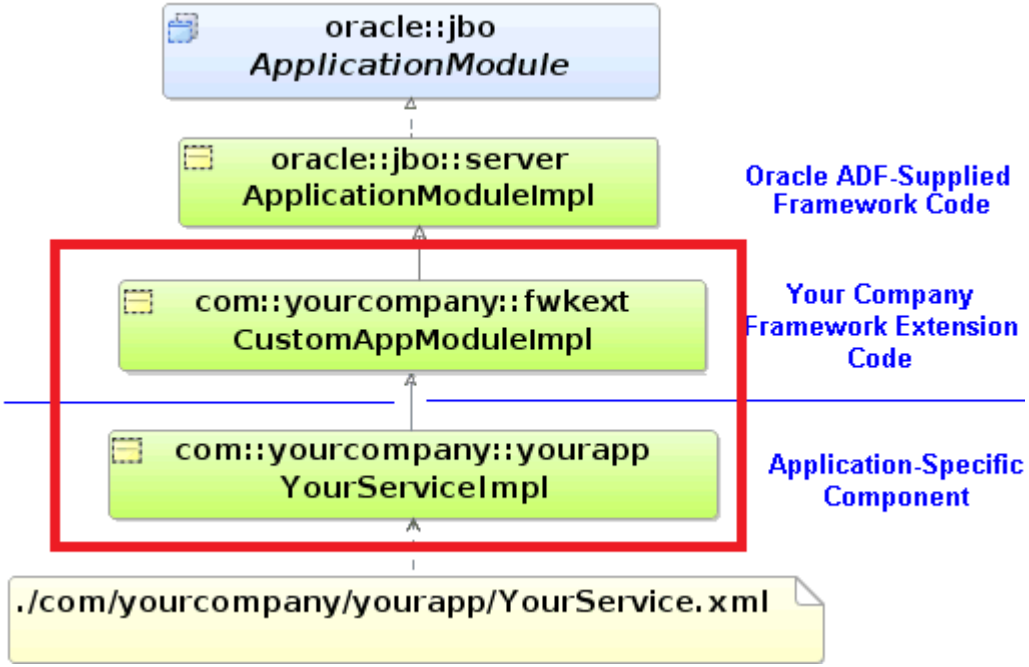


Errors (3) - ADF Callstack



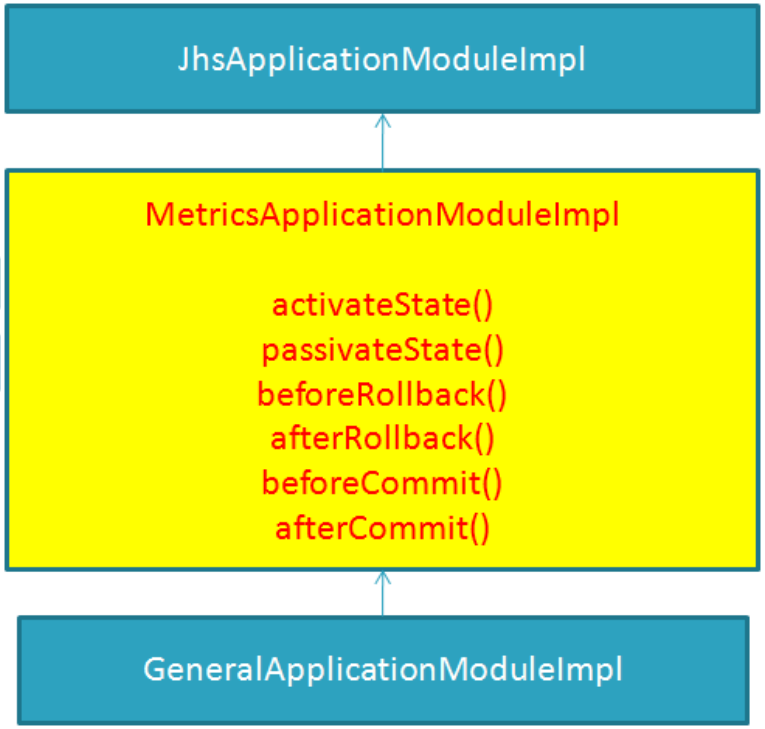
Instrument ADF BC

- Instrument with the use of ADF BC framework extension classes
  - Use your project or company base classes (or create a dedicated metrics package with extension classes)
  - Override specific methods to measure key actions inside the ADF framework



Creating a Metrics ApplicationModule

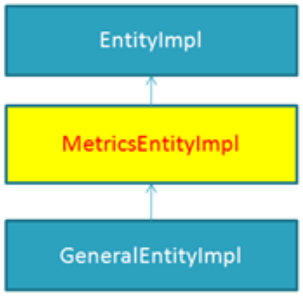
- Create a dedicated “metrics ApplicationModuleImpl” that extends from *oracle.jbo.server.ApplicationModulempl*
- Your project base class extends from this *MetricsApplicationModulempl*:

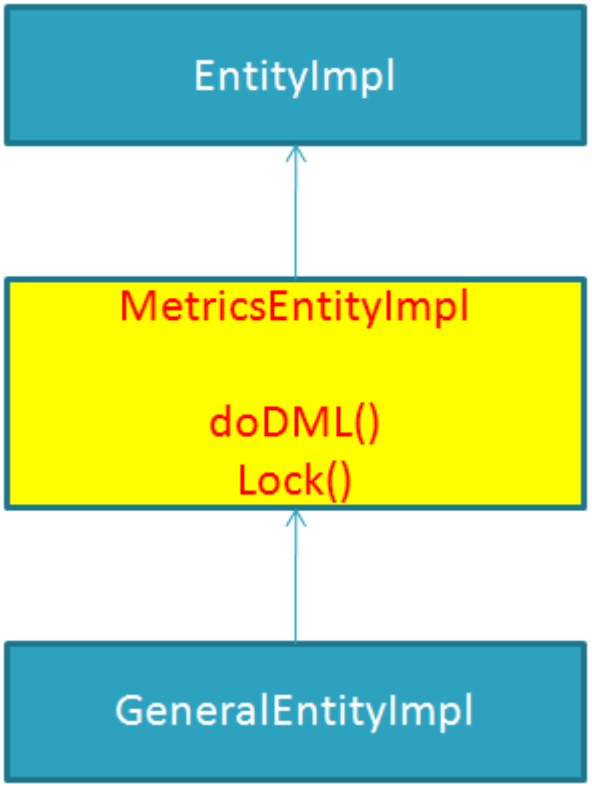


Creating a Metrics EntityObject

- Create a dedicated “metrics EntityObject” that extends from *oracle.jbo.server.ViewObjectImpl*
- Your project base class extends from this *MetricsViewObject*:

```
xml | GeneralViewObjectImpl.java | GeneralEntityImpl.java | MetricsApplica
1  package nl.amis.performance.model.framework;
2
3  import nl.amis.metrics.adf.MetricsEntityImpl;
4
5  public class GeneralEntityImpl extends MetricsEntityImpl {
6    public GeneralEntityImpl() {
7    }
8  }
```



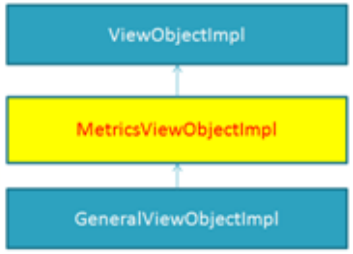


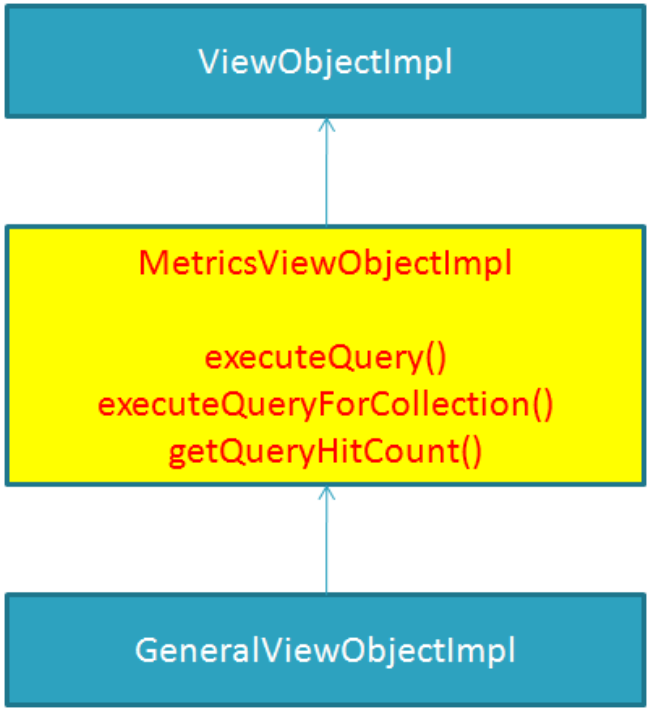
Creating a Metrics ViewObject

- Create a dedicated “metrics ViewObject” that extends from *oracle.jbo.server.ViewObjectImpl*
- Your project base class extends from this *MetricsViewObject*:

xml | GeneralViewObjectImpl.java | MetricsApplicationModuleImpl.java | GeneralApplic

```
1  package nl.amis.performance.model.framework;
2
3  import nl.amis.metrics.adf.MetricsViewObjectImpl;
4
5  public class GeneralViewObjectImpl extends MetricsViewObjectImpl {
6    public GeneralViewObjectImpl() { }
7  }
```





Look for Key Methods to instrument

- You can look at the *ViewObjectImpl* API doc for important and interesting methods to instrument & monitor

Overview Package Class Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES All Classes

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Oracle F

oracle.jbo.server

Class ViewObjectImpl

```
java.lang.Object
  └ oracle.jbo.common.NamedObjectImpl
      └ oracle.jbo.server.NamedObjectImpl
          └ oracle.jbo.server.ComponentObjectImpl
              └ oracle.jbo.server.ViewObjectImpl
```

All Implemented Interfaces:

java.util.EventListener, ViewCriteriaClauseBuilder, ViewCriteriaManagerOwner, ViewObject
NavigatableRowIterator, OperationContainer, Properties, RowIterator, RowMatchBindValueSt
EntityListener, TransactionListener, TransactionPostListener, StructureDef, VariableManagerC

Direct Known Subclasses:

DCDataVO, DSViewObjectImpl, FinderViewImpl, PlaceholderVOImpl, ViewObjectOnRowS

```
public class ViewObjectImpl
extends ComponentObjectImpl
implements ViewObject, EntityListener, RowSetListener, TransactionListener, TransactionPost
```

The implementation of the `ViewObject` interface, the middle-tier class that manages database queries a

A View Object manages a view row set (`ViewRowSetImpl`). When the application calls a `RowSet` metho

RowSet	deepCopy(java.util.HashMap voAttrMap, long options) This method delegates to the default RowSet.
void	defineNamedWhereClauseParam(java.lang.String name, java.lang.Object defau Defines a named bind variable to use with the view object's where-clause.
boolean	doesRowFilterMatch(Row[] masterRows, java.lang.Object[] rowFilterValues)
void	dumpQCs() **Internal:** *Applications should not use this method.*
Enumeration	enumerateRowsInRange() Creates and returns an enumerator of the rows in the range.
ng.Object[]	executeDetailQuery(Row[] masterRows)
void	executeEmptyRowSet()
void	executeQuery() Executes the query.
ected void	executeQueryForCollection(java.lang.Object qc, java.lang.Object[] params, This method is invoked right before the row set executes the query.
KeyDef	findAltKeyDef(java.lang.String keyName)
void	findAndSetCurrentRowByKey(Key key, int rangeIndex)

Case 3
Monitor SQL queries

- Override the ViewObject method *executeQueryForCollection()*
  - Which SQL queries are executed, their execution time, when and how often ?
  - Including (detail) queries from ViewAccessors, ViewLinks, programmatic ViewObjects, e.g.

```
public class SummitViewObjectImpl extends ViewObjectImpl {

    private static final ADFLogger logger = ADFLogger.createADFLogger(SummitViewObjectImpl.class);

    protected void executeQueryForCollection(Object object, Object[] objectl, int i) {
        long startTime = System.nanoTime();
        super.executeQueryForCollection(object, objectl, i);
        long totalTime = (System.nanoTime() - startTime) / 1000000;
        logger.fine("VO DATABASE EXECUTION VO usagename: " + getName() + " millis: " + totalTime);
    }
```

start

stop

select *
from customers

Database

See
http://technology.amis.nl/2014/03/11/adf-performance-tuning-instrumenting-your-adf-application-part-1/

```
VIEWOBJECT DATABASE EXECUTION VO usagename: Customers millis: 9
VIEWOBJECT DATABASE EXECUTION VO usagename: _LOCAL_VIEW_USAGE_oracle_summit_model_views_CustomerVO_CreditRatingVO1 millis: 0
VIEWOBJECT DATABASE EXECUTION VO usagename: _LOCAL_VIEW_USAGE_oracle_summit_model_views_CustomerVO_CountryVO1 millis: 0
VIEWOBJECT DATABASE EXECUTION VO usagename: OrdersForCustomer millis: 4
```

Instrumenting Tips

- To ensure that your start and stop code is executed in case an error occurs, use a **try-finally** block
- Be sure that your logging can be turned on and off at runtime
- Create a class that is responsible for starting/stopping logging methods
  - Retrieving the current nanotime just before and immediately after the method execution by calling *System.nanoTime()*
  - *System.currentTimeMillis()* can be a few milliseconds inaccurate (!)

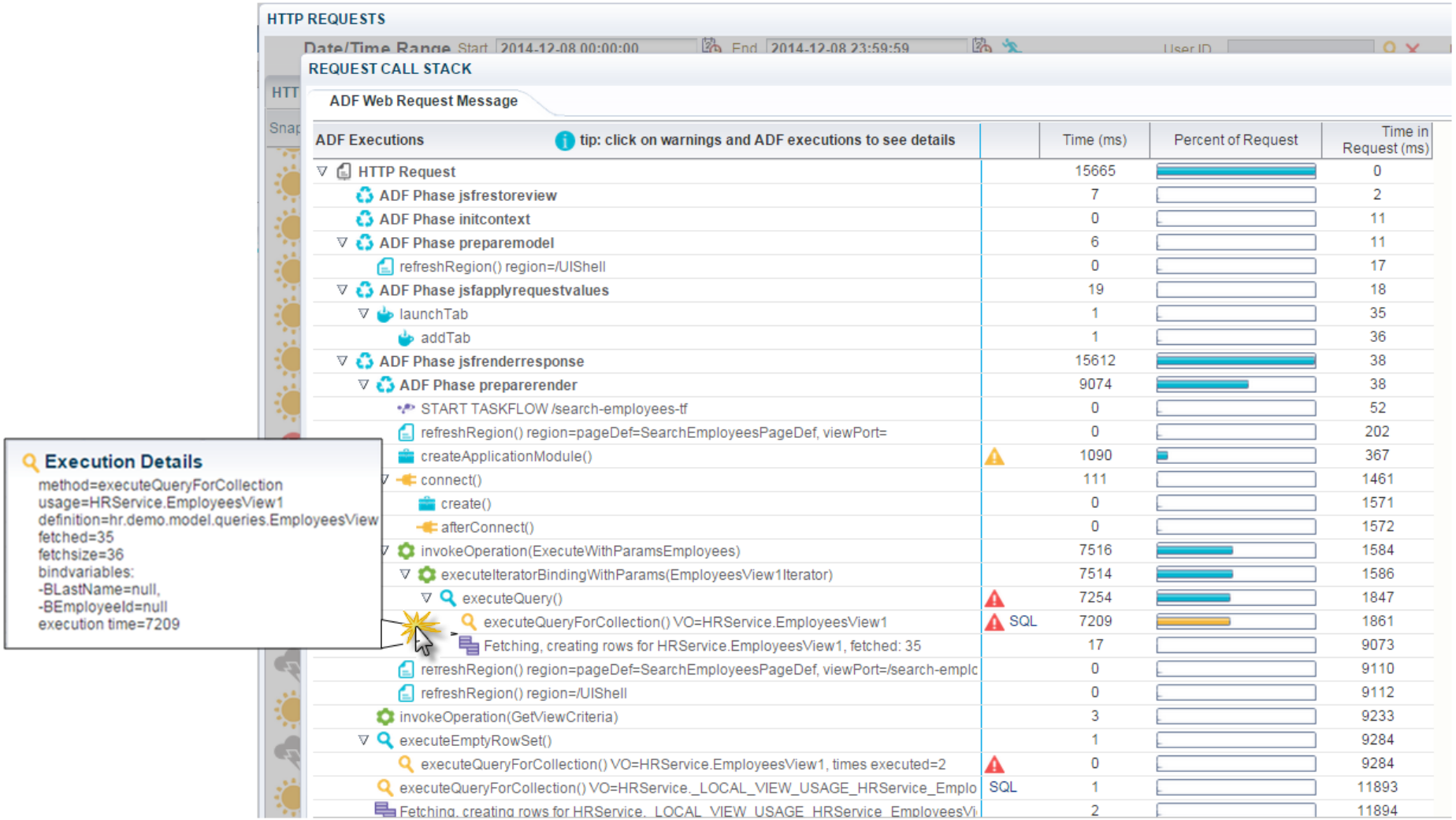
```
protected void executeQueryForCollection(Object object, Object[] objectl, int i) {

    boolean active = MetricsUtils.isActive();
    MetricsVOExecution metricsExecution = null;
    if (active) {
        metricsExecution = new MetricsVOExecution();
        cached = 0;
    }
    try {
        super.executeQueryForCollection(object, objectl, i);
    } finally {
        if (active)
            metricsExecution.stop(METHOD_EXECUTEQUERYFORCOLLECTION, this, objectl);
    }
}
```

Additional ViewObject Runtime Metrics (1)

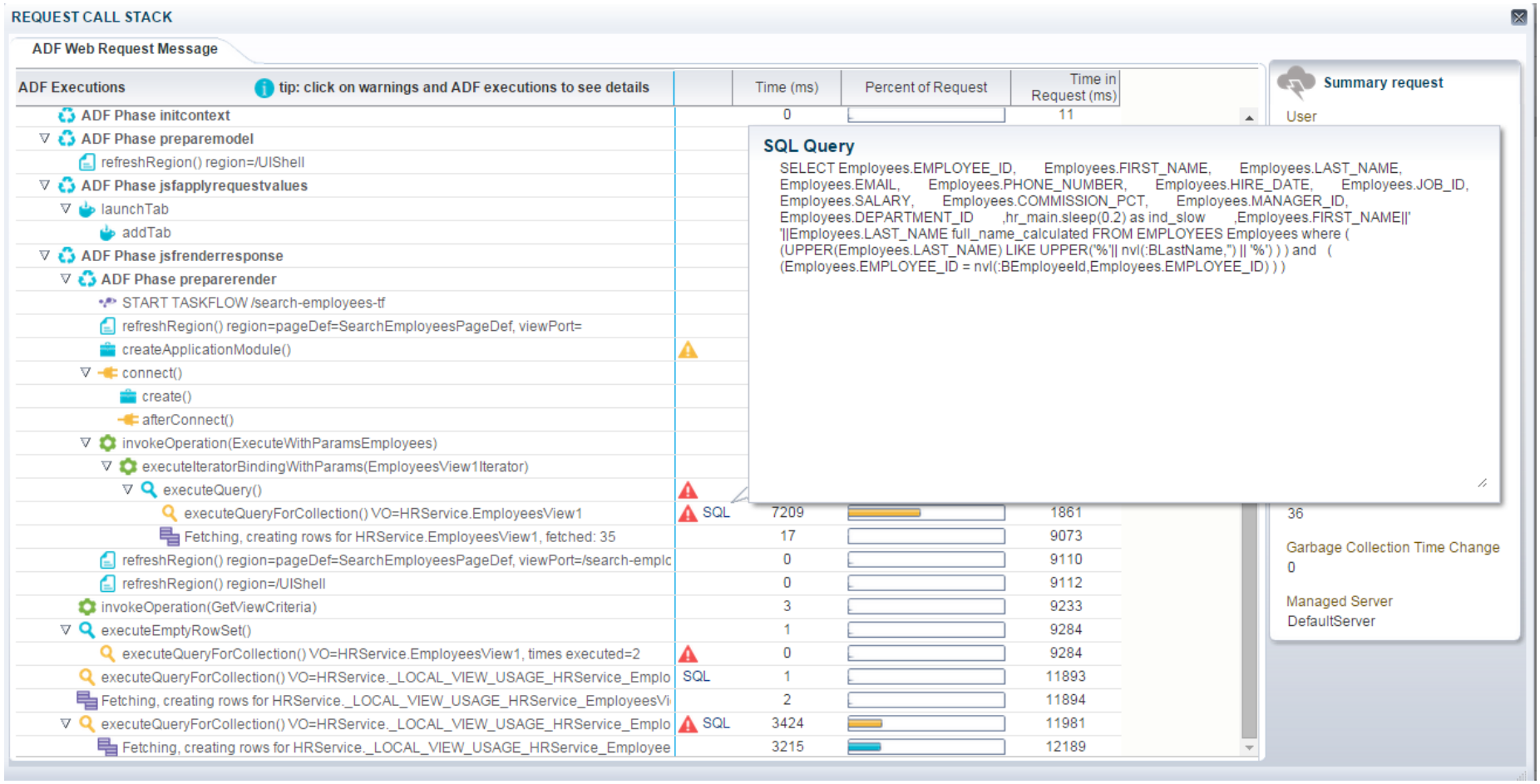
- Possible additional ViewObject runtime metrics
  - Bind variable names / values
  - Fetched rows loaded into ADF app
  - Applied ViewCriterias
  - SQL statement

```
/**
 * Queries from ViewObjectImpl
 * stop method for ViewObject.executeQueryForCollection()
 */
public void stop(int method, MetricsViewObjectImpl viewObjectImpl, Object[] params) {
    super.stop();
    this.method = method;
    bindVars = getParams(params);
    viewCriterias = getAppliedViewCriterias(viewObjectImpl);
    definitionName = viewObjectImpl.getDefFullName();
    usageName = viewObjectImpl.getApplicationModule().getName() + "." + viewObjectImpl.getName();
    fetched = viewObjectImpl.getCached();
}
```

ADF Callstack
Slow ViewObject Query

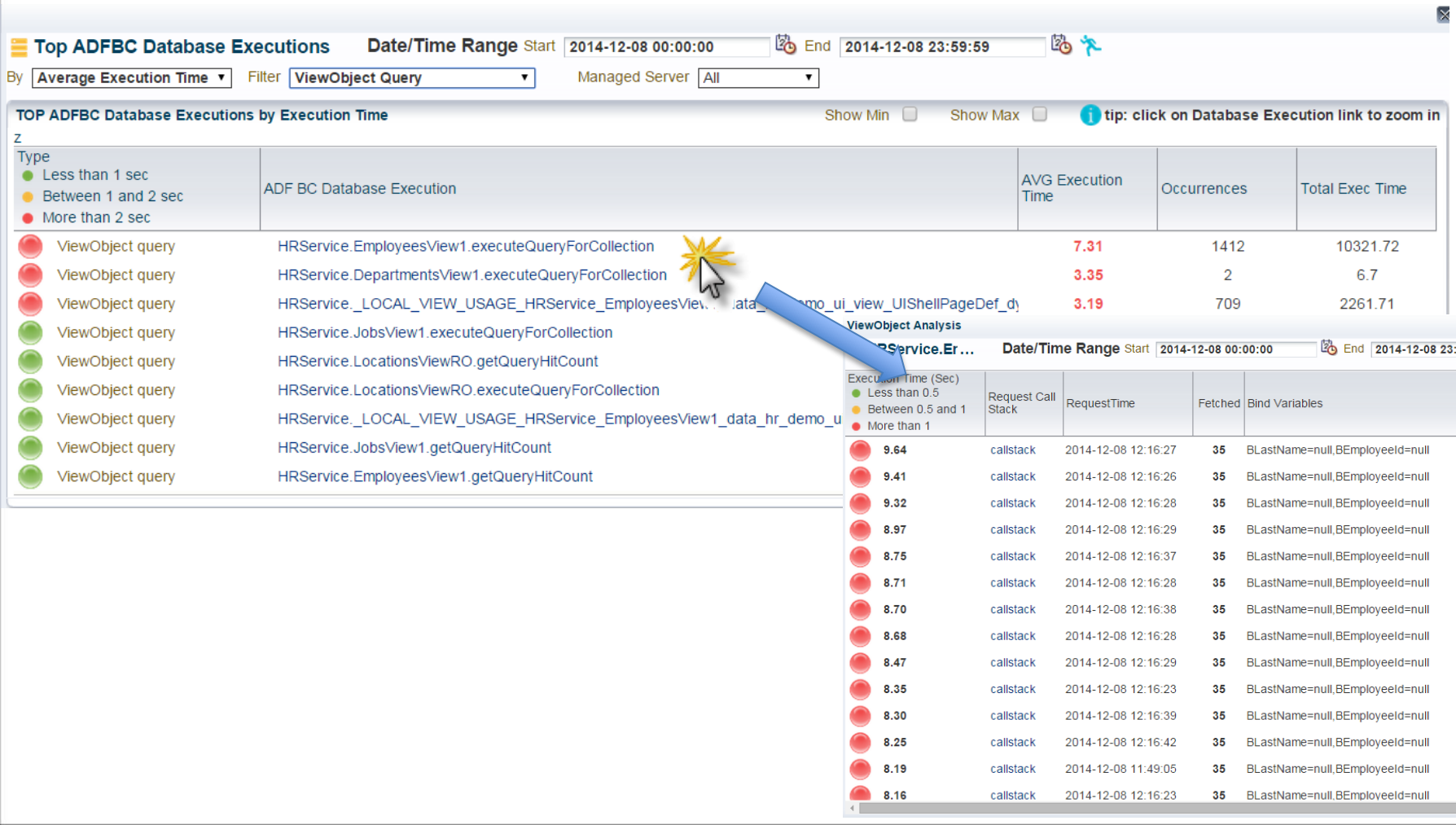


SQL Statement



ViewObject Query Performance Aggregated

- AVG, Occurrences, Total execution time, Min, Max



Case 4: Instrumenting how many rows are fetched/loaded

- Detecting an overload of ViewObject rows
  - Prevent JVM memory overconsumption (!)
  - http://technology.amis.nl/2014/02/03/adf-performance-monitor-detecting-and-analyzing-a-high-adf-bc-memory-consumption/

Instrumenting how many rows are fetched/loaded

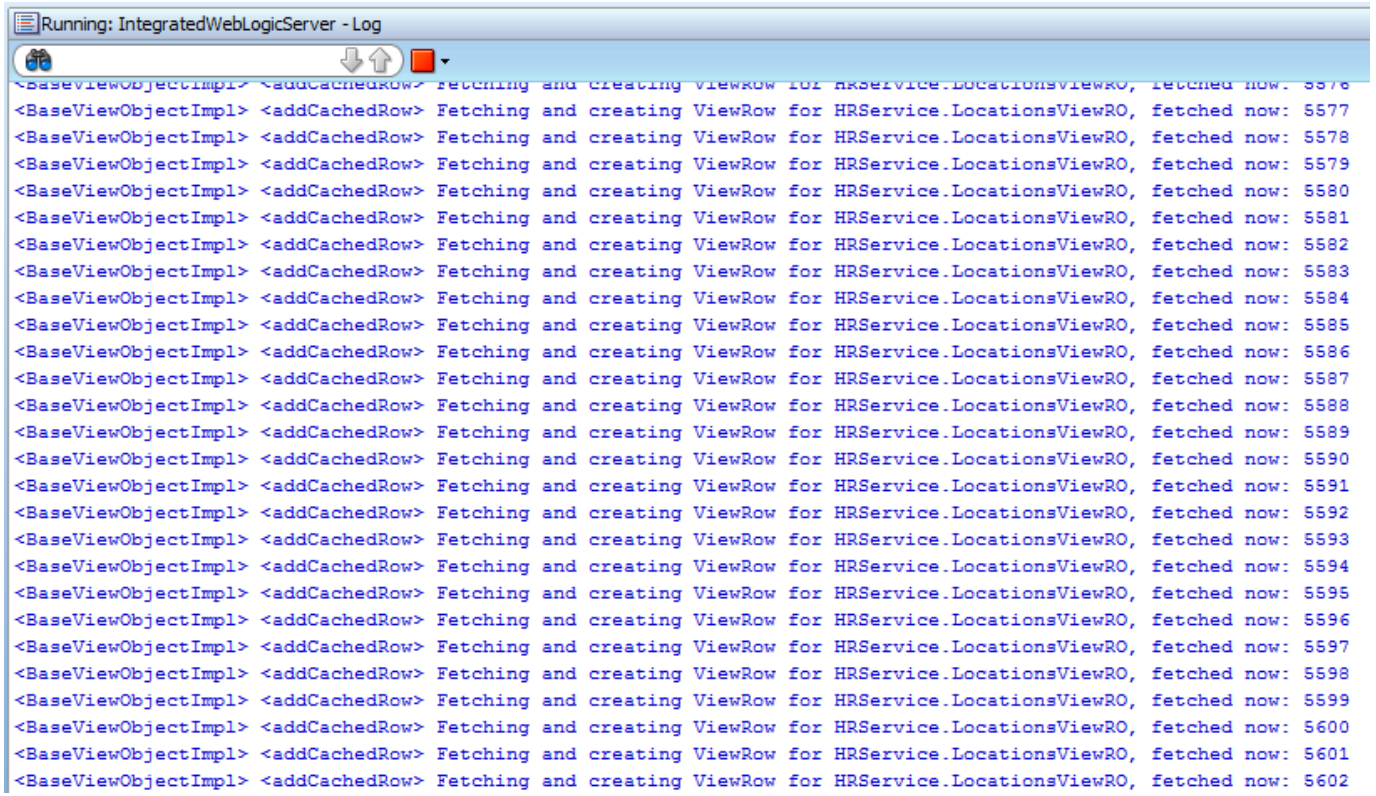
- Overriding the method *createRowFromResultSet()* and counting them in your *MetricsViewObjectImpl*.
  - Reset the counter before a query is executed (again):

BaseViewObjectImpl.java

```
10
11  public class BaseViewObjectImpl extends ViewObjectImpl {
12
13      private static final ADFLogger logger = ADFLogger.createADFLogger(BaseViewObjectImpl.class);
14      private int fetched = 0;
15
16      @Override
17      protected ViewRowImpl createRowFromResultSet(Object object, ResultSet resultSet) {
18          fetched++;
19
20          //Only log high load
21          if (fetched > 100) {
22              logger.finest("Fetching and creating ViewRow for " +
23                            this.getApplicationModule().getName() + "." + this.getName() +
24                            ", fetched now: " + fetched);
25          }
26          return super.createRowFromResultSet(object, resultSet);
27      }
28
29      @Override
30      protected void executeQueryForCollection(Object object, Object[] object2, int i) {
31          //After new ViewObject query reset fetched
32          fetched = 0;
33          super.executeQueryForCollection(object, object2, i);
34      }
35
```

Instrumenting how many rows are fetched/loaded

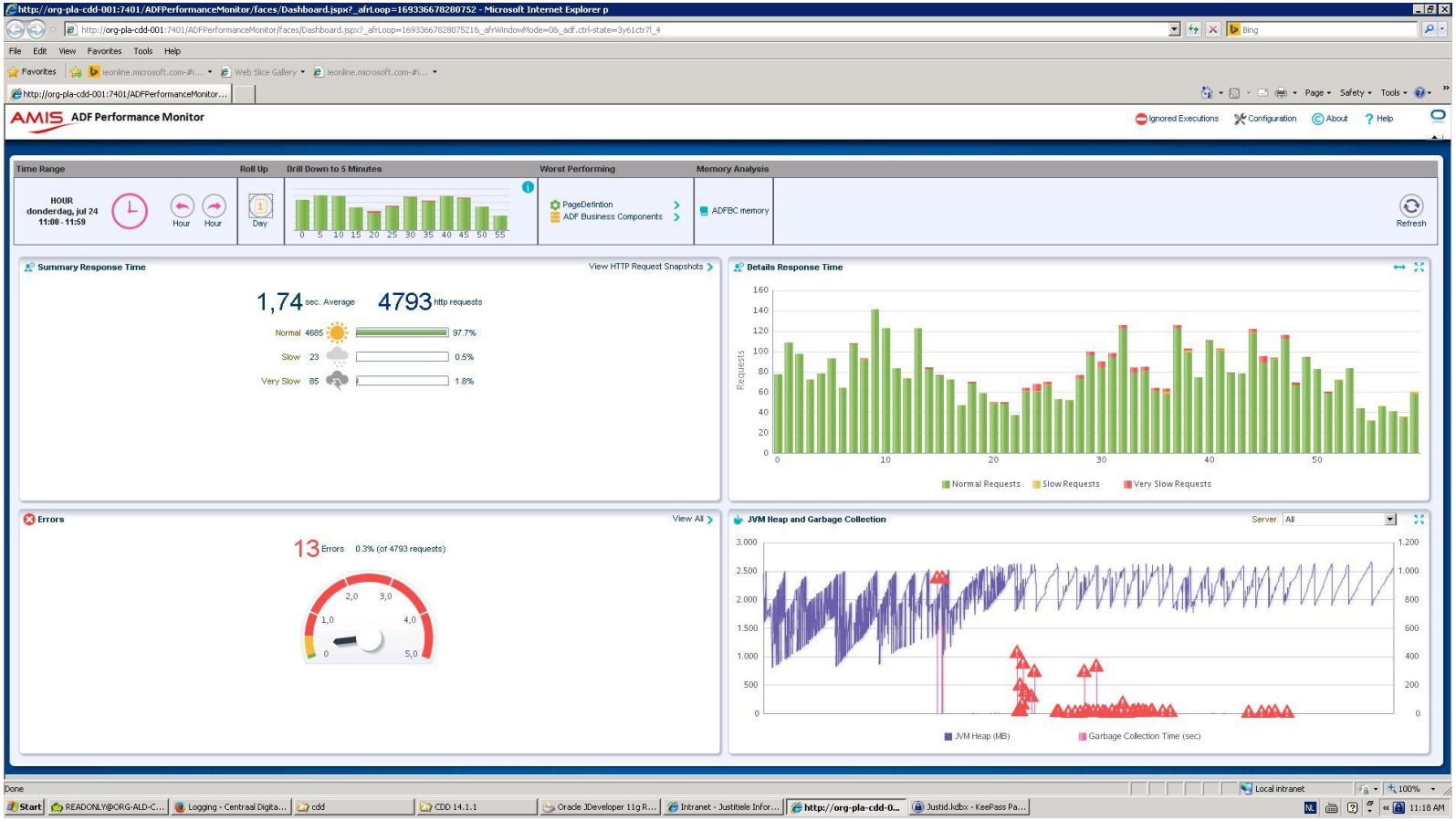
- A high load can now be detected in JDevelopers console log
  - You might want to not log every single row that is fetched like here but only above a certain threshold (for example for every hundred)



Loaded Rows in Memory (1)
JVM memory overconsumption

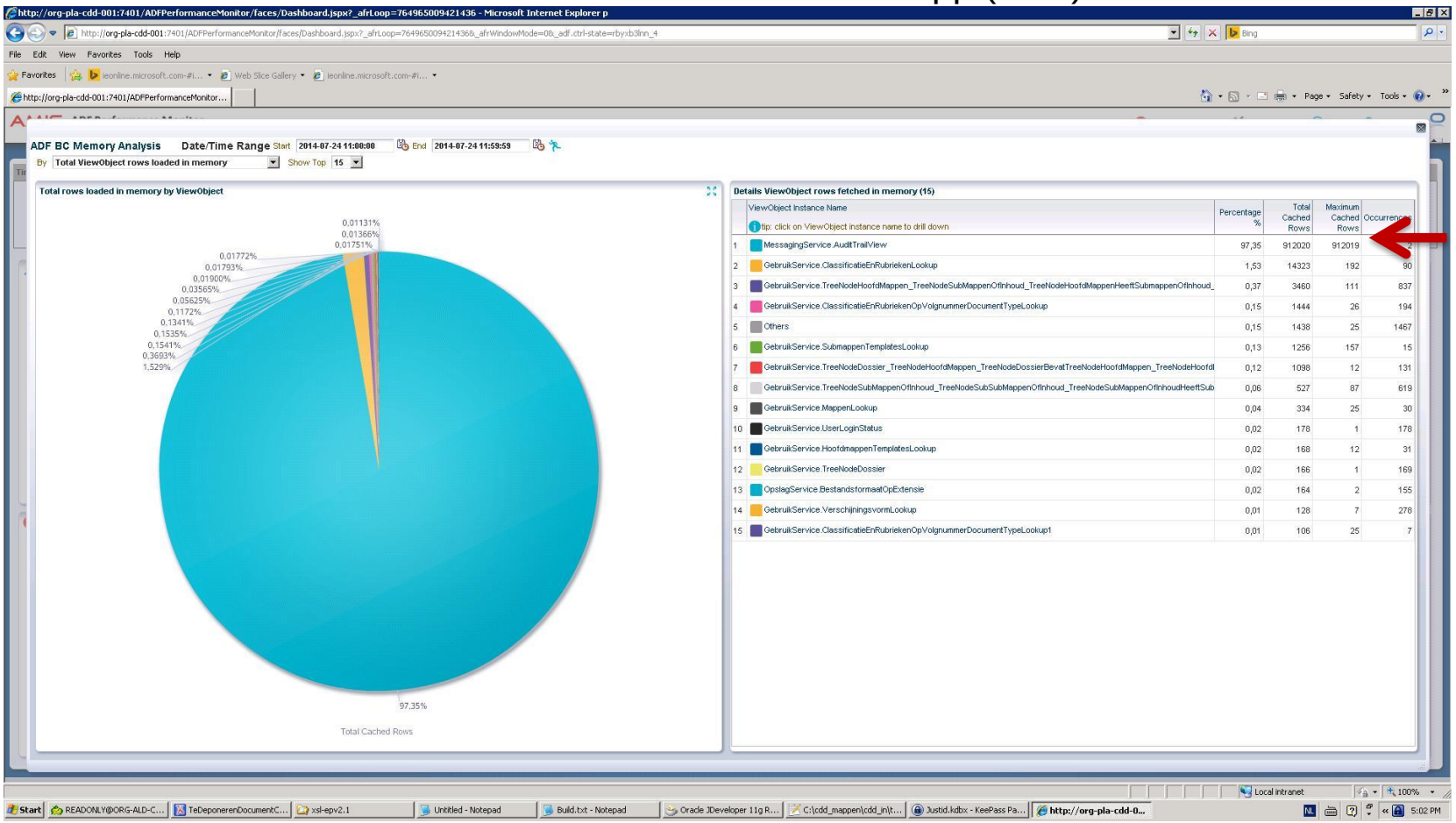
– Example: ADF application with JVM memory overconsumption

  We see many JVM warnings of long garbage collections; this should be a trigger to investigate



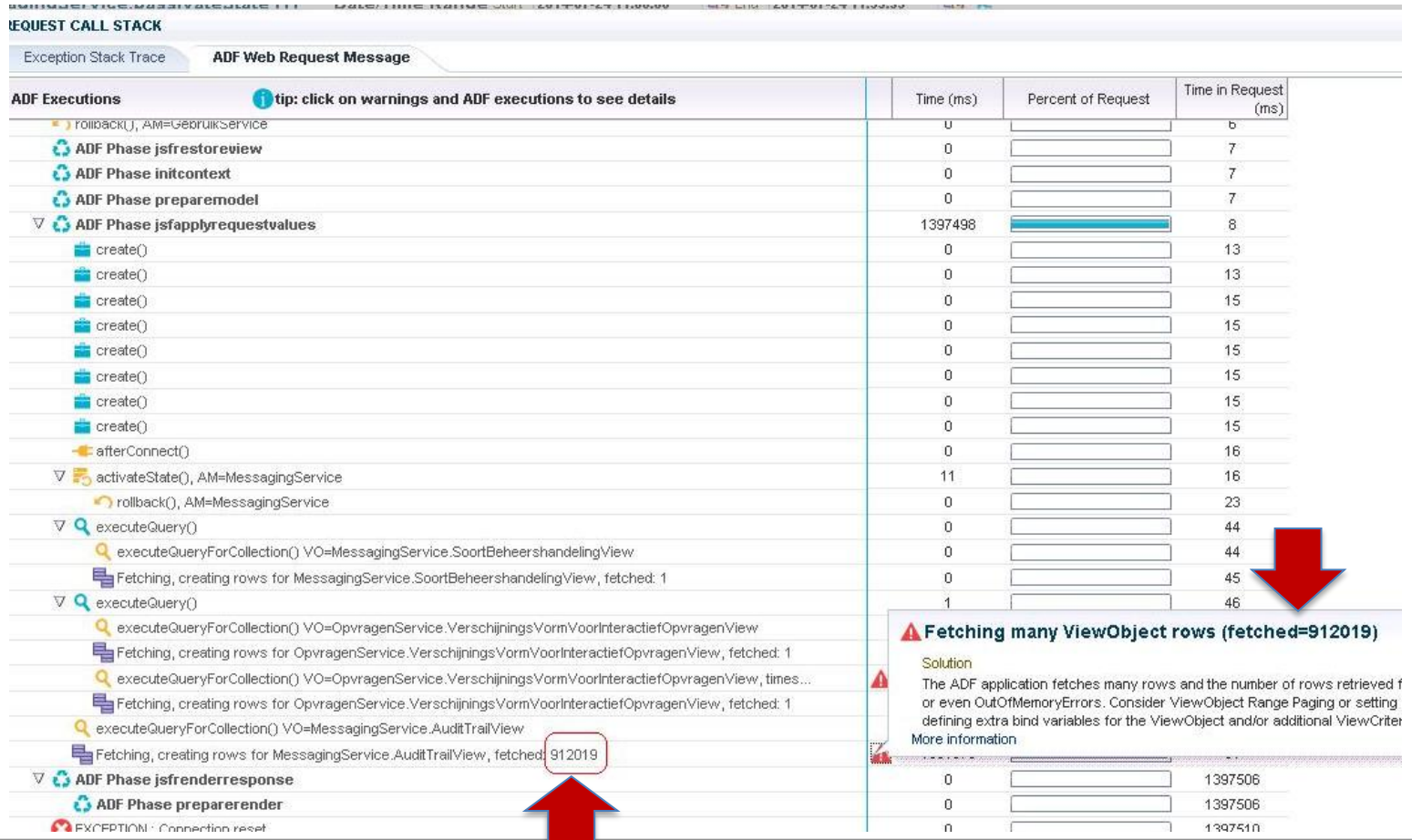
Loaded Rows in Memory (2)
ADF BC Memory Analyzer

- In this production ADF app, for a single ViewObject instance, more than **900.000** rows were loaded in the ADF app (Blue) !



Loaded Rows in Memory (3) ADF CallStack

- In this production ADF app, for a single ViewObject instance, more than **900.000** rows were loaded in the ADF app (!)
- Loading 900.000 rows takes more than 10 minutes

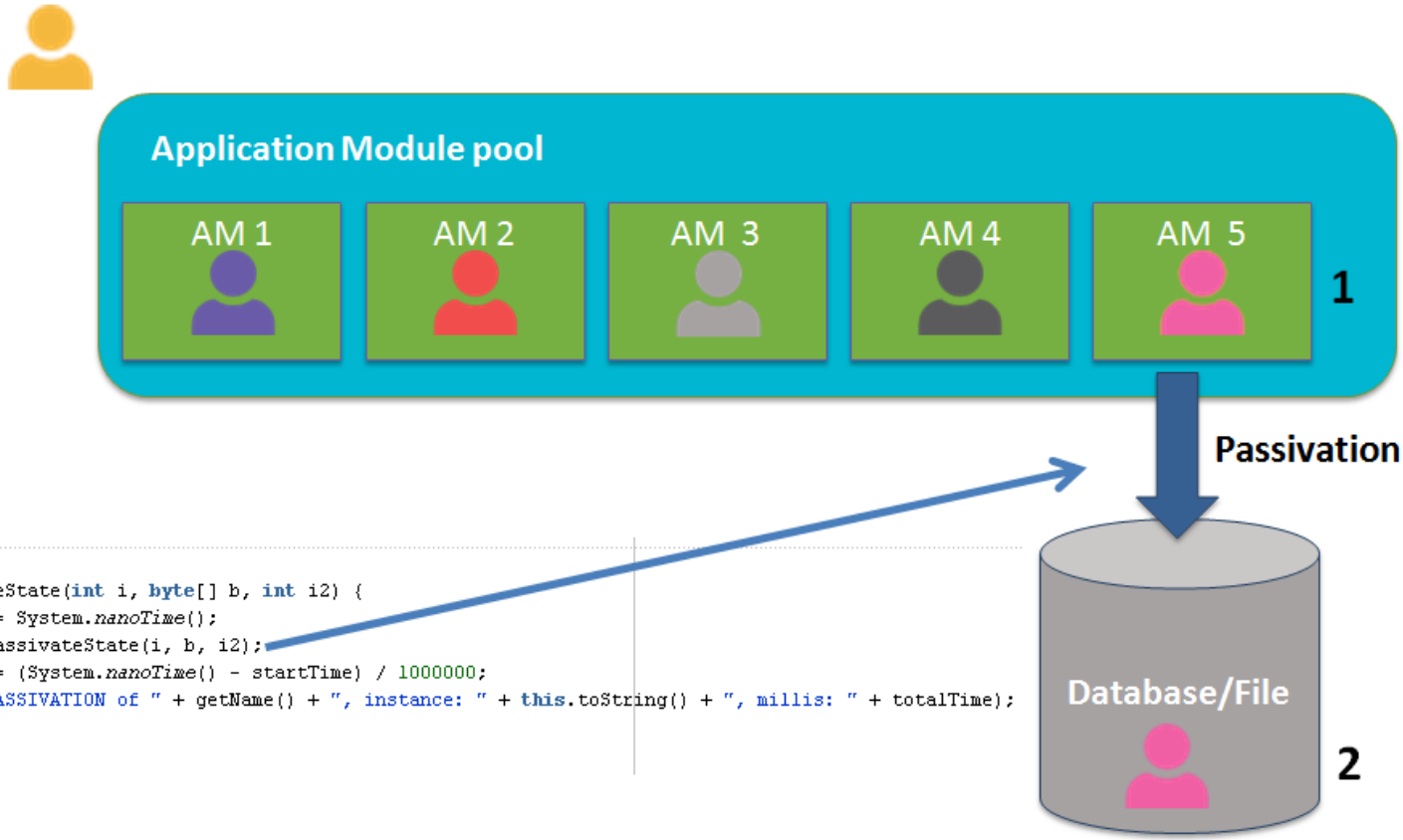


Case 5: Instrumenting AM activation and passivation

- ApplicationModule pooling enables multiple users to share several application module instances. It involves saving and retrieving session state data from the database, file, or Java memory.
- This mechanism is provided to make the application scalable and becomes very important under high load with many concurrent users.

Instrument ApplicationModule passivation

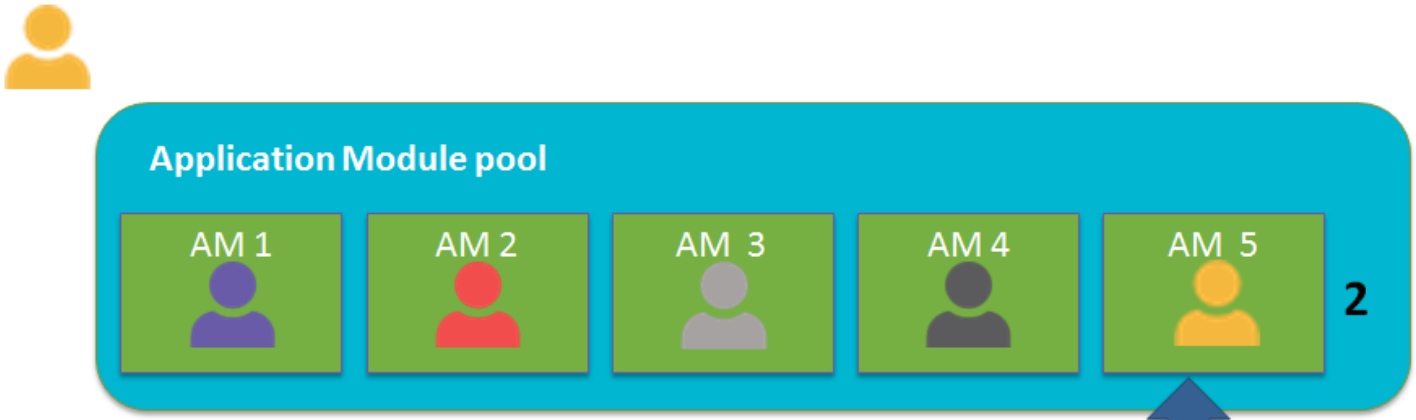
- Override the *ApplicationModule* method *passivateState()* in your metrics base class:
- See http://technology.amis.nl/2014/04/01/adf-runtime-diagnostics-instrumenting-your-adf-application-part-2/




```
@Override
public int passivateState(int i, byte[] b, int i2) {
    long startTime = System.nanoTime();
    int y = super.passivateState(i, b, i2);
    long totalTime = (System.nanoTime() - startTime) / 1000000;
    sLog.fine("AM PASSIVATION of " + getName() + ", instance: " + this.toString() + ", millis: " + totalTime);
    return y;
}
```

Instrument ApplicationModule activation

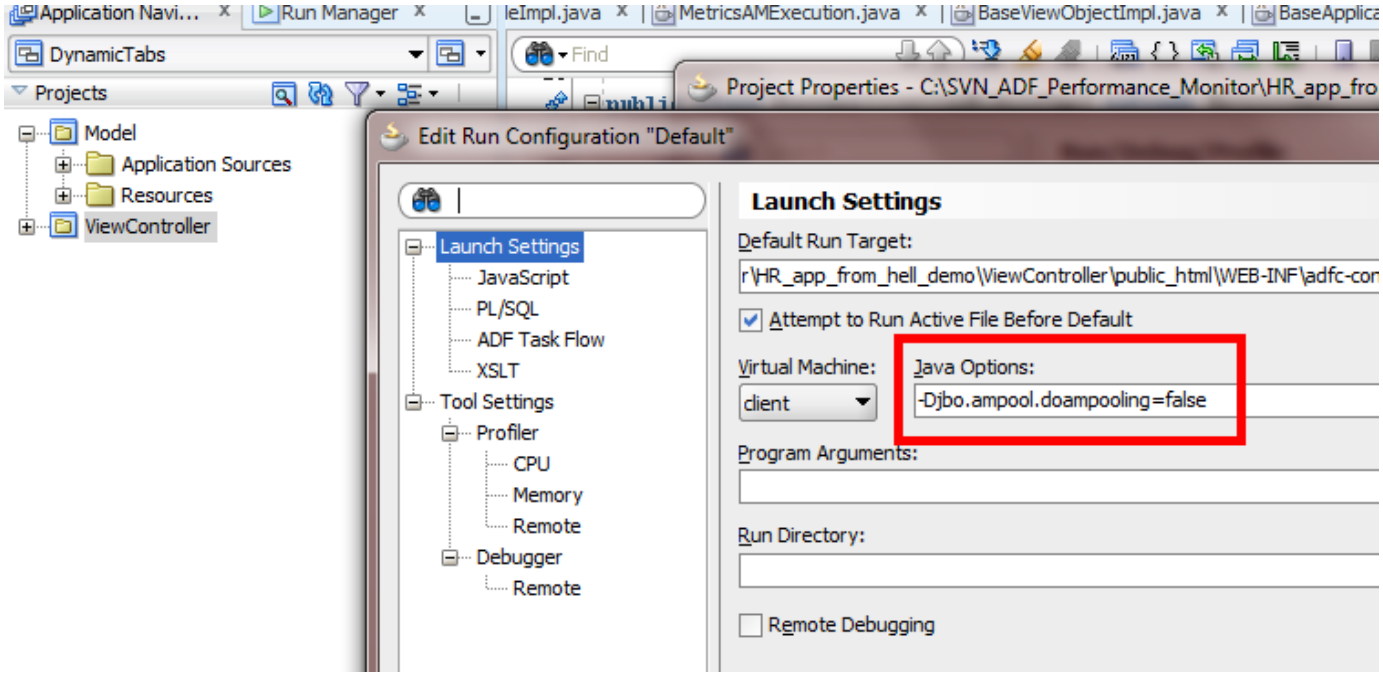
- Override the *ApplicationModule* method *activateState()* in your metrics base class:




```
26
   public class BaseApplicationModuleImpl extends MetricsApplicationModuleImpl implements BaseApplicationModule {
28
       private static final ADFLogger sLog = ADFLogger.createADFLogger(BaseApplicationModuleImpl.class);
30
       @Override
32     public byte[] activateState(int i, SessionData sessionData, int i2) {
33         long startTime = System.nanoTime();
34         byte[] bt = super.activateState(i, sessionData, i2);
35         long totalTime = (System.nanoTime() - startTime) / 1000000;
36         sLog.fine("AM ACTIVATION of " + getName() + ", instance: " + this.toString() + ", millis: " + totalTime);
37         return bt;
38     }
```

Test Instrumentation Activation and Passivation

- In this way you can monitor which ApplicationModules are activated/passivated, their activation/passivation time, when and how often it is executed.
- Run your ADF application with **Djbo.ampool.doampooling=false** to test this



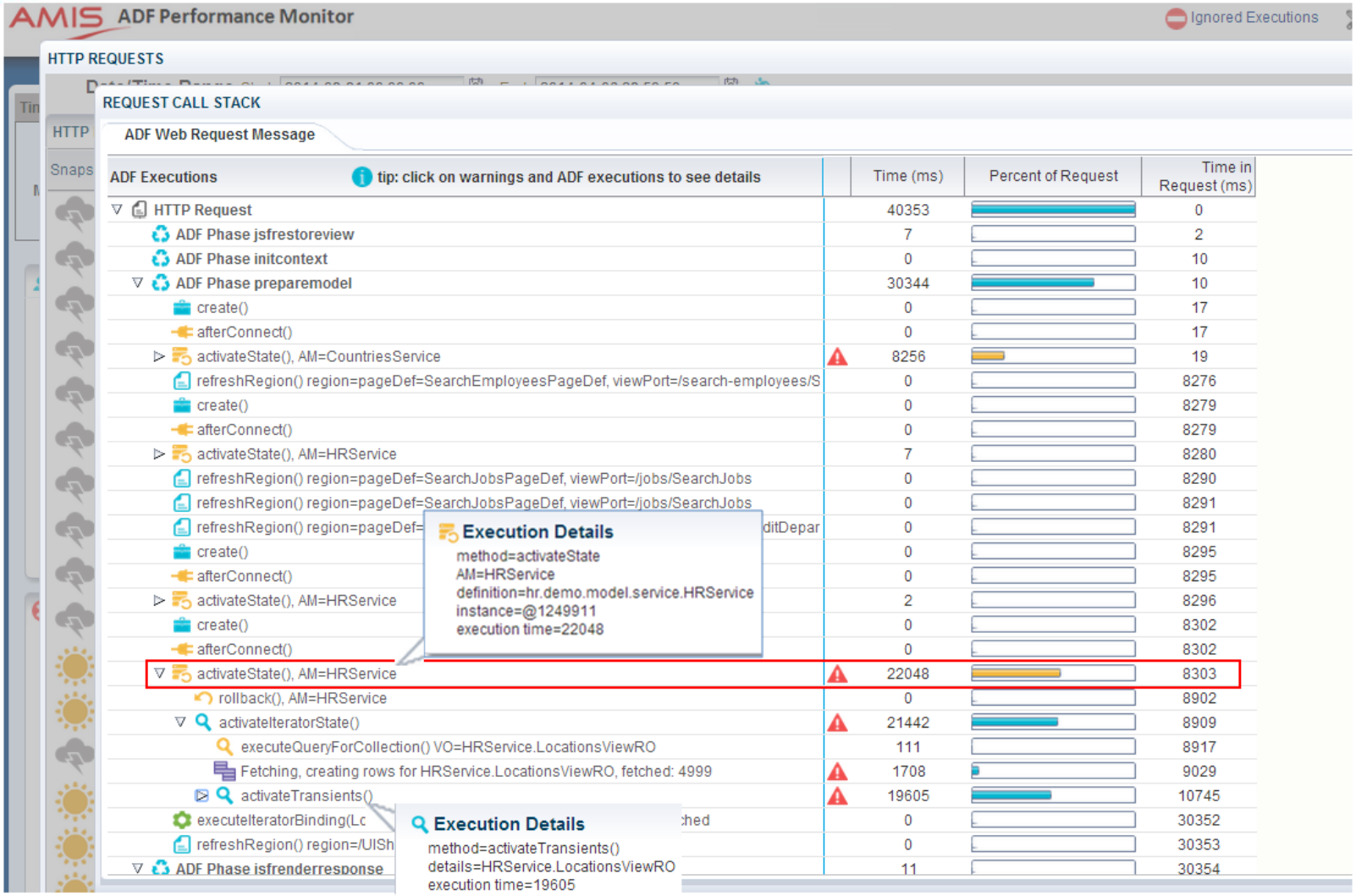
Test ApplicationModule Activation and Passivation

- Logging JDevelopers console log:

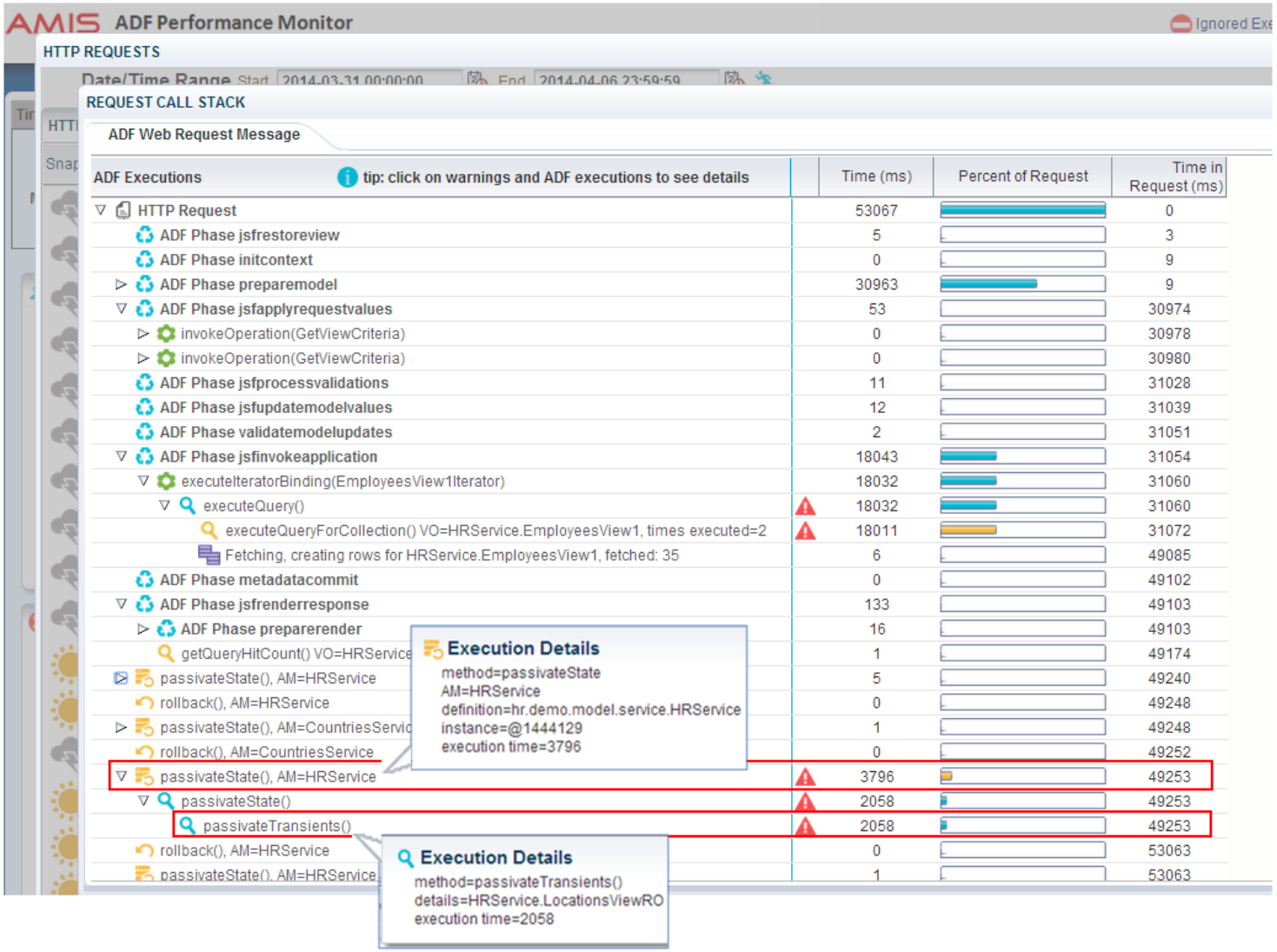
Running: IntegratedWebLogicServer - Log

```
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@1f0db8c, millis: 1
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@66d486, millis: 18162
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@66d486, millis: 11600
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@12b5481, millis: 2
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@1c249ea, millis: 1
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@1f0db8c, millis: 0
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@14700c, millis: 8148
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@2beab3, millis: 3
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@1f47af2, millis: 1
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@31e009, millis: 19233
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@31e009, millis: 3256
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@2beab3, millis: 17
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@14700c, millis: 4
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@1f47af2, millis: 4
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@1e4f021, millis: 8532
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@174c1f, millis: 8
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@26d30b, millis: 1
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@15ef8b, millis: 19354
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@15ef8b, millis: 6224
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@174c1f, millis: 2
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@1e4f021, millis: 2
<BaseApplicationModuleImpl> <passivateState> AM PASSIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@26d30b, millis: 1
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of CountriesService, instance: hr.demo.model.service.CountriesServiceImpl@1f5ea1c, millis: 7534
<BaseApplicationModuleImpl> <activateState> AM ACTIVATION of HRService, instance: hr.demo.model.service.HRServiceImpl@54bdb5, millis: 3
```

Example: Slow Activation of ApplicationModule

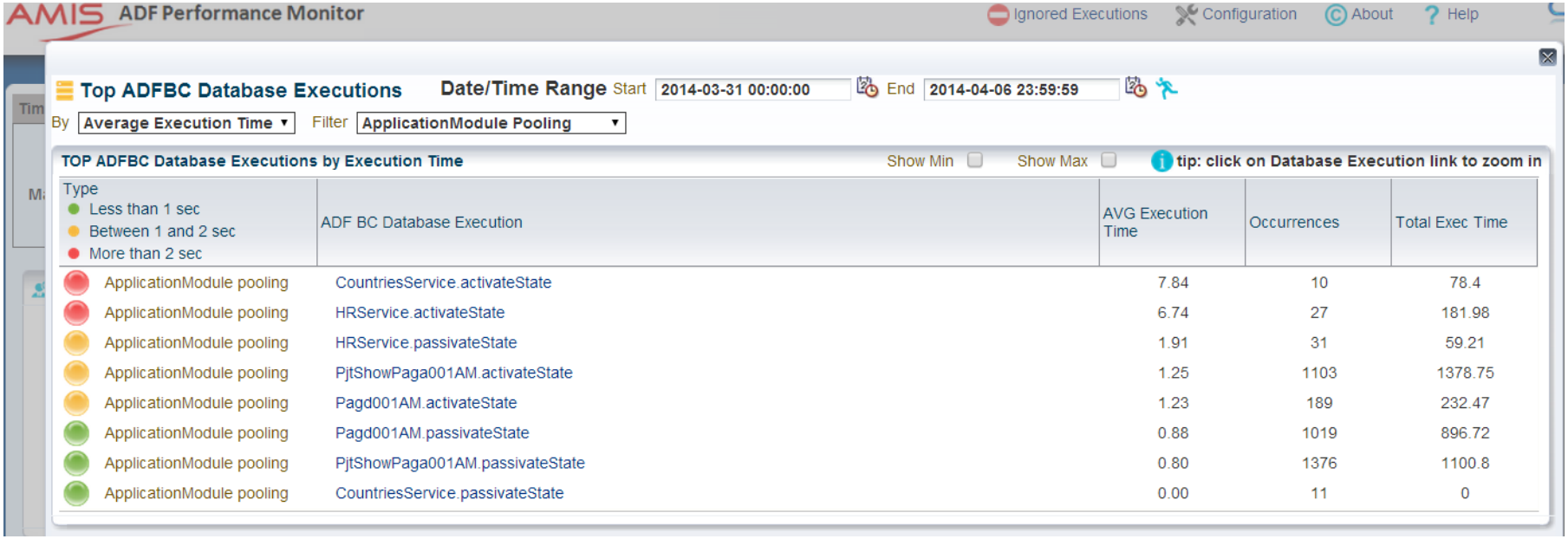


Example: Slow Passivation of ApplicationModule

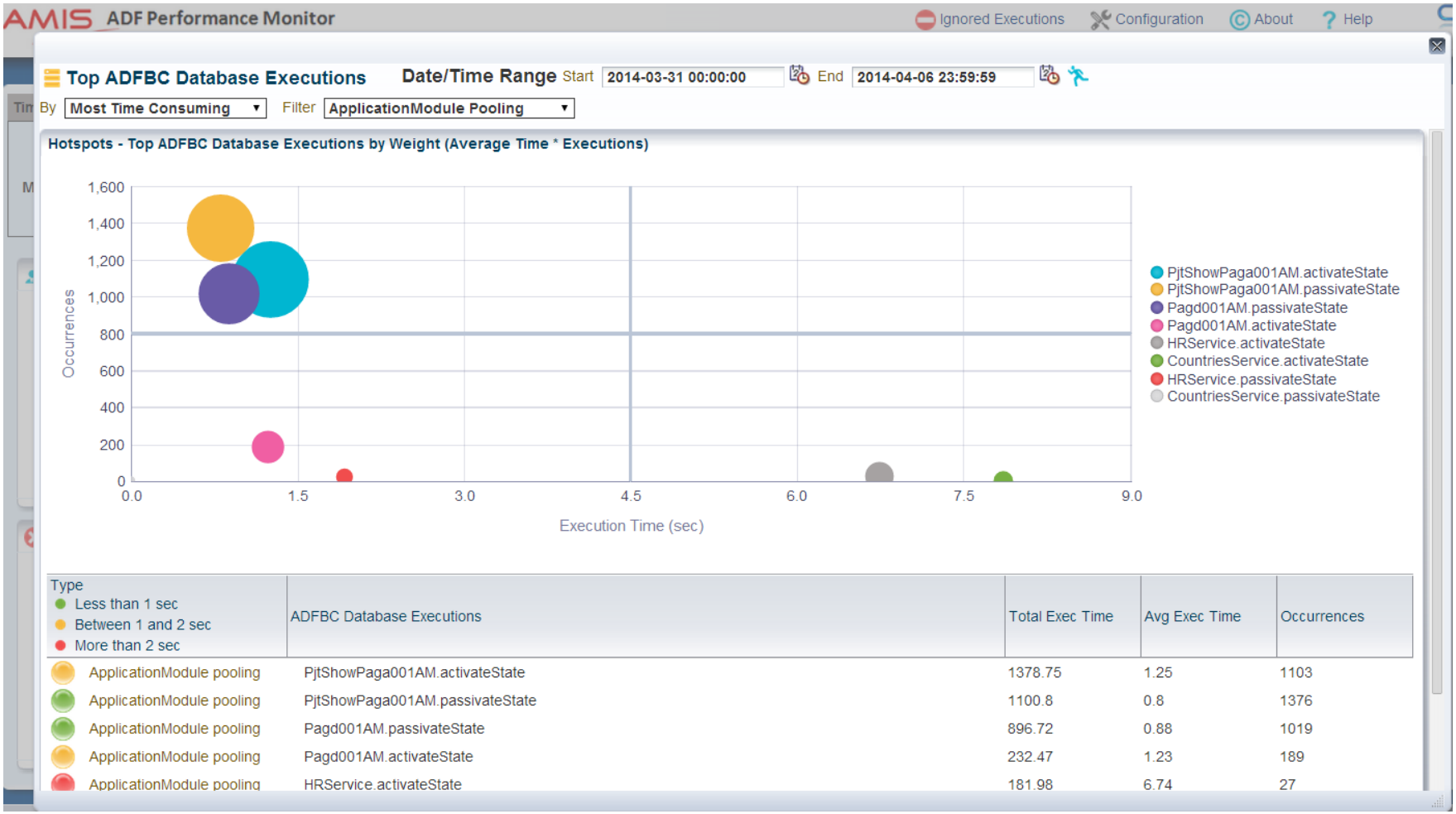


Example: Worst ApplicationModule activations & passivations overview

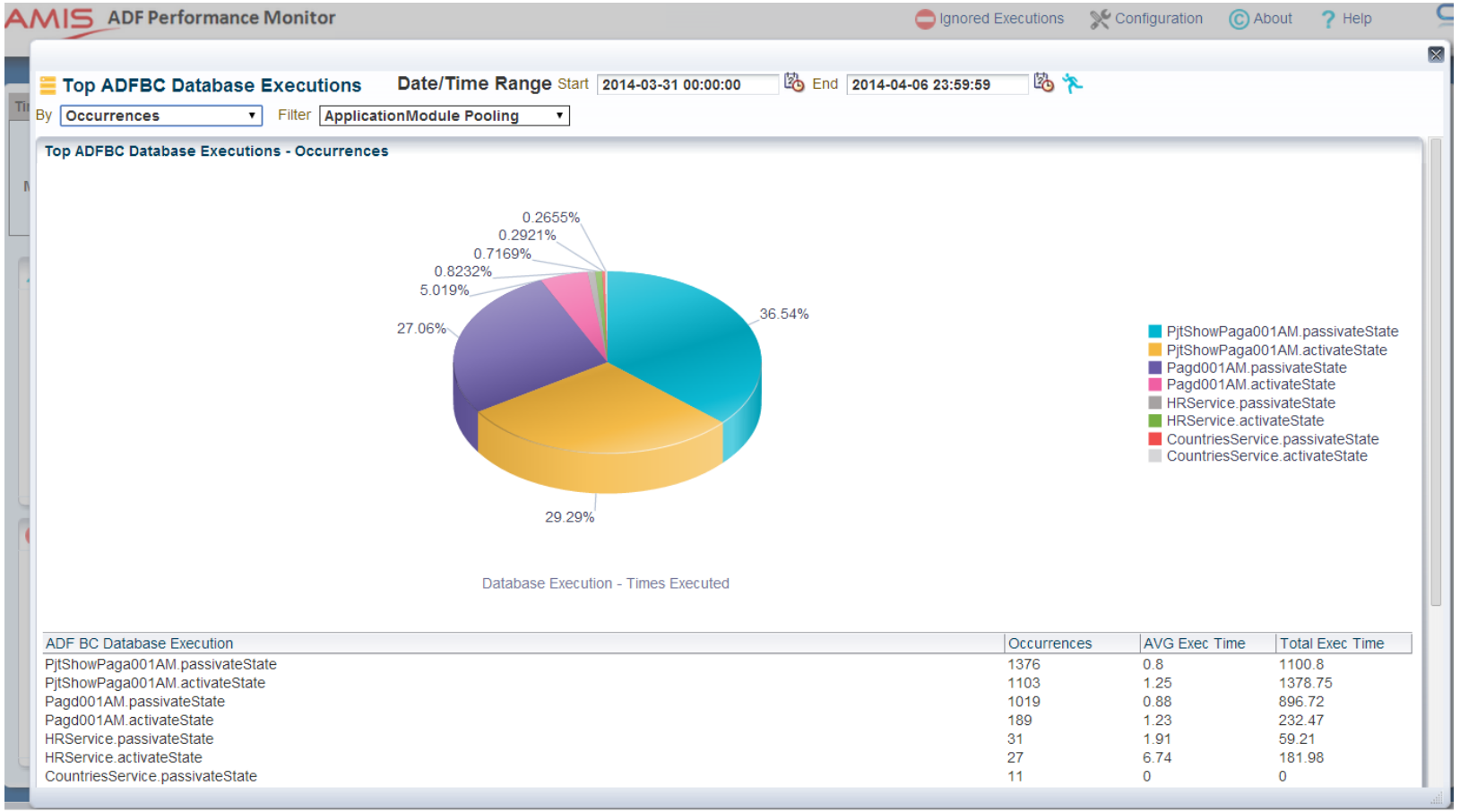
- Monitor aggregates of ApplicationModule activations and passivations
  - AVG, Occurrences, Total Exec Time, Min, Max
  - Zoom in into single executions



Example: Worst ApplicationModule activations & passivations overview (2)



Example: Occurrences overview



ADF BC Database Execution	Occurrences	AVG Exec Time	Total Exec Time
PjtShowPaga001AM.passivateState	1376	0.8	1100.8
PjtShowPaga001AM.activateState	1103	1.25	1378.75
Pagd001AM.passivateState	1019	0.88	896.72
Pagd001AM.activateState	189	1.23	232.47
HRService.passivateState	31	1.91	59.21
HRService.activateState	27	6.74	181.98
CountriesService.passivateState	11	0	0

Learn More

- Instrumenting your ADF Application – part 1
- Instrumenting your ADF Application – part 2
- Tips on How to Limit ADF BC Memory Consumption

- ADF Performance Monitor
  - Whitepaper
  - Information