Improve Your ADF Fusion Application's Response Time by as Much as 70 Percent
Frank Houweling

- Oracle ADF and Java specialist with AMIS (The Netherlands)
- Focus on performance diagnosis and performance management
- Lead architect behind the ADF Performance Monitor, an advanced monitor that shows slow requests, errors, JVM heap & garbage collection, and the layer (Database, Model, e.g.) that causes performance problems

Email: fjhouweling@hotmail.com
Customers do not accept slow applications anymore
Agenda:
Things are happening….

- Too Slowly
- Too Often
- Too Soon
- Too Big
- Too Little
Things are happening....

Too Slowly
Many bottlenecks are simply caused by slow ViewObject queries, PL-SQL calls or EntityObject DML operations

Often hard to track which queries are slow

Some queries are only slow for certain bind parameter values

Some queries are not slow during development but are slow in production environment (that has much more data)
Quick and simple way to log executed database queries

- Override `executeQueryForCollection()` method in the project base `ViewObject` class

```java
public class SummitViewObjectImpl extends ViewObjectImpl {

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

    protected void executeQueryForCollection(Object object, Object[] object1, int i) {
        long startTime = System.nanoTime();
        super.executeQueryForCollection(object, object1, i);
        long totalTime = (System.nanoTime() - startTime) / 1000000;
        logger.fine("VO DATABASE EXECUTION VO usagename: " + getUsename() + " millis: " + totalTime);
    }
}
```
Other ways to detect slow database executions (1)

- Oracle ADF Logger diagnostic tool in JDeveloper

![](ADF_data.png)

- Disadvantage: (too) much overhead for use in test- or production environment
Other ways to detect slow database executions (2)

• Set up a database trace
  - Queries from database perspective
  - Disadvantage: you don’t see the executions from ADF application’s perspective – it is often not easy to relate a database trace to ADF executions
  - Performance overhead
Too Often
Too many (mini) queries (1)

1 Query

Another 2 queries

Another 4 queries
Too often executed ViewObject mini-queries (2)

- Applies to a lot of hierarchical structures
  
  Master - Detail (… Detail, … Detail)

  - Default implementation of `<af:treeTable>` and `<af:tree>` with associations and viewlinks
  - Custom ADFBC overrides and programmatically executed detail ViewObject iterators in getter methods
  - ViewAccessor’s queries that are used for lookup items in `<af:table>`, `<af:treeTable>` and `<af:tree>` components
Solution: Single bulk retrieve replacing multiple queries

Diagram:
- Page
  - Page(Def)
    - ViewObject
      - Database
  - Managed bean
    - Page(Def)
      - ViewObject
      - Database
Fetch size is set too low (fetchsize=1)
Too many database roundtrips (2)

- The ViewObject fetch mode and fetch size properties control how many rows will be returned in each round-trip to and from the database.
- Set the fetchsize to $n+1$ where $n$ is the number of rows to be displayed in the user interface.
Fetch size is set correctly (fetchsize=rows needed +1)

Too Many Database roundtrips (3)
Unintentionally left iterators in PageDefs

Page (UI)

PageDef (Bindings)

Database

<executables>
<iterator Binds="CountriesView" RangeSize="-1"
DataControl="HRServiceDataControl" id="CountriesViewIterator"
Refresh="ifNeeded"/></iterator>
Too many HTTP Requests

- The iterator binding rangesize property represents the current set of objects to be displayed on the page.
- Rule of thumb: for `<af:tables>`, `<af:treetable>` and `<af:tree>` components set the `rangesize` to the max number of records that you can display in the browser, not more (usually not more than 50).

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfui/uimodel"
    version="11.1.1.59.23" id="SearchEmployeesPageDef"
    ControllerClass="hr.demo.ui.controller.TabRegionController"
    Package="hr.demo.ui.view.pageDefs">
<parameters/>
<executables>
    <variableIterator id="variables"/>
    <iterator Binds="EmployeesView1" RangeSize="35"
        DataControl="HRSeriveDataControl" id="EmployeesView1Iterator"
        Refresh="deferred"/>
    </iterator>
</executables>
</pageDefinition>
```
Too frequent ApplicationModule passivation & activation

- AM pooling enables multiple users to share several application module instances
- It involves saving and retrieving session state data from the database or file. This mechanism is provided to make the application scalable and becomes very important under high load with many concurrent users
- Default values can be very inefficient and may cause many unneeded passivations and activations
- Carefully read the documentation in the ADF Fusion developers Guide (Ch. 44 of the 11gR2)
Understanding Passivation

Application Module pool

AM 1  AM 2  AM 3  AM 4  AM 5

Passivation

Database/File
State Information Saved During Passivation

**Transactional State**
- New, modified, deleted EntityObjects

**Non-Transactional State**
- For each active ViewObject
  - Current row (key)
  - New rows
  - ViewCriteria
  - executed flag
  - Range start and Range size
  - Access mode
  - Fetch mode and fetch size
  - ViewObject custom data

*AM user session snapshot*

```xml
<AM MonVcl="0">
  <cd/>
  <TXN Def="1" New="0" Lox="2" tsi="0" pclid="28"/>
  <CONN/>
  <VO>
  <VO sig="1383492980695" qf="0" ut="0" ds="1" cs="27" It="1" St="25" St="0"
     in="1" Ex="1" Def="hr.demo.model.queries.DepartmentsView">
    <Name="DepartmentsView">
    <RSQ>
      <![CDATA[SELECT Departments.DEPARTMENT_ID,
               Departments.DEPARTMENT_NAME,
               Departments.MANAGER_ID,
               Departments.LOCATION_ID,
               Departments.REPORT
                FROM DEPARTMENTS Departments]]>
    </RSQ>
    </VO>
  </VO>
</AM>
```
Understanding Activation

Application Module pool

AM 1 | AM 2 | AM 3 | AM 4 | AM 5

Database/File

Activation
• Number of ApplicationModule instances to create when the pool is initialized

Init Pool Size

jbo.ampool.initpoolsize (default: 0)

• Tip: a high value avoids AM instantiation time when load increases – the hit is taken at server startup
Maximum Pool Size

- Maximum number of application module instances that the pool can allocate

**Application Module pool**

```
AM 1  AM 2  AM 3  AM 4  AM 5  AM 6  AM 7  AM 8  ...  AM
```

**Maximum Pool Size**

`jbo.ampool.maxpoolsiz` (default: 4096)
• Number of AMs in the pool that attempt to preserve session affinity for the next request

Tip: maintaining "session affinity" improves performance – bump up this value (and avoids expensive passivation and activation)
Pool Polling Interval

- Length of time in Millis between pool cleanups

Application Module Pool

AM 1  AM 2  AM 3  AM 4  AM 5  AM 6  AM 7  AM 8  ...  AM

Pool Polling Interval

jbo.ampool.monitorsleepinterval (default: 10 Minutes)
Max Available Size

- Number of instances that survive pool cleanup

Application Module pool

- AM 1
- AM 2
- AM 3
- AM 4
- AM 5
- AM 6
- AM 7
- AM 8
- …
- AM

Max Available Size
jbo.ampool.maxavailablesize (default: 25)

- Tip: a higher value makes more AMs available and improves performance
Idle Instance Timeout

- Millis after which to **mark an inactive AM** for removal during next pool cleanup

Application Module pool

AM 1  AM 2  AM 3  AM 4  AM 5  AM 6  AM 7

Idle Instance Timeout
jbo.ampool.maxinactivage (default: 10 Minutes)

- Tip: increase this value to make more AMs available – this will improve performance
Maximum Instance Time to Live

- Millis that an application module instance lives in the pool

Tip: set this value to -1 to make more AMs available – this will improve performance
Recommendations

• First determine in your application how many AMs on average a user session uses. Calculate how many AMs you will need during peak times and set the **maxavailablesize** and **recyclethreshold** to this value (number of sessions with short think times * average needed AMs a session)

• Set **minavailablesize** and **initpoolsize** to 80% of the needed AMs during peak times

  – jbo.ampool.maxavailablesize = jbo.recyclethreshold
  – jbo.ampool.minavailablesize = jbo.ampool.initpoolsize = 80 % of jbo.ampool.maxavailablesize
  – jbo.ampool.timetolive = -1
  – increase **jbo.ampool.maxinactiveage**
  – jbo.ampool.doampooling=true (default)
  – jbo.doconnectionpooling=false (default)

Result:

• Avoids AM instantiation time when load increases - the hit is taken at server startup
• Avoids ‘expensive’ passivations and activations of AMs under normal load
More Application Module pooling guidelines

- Discover AM pooling problems in development- and test- and not production environment
- Develop and test with AM pooling disabled! AMs will always be passivated and activated
Do not passivate state **All Transient values** at ViewObject level

- If checked SQL calculated and transient values of all ViewRows (!) will be passivated - and activated when the session state is reloaded - this may lead to long running AM passivations and activations.
More ApplicationModule pooling guidelines

- Uncheck **Including All Transient values**
More ApplicationModule pooling guidelines

- Only if absolutely necessary (test it with AM pooling disabled), passivate at the **attribute level** of your ViewObject
- Very often you don’t need to passivate it
Too many full HTTP Requests

- Make use of the powerful AJAX capabilities
- Use partial page requests instead of full page requests
- Set *where possible* on all buttons, links, menu-items `partialSubmit="true"`

```xml
<af:commandButton partialSubmit="true" text="Submit" id="cbl"/>
```

- Applies to
  - `<af:commandLink>`
  - `<af:commandImageLink>`
  - `<af:commandButton>`
  - `<af:commandMenuitem>`
  - `<af:commandNavigationItem>`
  - `<af:commandToolbarButton>`
Things are happening....

Too Big
Too much data in ADFBC memory

- Try to avoid loading more database rows and columns than you need
Case: Dutch ministry of Justice

- Huge JVM memory usage, long running garbage collections (>40 sec)
- Root cause:
  - Application data retrieved from the database into memory was not properly limited
  - Many rows (>25,000) with too many attributes in memory
  - Also rows and their attributes were retained in session for an unnecessary period of time
What causes it

- ViewObject’s **accessmode** is default *Scrollable* (VO tuning section)
- Scrolling down an af:table retrieves and loads all rows from the database (!)
Pattern: Table-Form layout

setCurrentRowWithKey

Screen
- Number
- Name

Same ViewObject usage
- Number
- Name
- Job
- Street
- ZipCode
- Attribute N
Rows and their attributes retrieved

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Job</th>
<th>Street</th>
<th>ZipCode</th>
<th>Attribute N</th>
</tr>
</thead>
<tbody>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Rows and their attributes needed

<table>
<thead>
<tr>
<th>No</th>
<th>Name</th>
<th>Job</th>
<th>Street</th>
<th>ZipCode</th>
<th>Attribute N</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ViewObject
Solution

• Reduce No. Columns retrieved
  – Dedicated ViewObjects for table and form
  – After selecting a row in table: query form VO with its ID as bind parameter

• Reduce No. Rows Retrieved
  – Set appropriate \textbf{maximum} fetchsize
  – Range Paging for table VO
Limit the impact of non-selective queries that may return thousands of rows

Guidelines:
• Table-layout: ± 250 rows
• Form-layout: 1 row
• Create-layout: 0 rows

Alternatively set globally
• META-INF\adf-config.xml

Advantages
• Low memory consumption
• Fast execution
ViewObject Range Paging

- “I want to see page 9 of the search results (and 10 per page)”
- Range Size of $R$ Rows per Page
- See Page $P$ of the Results

```
SELECT * FROM (  
  SELECT /*+ FIRST_ROWS */ IQ.*, ROWNUM AS Z_R_N FROM (  
    <VIEWOBJECTSQL QUERY>  
  ) IQ WHERE ROWNUM < :0)  
WHERE Z_R_N > :1
```

$P \times R + 1 = $ Last Row In Page

$(P - 1) \times R = $ First Row In Page

- Keeps only the current range (or "page") of rows in memory!
ViewObject Range Paging

- If you have to ‘display’ > 500 records in a table
### Locations (47104)

<table>
<thead>
<tr>
<th>LocationId</th>
<th>StreetAddress</th>
<th>PostalCode</th>
<th>City</th>
<th>StateProvince</th>
<th>CountryId</th>
<th>CountryName</th>
<th>Report</th>
</tr>
</thead>
<tbody>
<tr>
<td>2662500</td>
<td>2017 Shinjuku-ku</td>
<td>1689</td>
<td>Tokyo</td>
<td>Tokyo Prefecture</td>
<td>JP</td>
<td>Japan</td>
<td>Report</td>
</tr>
<tr>
<td>2662600</td>
<td>9450 Kamiya-cho</td>
<td>6823</td>
<td>Hiroshima</td>
<td></td>
<td>JP</td>
<td>Japan</td>
<td>Report</td>
</tr>
<tr>
<td>2662700</td>
<td>2014 Jabberwocky...</td>
<td>26192</td>
<td>Southlake</td>
<td>Texas</td>
<td>US</td>
<td>United States of America</td>
<td>Report</td>
</tr>
<tr>
<td>2662800</td>
<td>2011 Interiors Blvd</td>
<td>99236</td>
<td>South San Francisco</td>
<td>California</td>
<td>US</td>
<td>United States of America</td>
<td>Report</td>
</tr>
<tr>
<td>2662900</td>
<td>2007 Zagora St</td>
<td>50090</td>
<td>South Brunswick</td>
<td>New Jersey</td>
<td>US</td>
<td>United States of America</td>
<td>Report</td>
</tr>
<tr>
<td>2663100</td>
<td>147 Spadina Ave</td>
<td>M5V 2L7</td>
<td>Toronto</td>
<td>Ontario</td>
<td>CA</td>
<td>Canada</td>
<td>Report</td>
</tr>
<tr>
<td>2663200</td>
<td>6092 Boxwood St</td>
<td>YSW 9T2</td>
<td>Whitehorse</td>
<td>Yukon</td>
<td>CA</td>
<td>Canada</td>
<td>Report</td>
</tr>
<tr>
<td>2663300</td>
<td>40-5-12 Laogiang...</td>
<td>190518</td>
<td>Beijing</td>
<td></td>
<td>CN</td>
<td>China</td>
<td>Report</td>
</tr>
<tr>
<td>2663400</td>
<td>147 Spadina Ave</td>
<td>55530</td>
<td>Toronto</td>
<td>Ontario</td>
<td>CA</td>
<td>Canada</td>
<td>Report</td>
</tr>
<tr>
<td>2663500</td>
<td>6092 Boxwood St</td>
<td>YSW 9T2</td>
<td>Whitehorse</td>
<td>Yukon</td>
<td>CA</td>
<td>Canada</td>
<td>Report</td>
</tr>
<tr>
<td>2663600</td>
<td>40-5-12 Laogiang...</td>
<td>190518</td>
<td>Beijing</td>
<td></td>
<td>CN</td>
<td>China</td>
<td>Report</td>
</tr>
<tr>
<td>2663700</td>
<td>1298 Villeparle (E)</td>
<td>490231</td>
<td>Bombay</td>
<td>Maharashtra</td>
<td>IN</td>
<td>India</td>
<td>Report</td>
</tr>
<tr>
<td>2663800</td>
<td>12-98 Victoria Street</td>
<td>2901</td>
<td>Sydney</td>
<td>New South Wales</td>
<td>AU</td>
<td>Australia</td>
<td>Report</td>
</tr>
<tr>
<td>2663900</td>
<td>198 Clementi North</td>
<td>540198</td>
<td>Singapore</td>
<td></td>
<td>SG</td>
<td>Singapore</td>
<td>Report</td>
</tr>
<tr>
<td>2664000</td>
<td>8204 Arthur St</td>
<td></td>
<td>London</td>
<td></td>
<td>UK</td>
<td>United Kingdom</td>
<td>Report</td>
</tr>
<tr>
<td>2664100</td>
<td>Magdalen Centre,</td>
<td>OX9 92B</td>
<td>Oxford</td>
<td>Oxford</td>
<td>UK</td>
<td>United Kingdom</td>
<td>Report</td>
</tr>
<tr>
<td>2664200</td>
<td>9702 Chester Road</td>
<td>09629850293</td>
<td>Stretford</td>
<td>Manchester</td>
<td>UK</td>
<td>United Kingdom</td>
<td>Report</td>
</tr>
<tr>
<td>2664300</td>
<td>Schwanthalerstr...</td>
<td>80925</td>
<td>Munich</td>
<td>Bavaria</td>
<td>DE</td>
<td>Germany</td>
<td>Report</td>
</tr>
<tr>
<td>2664400</td>
<td>Rua Frei Caneca 1...</td>
<td>01307-002</td>
<td>Sao Paulo</td>
<td>Sao Paulo</td>
<td>BR</td>
<td>Brazil</td>
<td>Report</td>
</tr>
<tr>
<td>2664500</td>
<td>20 Rue des Corps...</td>
<td>1730</td>
<td>Geneva</td>
<td>Geneva</td>
<td>CH</td>
<td>Switzerland</td>
<td>Report</td>
</tr>
<tr>
<td>2664600</td>
<td>Murtenstrasse 921</td>
<td>3095</td>
<td>Bern</td>
<td>BE</td>
<td>CH</td>
<td>Switzerland</td>
<td>Report</td>
</tr>
<tr>
<td>2664700</td>
<td>Pieter Breughelstr.</td>
<td>3029SK</td>
<td>Utrecht</td>
<td>Utrecht</td>
<td>NL</td>
<td>Netherlands</td>
<td>Report</td>
</tr>
<tr>
<td>2664800</td>
<td>Mariano Escobedo</td>
<td>11932</td>
<td>Mexico City</td>
<td>Distrito Federal,</td>
<td>MX</td>
<td>Mexico</td>
<td>Report</td>
</tr>
<tr>
<td>2664900</td>
<td>9702 Chester Road</td>
<td>09629850293</td>
<td>Stretford</td>
<td>Manchester</td>
<td>UK</td>
<td>United Kingdom</td>
<td>Report</td>
</tr>
<tr>
<td>2665000</td>
<td>Schwanthalerstr...</td>
<td>80925</td>
<td>Munich</td>
<td>Bavaria</td>
<td>DE</td>
<td>Germany</td>
<td>Report</td>
</tr>
<tr>
<td>2665100</td>
<td>Rua Frei Caneca 1...</td>
<td>01307-002</td>
<td>Sao Paulo</td>
<td>Sao Paulo</td>
<td>BR</td>
<td>Brazil</td>
<td>Report</td>
</tr>
<tr>
<td>2665200</td>
<td>20 Rue des Corps...</td>
<td>1730</td>
<td>Geneva</td>
<td>Geneva</td>
<td>CH</td>
<td>Switzerland</td>
<td>Report</td>
</tr>
</tbody>
</table>
Too big scope for managed beans

- Use as small memory scopes as possible
Make IDs of the following ADF faces **container components** as small as possible (max 2 characters):

- `<af:pageTemplate>`
- `<af:region>`
- `<af:panelCollection>`
- `<af:table>`
- `<af:treetable>`
- `<af:tree>`
- `<af:iterator>`
Too much HTML to the browser (2)

- Monitor HTTP traffic in browser (for example firebug)
- Look for big files
**Too much HTML to the browser (3)**

- Make the container component IDs as small as possible, for example:
  - `<af:pageTemplate id="t"/>
  - `<af:panelCollection id="c"
  - `<af:treeTable id="utt">
  - `<af:region id="r4">`

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>html_long_ids</td>
<td>351 KB</td>
<td>XML File</td>
</tr>
<tr>
<td>html_small_ids</td>
<td>262 KB</td>
<td>XML File</td>
</tr>
</tbody>
</table>
Too much logging on

- Switch to **SEVERE** at the WLS / EM level
Things are happening….

Too Soon
Executed too soon (1)

- Example: `<af:panelTabbed>` component with in each `<af: showDetailItem>` an `<af:region>` that starts a taskflow execution
Executed too soon (2)

.jsff page:

```
<af:panelTabbed id="pt2">
  <af:showDetailItem text="Eenheden" id="sdi1" flex="0">
    <af:region value="${bindings.EhdChangePagb003TF1.regionModel}" id="r1" />
  </af:showDetailItem>
  <af:showDetailItem text="Projectfases" id="sdi2" flex="0">
    <af:region value="${bindings.PfeChangePagb004TF1.regionModel}" id="r2" />
  </af:showDetailItem>
  <af:showDetailItem text="Objecttypen" id="sdi3">
    <af:region value="${bindings.0jeChangePagb005TF1.regionModel}" id="r3" showDisclosure="false" />
  </af:showDetailItem>
  <af:showDetailItem text="Budgetfases" id="sdi4" />
</af:panelTabbed>
```

PageDefinition:

```xml
<afm:panelTabbed id="pt2">
  <afm:showDetailItem text="Eenheden" id="sdi1" flex="0">
    <afm:region value="${bindings.EhdChangePagb003TF1.regionModel}" id="r1" />
  </afm:showDetailItem>
  <afm:showDetailItem text="Projectfases" id="sdi2" flex="0">
    <afm:region value="${bindings.PfeChangePagb004TF1.regionModel}" id="r2" />
  </afm:showDetailItem>
  <afm:showDetailItem text="Objecttypen" id="sdi3">
    <afm:region value="${bindings.0jeChangePagb005TF1.regionModel}" id="r3" showDisclosure="false" />
  </afm:showDetailItem>
  <afm:showDetailItem text="Budgetfases" id="sdi4" />
</afm:panelTabbed>
```
• Use `childCreation="lazyUncached"` or `childCreation="lazy"` to defer taskflow execution of all tabs until the tab is opened

```xml
<af:panelTabbed id="pt2" childCreation="lazyUncached">
    <af:showDetailItem text="Eenheden" id="si1" flex="0"
        stretchChildren="first">
        <af:region value="#{bindings.EhdChangePag003TF1.regionModel}"
            id="r1" />
    </af:showDetailItem>
</af:panelTabbed>

<af:showDetailItem text="Projectfases" id="si2" flex="0"
    stretchChildren="first">
```

• For popups, use `childCreation="deferred"`

```xml
<af:popup id="pl" childCreation="deferred"/>
```

• For tasklows in regions, use `activation="conditional"` and a RefreshCondition on the taskflow executable

```xml
<taskFlow id="BgtShowPag001TF1"
    taskFlowId="/bgtshow/BgtShowPag001TF.xml#BgtShowPag001TF"
    activation="conditional" RefreshCondition="#{myBean.bgtVisible}"
    xmlns="http://xmlns.oracle.com/adf/controller/binding"
    Refresh="ifNeeded"/>
```
• Defer the runtime instantiation of ViewObject and nested ApplicationModule instances until the time they are used.
Important to consider for the following components:

- `<af:table>`
- `<af:treeTable>`
- `<af:tree>`
- `<af:popup>`
- `<af:menu>`
- `<dvt:%graph%>`

- For a better user experience, lazy delivery should be used for tables, or other stamped components, which are known to have a slow fetch time (when they show many records, or the query is slow)
Things are happening….
Use a shared application module to group view instances when you want to reuse lists of static data across the application.
Too little JVM Heap size

Best practice:

- Set (-Xms and -Xmx) as large as possible within available physical memory

- Generational parallel garbage collection strategy is recommended to maximize throughput: -Xgc:genpar (JRockit)
Load tests:

- Should be done in time (not 1 week before production)
- Are very useful to test scalability and SLA
- All load testing tools take time to become familiar

- **Apache JMeter**
  - not easy to configure
  - but free

- **Oracle Application Testing Suite (OATS)**
Design your pages smart

• Do not display too much data on a page, keep your page design simple if possible
• Do not unnecessarily query data that is not immediately needed (unopened tree nodes, inactive tabs, invisible popups, unopened dropdown lists, e.g.)
ADF BC
• Detect & tune ViewObject slow queries
• Setting the appropriate tuning-values on the ViewObject
• Implement the Table-Form pattern using 2 separate view objects
• Use ViewObject *Range paging* if table rows > 250
• Use lazy loading on ApplicationModules
• Sizing the Application module pool
ADF Model
• Set efficient PageDef Iterator rangesizes

ADF View
• Use partialSubmit=true where possible on all link, button and menu components

System
• Set your JVM Heap size appropriately and choose an effective garbage collection strategy

Design
• Smart design - do not retrieve data that is not immediately needed