CloudaIDE
web framework for database developers
CloudaIDE – web framework for database developers

- Database and User Interface – common denominators of web applications
- The idea of CloudaIDE – use the database development methods to design a web application
Goals

- easy to learn
- fast to develop
- ergonomics – mouseless data entry
- easy migration of Oracle Forms applications
- easy deployment
- scalability
- extensibility
- mobile apps

Reasons (among many others)

Removal of the Java plug-in (applets) support from browsers
Three-tier architecture

- **User Interface**
  - Layout – Screen Editor
  - User and Server Interaction – “triggers”. Programs in MT a simple language resembling PSQL

- **Application server**
  - Security tasks. Checking validity of client requests
  - Translating request/response between client and server
  - Application logic – gateway between MT and Java (in case of not sufficient power of database Stored Procedures)

- **Database**
  - Data Storage
  - Application logic implemented using Stored Procedures
Components of the CloudaIDE

- CloudaIDE Designer – an Eclipse plug-in
- CloudaIDE applications:
  - Administration Console
  - User self-service
- CloudaIDE database supporting objects
Fast-Track Development

To develop a screen a programmer can drag a database view or table from the Data Source Explorer and drop onto the Screen Editor.
Fast-Track Development

- Creates one of two selectable layouts – Table or Form
- Creates screen items based on columns and gives them properties
- Binds screen items with database columns
- Provides the programmer with default CRUD and QBE
Screen Editor -
Tool to sketch of screen layout

Features:
- Drag and Drop
- Undo Redo
- Cut and Paste
- Drop from Data Source Explorer
Properties View displays properties of a selected object

- Block is UI representation of a database table
- Item is a client object that corresponds to a database column
Different tables for DML and Query can be specified in order to facilitate updates on unupdatable views.

Instead of Insert, Instead of Update, Instead of Delete

Programmer can define own procedures to handle DML. Updates can be replaced by application specific processing. Similar to database triggers. They give extra flexibility. For example – Instead of Delete can in fact mark a row as deleted without deleting a database row. The “Instead” procedures can also disallow performing DML operations or add extra database processing to UI events.
Properties View - Item

Item is a client object that corresponds to a database column

- **Master Item** – each item can have a master item, an Item in the master block. This allows to express arbitrarily complex master-detail relationships between blocks
REGISTER HEADER

SALES ORDER HEADER

INSTALLMENTS

ORDER DETAILS
Properties View – Item

- **Query Only** – if set to yes then the item belongs only to Query table (not to DML table). Because of this Query Only Item does not take part in DML operations.

- **Returnable** – Similar to SQL return column. After any DML operation this item is returned to the client.

- **Tab Index** – programmer can statically arrange any sequence of cursor navigation. The programmer can dynamically set next navigation item and also force cursor navigation using:
  ```
  set_item_property(next_item,'BLOCK_NAME.ITEM_NAME');
  ```

- **LOV call** Programmer can specify the name of a Form with parameters. This turns Textbox into a List Item.
Tool Palette serves to create/select screen elements

It has two layout elements:

- **Grid** – to place other elements in HTML table
- **Table** – to place other elements in a table of horizontal rows (spreadsheet like)
Outline View

Tree structure of the form. Using it the programmer can see all the data elements of the form. Blocks, items and code

Through Outline View the programmer has also access to non UI elements:

- **Off-Screen items** – items that are never displayed
- **Triggers** – pieces of code reacting to client events
- **Procedures**
MT Triggers Language

- Isolates the programmer from the complexity of asynchronous nature of screen interaction and AJAX calls
- No callbacks
- Close to PL/SQL. Key differences:
  - no SQL
  - case sensitive
  - datatypes
### MT Triggers Language

<table>
<thead>
<tr>
<th>Data Types of MT</th>
<th>Main constructs</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>Procedures</td>
</tr>
<tr>
<td>number</td>
<td>Triggers</td>
</tr>
<tr>
<td>date (timestamp)</td>
<td>Loops</td>
</tr>
<tr>
<td>oidn – encrypted number</td>
<td>Conditionals</td>
</tr>
<tr>
<td>etext – encrypted text</td>
<td>Exceptions</td>
</tr>
</tbody>
</table>

**Data Types of MT**
- text
- number
- date (timestamp)
- oidn – encrypted number
- etext – encrypted text
Trigger Types

Depending on the declaration level

**Form**
- PRE_FORM, NEW_FORM

**Block**
- VALIDATE_RECORD, NEW_RECORD_INSTANCE,
  PRE_DELETE, PRE_INSERT, PRE_UPDATE,
  POST_DELETE, POST_INSERT, POST_UPDATE

**Item**
- VALIDATE_ITEM,
  KEY_DUPLICATE_RECORD,
  BUTTON_PRESSED, KEY_NEXT_ITEM,
  KEY_LIST_VALUES,
  KEY_DUPLICATE_RECORD,
  KEY_ITEM,
Example trigger

```plaintext
-- NEW_FORM
begin

  set_title('CloudaIDE user manager');
  if url_parameter('p_user') is not null then
    e_mail_verification;
    return;
  end if;
  set_block_property('THANKS', visible, false);
  if url_parameter('email_user') is not null then
    set_block_property('USER', visible, false);
    set_block_property('PASSWORD_EMAIL', visible, false);
    go_item('PASSWORD_CHANGE.PASSWORD');
    return;
  end if;
  set_block_property('PASSWORD_CHANGE', visible, false);
exception
  when others then
    message_error(error_message);
end;
```
procedure before_delete(p_id oidn) is
begin
    if p_id is null then
        return;
    end if;
    mr_dictionary_p.before_delete_test(p_id);
    confirm('Do you want to remove ?', true);
    if not success then
        return;
    end if;
    mr_dictionary_p.prune_bushes(p_id);
exception
    when others then
        message_error(error_message);
end;
What the client-side code is?

- Single page architecture - everything is downloaded as a single page
- Javascript:
  - Creates HTML
  - Reacts for events
  - Executes AJAX to communicate with the server
- Downloaded once for a compilation
- Downloaded incrementally – minimizing initial download
What the client-side code is?

- This way screen layout and behaviour is supplied to the client only once.
- The main network traffic after initial download is data.
- The layout and behaviour code is cached on the client (until next release)
The application is built of forms
Following this, code also is split into forms
Code delivery to the client is split into engine (common functionality) initial download and form (on demand) downloads. This helps to minimize network traffic because no monolithic initial download is carried out and unused code never gets to a client
Code is loaded in gzipped form (most browsers handle this)
### Demo Application code splitting report

#### Compile report: Permutation 0

<table>
<thead>
<tr>
<th></th>
<th>Full code size</th>
<th>Initial download size</th>
<th>Left over code</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>432 258 Bytes</td>
<td>300 184 Bytes</td>
<td>52 366 Bytes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Split Points</th>
<th>Location</th>
<th>Size (Bytes)</th>
<th>% of total</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 @pl.mroovka.ap.client.AAAMain$1::run()</td>
<td></td>
<td>26 891</td>
<td>6,2%</td>
</tr>
<tr>
<td>2 @pl.mroovka.ap.client.__LoginForm$1::run()</td>
<td></td>
<td>8 021</td>
<td>1,9%</td>
</tr>
<tr>
<td>3 @pl.mroovka.ap.client.Administration$1::run()</td>
<td></td>
<td>3 522</td>
<td>0,8%</td>
</tr>
<tr>
<td>4 @pl.mroovka.ap.client.Orders$1::run()</td>
<td></td>
<td>18 827</td>
<td>4,4%</td>
</tr>
<tr>
<td>5 @pl.mroovka.ap.client.Sales$1::run()</td>
<td></td>
<td>4 563</td>
<td>1,1%</td>
</tr>
<tr>
<td>6 @pl.mroovka.ap.client.Customers$1::run()</td>
<td></td>
<td>4 565</td>
<td>1,1%</td>
</tr>
<tr>
<td>7 @pl.mroovka.ap.client.Products$1::run()</td>
<td></td>
<td>5 215</td>
<td>1,2%</td>
</tr>
<tr>
<td>8 @pl.mroovka.ap.client.EmployeeList$1::run()</td>
<td></td>
<td>885</td>
<td>0,2%</td>
</tr>
<tr>
<td>9 @pl.mroovka.ap.client.Offices$1::run()</td>
<td></td>
<td>3 405</td>
<td>0,8%</td>
</tr>
<tr>
<td>10 @pl.mroovka.ap.client.Employees$1::run()</td>
<td></td>
<td>3 814</td>
<td>0,9%</td>
</tr>
</tbody>
</table>
Report integration

- CloudaIDE is integrated with BIRT – Business Intelligence and Reporting Tool
- CloudaIDE can call a BIRT report using:
  - Parameters
  - SQL where phrase of a query last executed on a selected database block
Report integration

In the last case a programmer can use the clause in a query in the report.

The execution of the report is protected by a checksum and can be carried out only once for a call.

```
-- BUTTON_PRESSED
begin
  post;
  if not success then
    return;
  end if;
  if :print.kind = 'D' then
    if :orders.id is null then
      message_error('Please select order to print');
      return;
    else
      run_report(mv_order, :orders.id);
    end if;
  else
    run_report(mv_order, null) criteria_block orders;
  end if;
exception
  when others then
    message_error(error_message);
end;
```
Report integration - sequence of events

1. The client asks the server for a URL.
2. The server prepares the URL and writes its sequence number and MD5 hash to the database.
3. The client receives the URL and sends it to the report server.
4. Reports server calculates the MD5 hash of the URL. If the URL is not - HTML error 404 is reported.
5. Reports server deletes the URL's hash in order to disable recurring reports call (from the browser).
6. Reports server executes the report.
Lists of Values
Lists of Values

Realized as forms

- Items of called form bound to Calling form items by a naming convention
- Automatic (in the background) selection of a list element (item List validable property)
- Possibility to enter missing list elements on the fly
- Using the list to drill/navigate data – after the selection is made
Menus

- Come in two flavours:
  - Vertical
  - Horizontal

- Application – wide choice
Menus

- Forms create stack
- Each Form can contribute options to the menu
- These options stay in the menu, until the form is closed
- If an option belonging to a particular form is clicked, all forms above the form are closed, with one exception
- An option resulting in opening a form stores reference to the opened form
- When clicked again the menu system closes all forms above the called form and displays the form opened by this option (not the one that has opened it). It gives an effect similar to a breadcrumb navigation
Built-in Security features

- Built-in authentication:
  - Table
  - Database Account
  - CAS
- Authorizations – definable by the administrator
- Protection against injection
- Object protection
Protection against injection
Protection against injection

- The application server knows application metadata. No direct SQL statements, phrases and procedure calls are passed to the server. Everyone of those are handled indirectly against application metadata.
Object protection

CloudaIDE uses encrypted numbers and texts. The encrypted data contains information about the source of this data. The system analyses the graph of possible assignments. The data of oidn and etext types is sent to a client in encrypted form. No other processing than assignments of this data is possible. When the server receives this data back from the client it checks whether it conforms to the assignment graph. Whether the target entity can be reached by the source. If not, then security exception is reported, otherwise the data after decryption goes to further processing.
Locks

- System uses connection pooling. Because of this every server call (default block DML, or a Stored Procedure call) constitutes a separate transaction

- Locking within the Updates are of two kinds:
  - Last in wins (no locking)
  - Optimistic locking
More information on:

cloudaide.org
QUESTIONS