Tutorial Activiti and SQL
Architecture of the System

process-related information, e.g.
• customers,
• suppliers,
• items,
• orders,
• …
maintained by you

engine-related information, e.g.
• users,
• roles,
• active process instances,
• …
maintained by Activiti

MySQL server

Database `acme`

Database `activiti`

Process Model

Activiti Engine
In this Tutorial

- **First Example**
- Writing to the DB
  - where SQL queries can be annotated
  - what can be used
- Reading from the DB
  - populating forms with database contents
- Triggers based on Database contents
- Interactions between processes
First Example – Form Properties

- Open Eclipse
- Open project example_bpmn_model1
- Open diagram src/main/resources/diagrams/CreateQuote.bpmn
- Select task **Add Quote**
- Open the **Properties** view (see Tutorial 01 Activiti Basics)
- Select **Form**
- The properties show a number of form properties
- Activiti renders these properties as form fields when the task is created (to be executed by a user)
First Example – Execution Listeners

- open **Listeners** in the **Properties** view
- there is a listener on event **complete**
- the listener is of **Type** expression
- the **Implementation** is an expression, preceded by the keyword **sql**: 
- the entire query following after **sql**: gets executed when the task gets completed

(= when the user clicks the “Complete” button of this task in the UI)
First Example – SQL Expression in Listener

- entire query of **Add Quote**

```sql
INSERT INTO `quotes` (`customer`, `item`, `itemprice`, `quantity`, `totalprice`, `handledBy`) VALUES
('${customer}','${item}',${itemprice/100},${quantity},${itemprice/100 * quantity},${execution.processInstanceId})
```

- what it does
  - inserts a new row into table `quotes`
  - where values for columns (`customer`, `item`, `itemprice`, `quantity`, `totalprice`, `handledBy`) are specified
  - the values in the query are set from variables, e.g., `${customer}`, or calculated from expressions, e.g., `${itemprice/100 * quantity}`
  - values of variables have been set by the user (see **Form Properties** of this task)
variables can also be evaluated on arcs
in CreateQuote.bpmn, select the arc between the two XOR-gateways
open **Main config**
the arc expression

```
${addAnother == 'true'}
```

defines that this arc is only taken when the variable addAnother has been set to true
First Example – Try Yourself

- Log into Activiti and **start** the process **Add Quote**
- Fill in the **Customer** field in the start form of the **Add Quote** process

- Go to the **Tasks** tab in Activiti
- There will be an **Add Quote** task assigned to you
- Fill in **Items**, **Price**, and **Quantity** and choose whether another item shall be added.
- Repeat until you are done (**Add another item to this quote** is unchecked)

- Log into MySQL and execute
  
  ```sql
  SELECT * FROM `quotes``;
  ```

- The items that you’ve entered will show up in the query result
In this Tutorial

- First Example
- **Writing to the DB**
  - where SQL queries can be annotated
  - what can be used
- Reading from the DB
  - populating forms with database contents
- Triggers based on Database contents
- Interactions between processes

```
MySQL server
```
```
Database `acme`
```
```
Database `activiti`
```
```
Activiti Engine
```
```
Process Model
```
```
SQL Queries in the process model
```
```
do not touch
```
Where can SQL queries can annotated

- INSERT, UPDATE, DELETE queries can be annotated
  - as an Expression of a task listeners of any task, preferably a listener of type complete
    see also http://www.activiti.org/userguide/#taskListeners

- SELECT, INSERT, UPDATE, DELETE queries can be annotated
  - in a service task that has type Expression
  - the result of the query can be stored in a variable

- … anywhere, where Activiti allows an expression
  http://www.activiti.org/userguide/#apiExpressions

- an SQL expression must be preceded by the keyword sql:
What can be used inside SQL queries

- each SQL query in an Activiti expression has to be a valid SQL query
- values in an SQL query can be
  - constants
  - Activiti expressions using ${...}, i.e., no nesting of SQL queries
- examples:
  - `UPDATE `quotes` SET `state`='checking', `handledBy`=${execution.processInstanceId} WHERE `state`='added';`
  - `UPDATE `quotes` SET `state`='${quoteOK}', `itemprice`=${itemPrice},${quantity}=${quantity} WHERE `id`=${quoteID};`
  - `INSERT INTO `quotes` (`customer`, `item`, `itemprice`, `quantity`, `totalprice`, `handledBy`) VALUES ('${customer}', '${item}', ${itemprice/100},${quantity},${itemprice/100 * quantity}, ${execution.processInstanceId} );`
In this Tutorial

- **First Example**
- **Writing to the DB**
  - where SQL queries can be annotated
  - what can be used
- **Reading from the DB**
  - populating forms with database contents
- **Triggers based on Database contents**
- **Interactions between processes**
Read from the DB

- SELECT queries can be annotated
  - in a Service Task of type Expression → the result of the query can be stored in a variable
  - in Default expressions of a Form of a User Task, as follows…
Populating Forms with DB Contents (1)

- Open Eclipse
- Open project `example_bpmn_model1`
- Open diagram `src/main/resources/diagrams/CheckQuote.bpmn`
- Select task `Check Quote` and open Properties > Form
- The Default column contains expressions with SQL queries
Populating Forms with DB Contents (2)

- for task **Check Quote**, the form property **quoteID**
  - is of type **enum**
  - has as **default expression** a query preceded by the keyword **sql_ui**:
    - query: SELECT `id` WHERE `state`='checking'

- when the user form is shown
  - the query is executed, and
  - the results of the query are shown in a drop-down list
  - from which the user can pick a value
for task **Check Quote**, the form property **customerName**

- is of type **String** (default type) and
- has as **default expression** a query preceded by the keyword `sql_ui`:
  
  ```sql
  SELECT `customer` FROM `quotes` WHERE `id`=&{quoteID}
  ```

- the expression &{quoteID} refers to the current value in the field **quoteID**

- when a new value is set in **quoteID**, the query gets executed and the first returned value is shown in **customerName**
Populating Forms with DB Contents (4)

- A field with Readable=True and Writeable=False will be read-only (values just displayed).
- A field with Writeable=True will be pre-filled with a queried value, the value can be changed by the user.

- Only values of the **first column** of a SELECT query will be used to populate a field.
- For field of type **enum**:
  - **All** values will be put into the drop down list.
- For other types (**String, Long, Double**):
  - Only the **first** value will be put into the field.
default expressions with \texttt{sql\_ui}: queries can refer to values of any number of fields \&\{field1\}, \&\{field2\}, …

the fields referred to in a query have to be defined \textbf{before} the field that uses the query

- there must not be a cycle of references
- use the \textbf{Up} and \textbf{Down} buttons to define a correct order of fields
In this Tutorial

- First Example
- Writing to the DB
  - where SQL queries can be annotated
  - what can be used
- Reading from the DB
  - populating forms with database contents
- Triggers based on Database contents
- Interactions between processes
Trigger by Start Event (1)

- Open Eclipse
- Open project example_bpmn_model1
- Open diagram src/main/resources/diagrams/CheckQuote.bpmn
- select the **start timer event** and open **Properties > Main config**

  - the **Time cycle** field has a value `0 0/1 * * * ?`
  - = **UNIX cron** expression
  - → Activiti creates a new instance of this process **every minute**
To create guarded start timer events:
- select the **start timer event**, go to **Properties > Form**
- create a form property with
  - **Id**=sql_trigger and
  - **Default** expression being an **sql**: query

When the timer fires, the query gets executed
- if the result is empty, then **no** process instance is created
- otherwise, a new instance is created
Example in CheckQuote.bpmn

- the sql_trigger in the start timer event has the expression
  `SELECT * FROM `quotes` WHERE `state`='added';`
- a new instance will be created whenever there is a quote that is in state `added`
- to prevent creation of infinitely many instances, table quotes should be updated so that it does not contain quotes in state `added` anymore
Trigger by Intermediate Event (1)

- Open Eclipse
- Open project example.bpmn_model1
- Open diagram src/main/resources/diagrams/WaitForEntry.bpmn
- Select the Intermediate Catch Event

- The intermediate event pauses the process execution until a signalX is raised (Properties > Main config > Signal ref)

- Signals can be raised in various ways
  see [http://www.activiti.org/userguide/#bpmnEvents](http://www.activiti.org/userguide/#bpmnEvents)
signals have to be defined
- unfortunately, the Activiti Designer has problems showing signal definitions in the graphical editor

to create/edit a signal definition
**right click** on the model file, **Open With > XML Editor**
Trigger by Intermediate Event (3)

- to create/edit a signal definition in the XML Editor, select the Source View

- every signal definition
  - is a child of `<definitions ...>`
  - has the form
    <signal id="idString" name="text or SQL query"/>
  - see http://www.activiti.org/userguide/#bpmnSignalEventDefinition
at any catch event, process execution will halt, and
continue only when a signal of the referred signal id is raised
if signal name contains an sql: query, then
• Activiti regularly executes the query
• when the query returns a result, a signal of the given id is raised, and any halted execution continues
In this Tutorial

- First Example
- Writing to the DB
  - where SQL queries can be annotated
  - what can be used
- Reading from the DB
  - populating forms with database contents
- Triggers based on Database contents
- Interactions between processes
Interactions between processes

- in project example_bpmn_model1
- processes CreateQuote.bpmn and CheckQuote.bpmn are interacting

CreateQuote.bpmn

![CreateQuote BPMN diagram]

Database `acme`

Insert triggered by SELECT

CheckQuote.bpmn

![CheckQuote BPMN diagram]

UPDATE

UPDATE
Good Luck!

21071 DBL Information Systems

Dirk Fahland

TU/e
Technische Universiteit Eindhoven
University of Technology
Where innovation starts