



Dirk Fahland

21071 DBL Information Systems

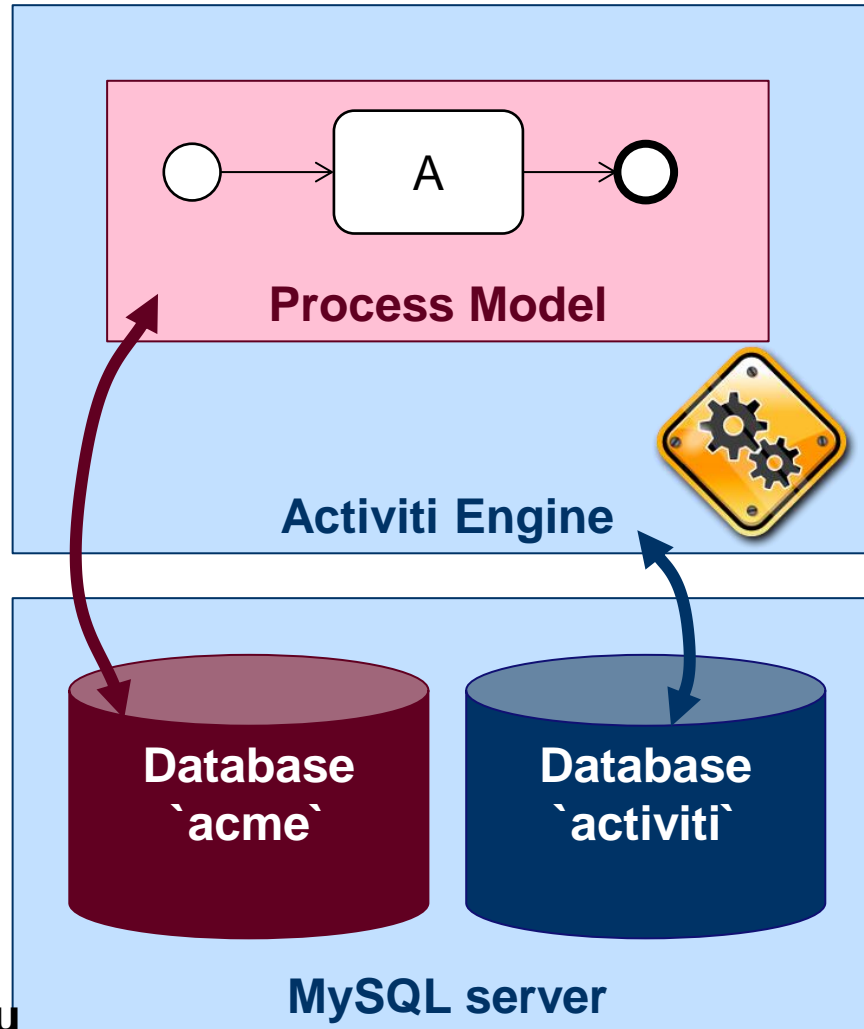
Tutorial **Activiti** and **SQL**

TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts

Architecture of the System



process-related information, e.g.

- customers,
- suppliers,
- items,
- orders,
- ...

maintained by you

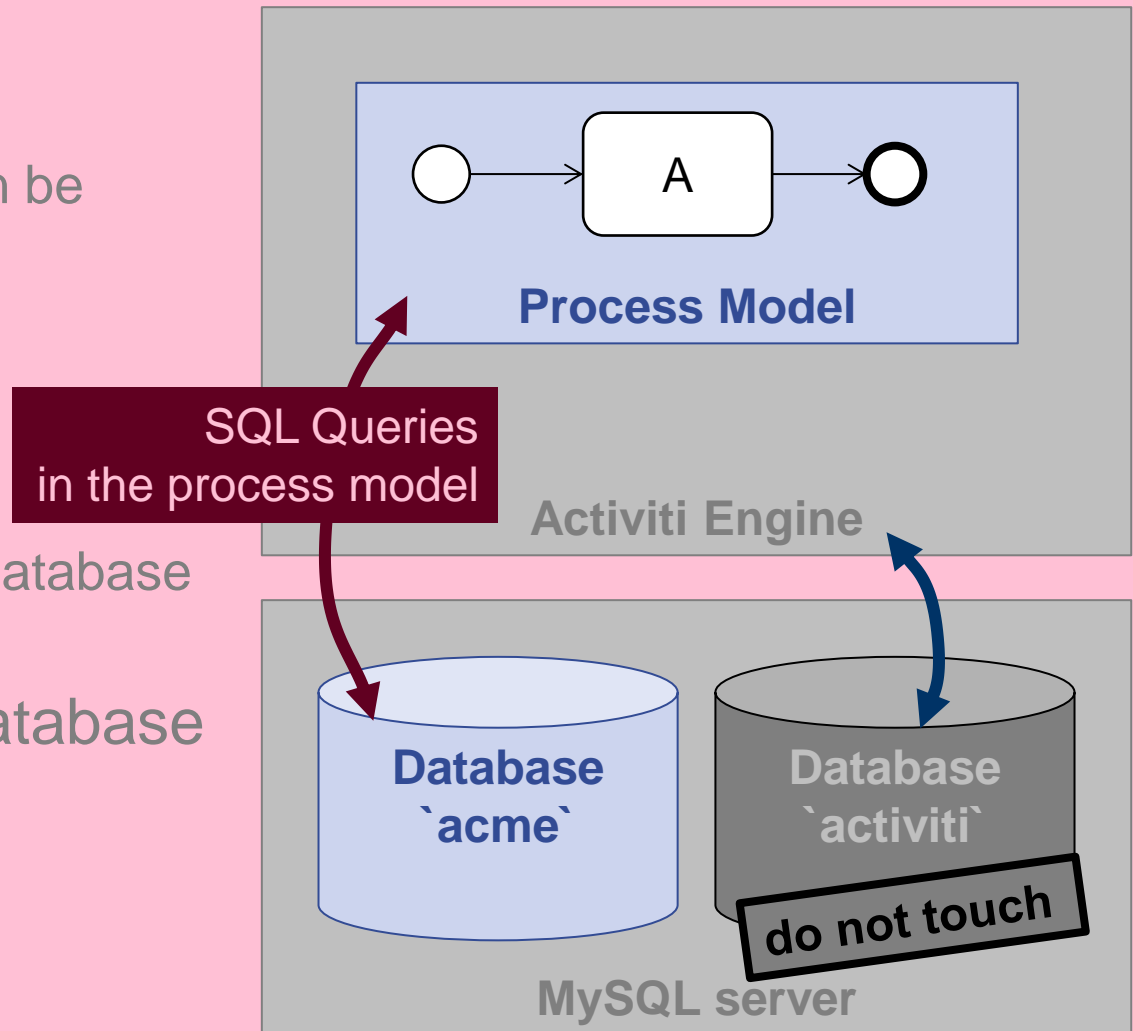
engine-related information, e.g.

- users,
- roles,
- active process instances,
- ...

maintained by Activiti

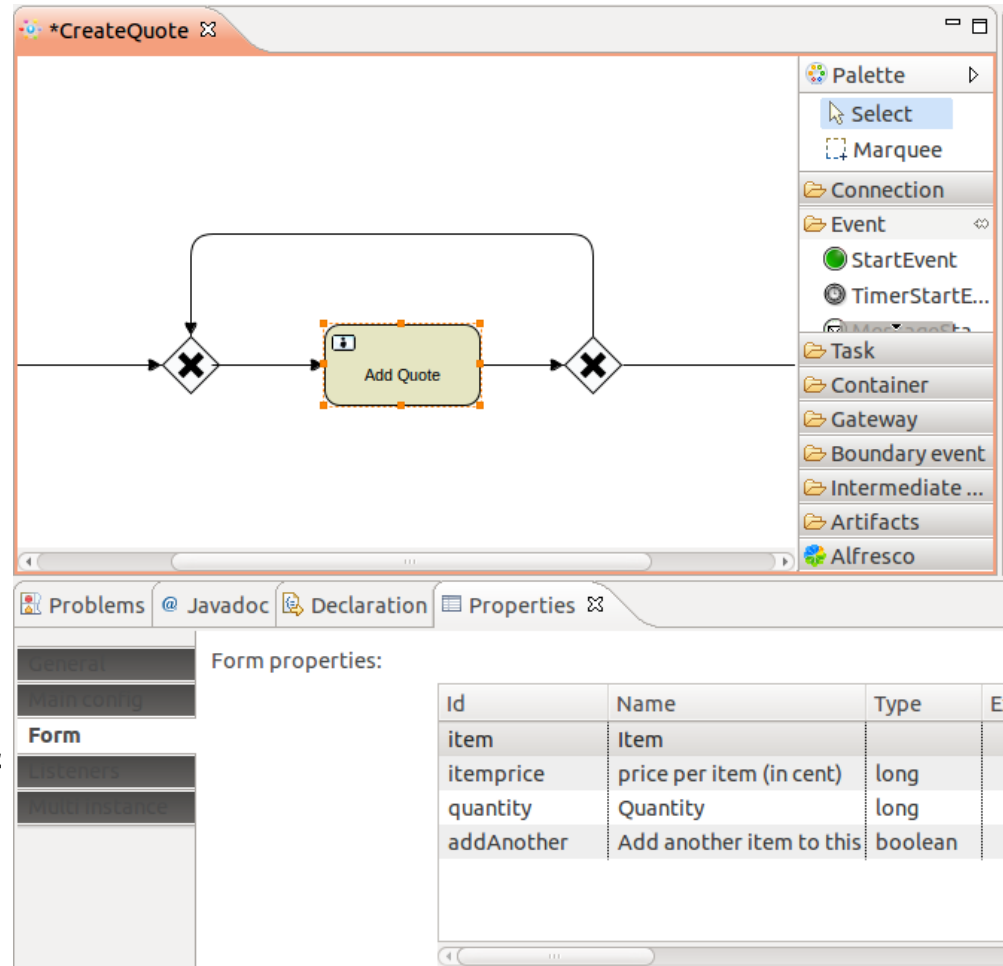
In this Tutorial

- **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_model11`
- 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)



The screenshot shows the Eclipse IDE interface. The main editor displays a BPMN diagram for a task named 'Add Quote'. The task is highlighted with a dashed orange border. The Properties view is open at the bottom, showing the 'Form' tab. The 'Form properties' table is visible, listing various form fields and their types.

Id	Name	Type	E
item	Item		
itemprice	price per item (in cent)	long	
quantity	Quantity	long	
addAnother	Add another item to this	boolean	

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)

The screenshot shows a BPMN diagram with a task named 'Add Quote' between two gateway symbols. Below the diagram is the Properties view, which is open to the 'Listeners' tab. The 'Task listeners:' section contains a table with the following data:

Listener implementation	Type	Event
sql:INSERT INTO `quotes` (`c	expression	complete

First Example – SQL Expression in Listener

- entire query of **Add Quote**

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

- what it does

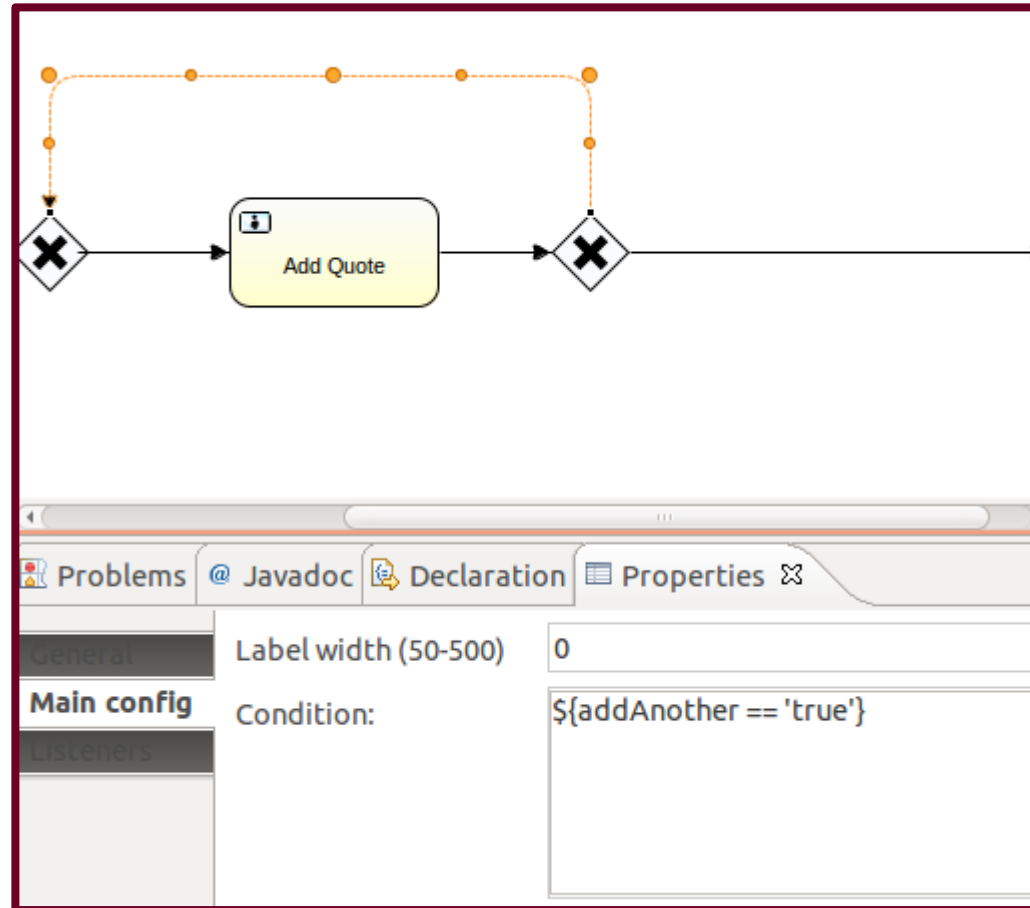
- inserts a new row into table ``quotes``
- where values for columns (``customer``, ``item``, ``itemprice``, ``quantity``, ``totalprice``, ``handledBy``) are specified
- the values in the query are set from variables, e.g., `${customer}`, or calculated from expressions, e.g., `${itemprice/100 * quantity}`
- values of variables have been set by the user (see **Form Properties** of this task)

First Example – Expressions on Gateways

- 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

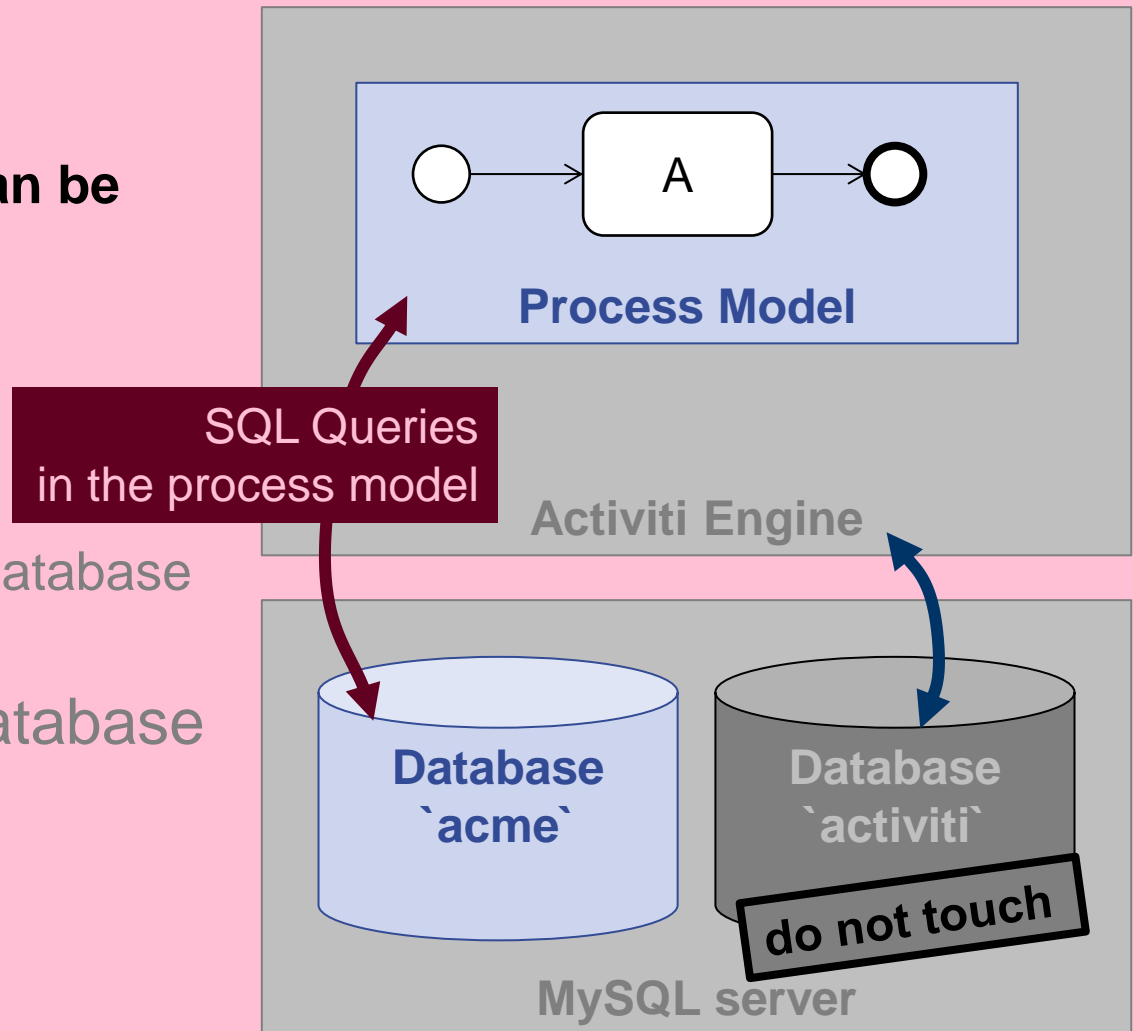
- Log into Activiti and **start** the process **Add Quote**
- Fill in the **Customer** field in the start form of the **Add Quote** process

- 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
`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



Where can SQL queries can annotated

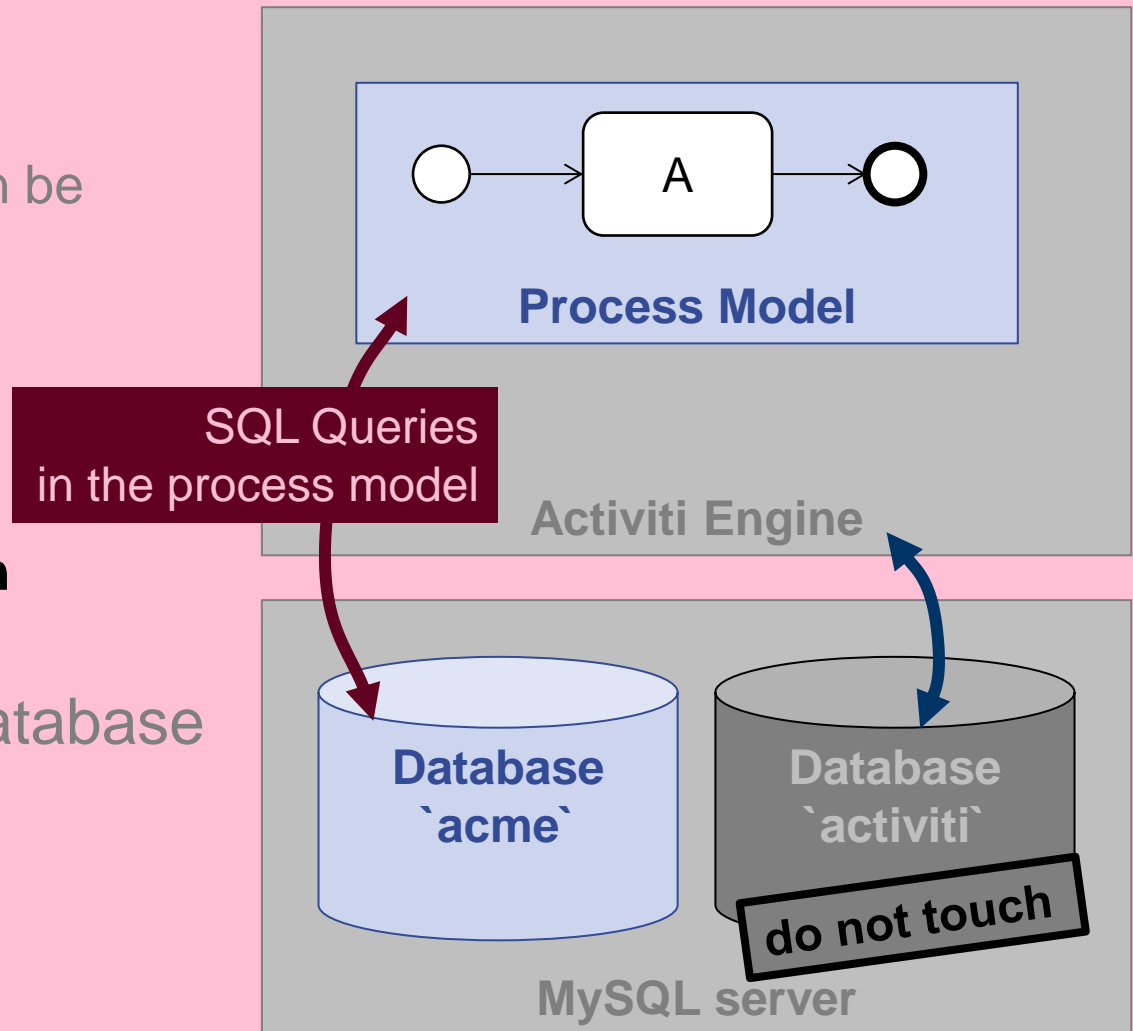
- INSERT, UPDATE, DELETE queries can be annotated
 - as an **Expression** of a **task listeners** of any task, preferably a listener of type **complete**
see also <http://www.activiti.org/userguide/#taskListeners>
- SELECT, INSERT, UPDATE, DELETE queries can be annotated
 - in a **service task** that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



Reading 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

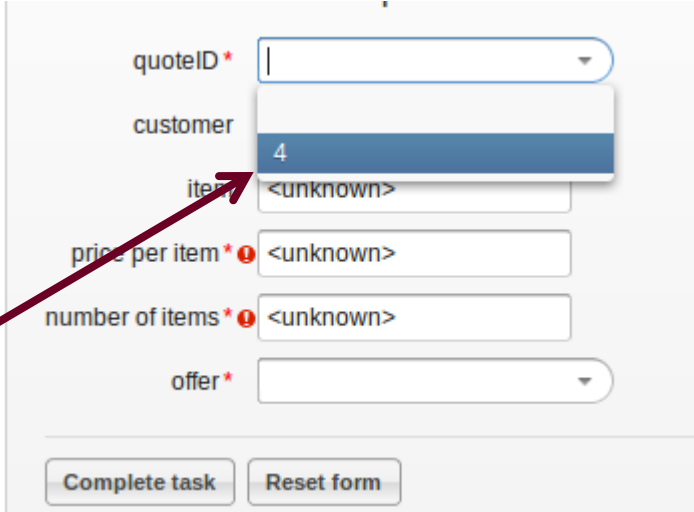
The screenshot shows the Eclipse IDE interface. At the top, a BPMN diagram is displayed with three elements: a Start Event, a Task named 'set quotes to checking', and a Task named 'Check Quote'. The 'Check Quote' task is selected and highlighted with a dashed orange border. To the right of the diagram is a palette with categories: Event (containing StartEvent), Task (containing UserTask), and Container. Below the diagram, the Properties view is open, showing the 'Form' tab. The 'Form properties:' section contains a table with the following data:

Id	Name	Type	Expression	Variable	Default
quoteID	quoteID	enum			sql_ui:SELECT `id`, `cu
customerName	customer				sql_ui:SELECT `custor
item	item				sql_ui:SELECT `item`
itemPrice	price per item	double			sql_ui:SELECT `itempr
quantity	number of ite	long			sql_ui:SELECT `quanti
quoteOK	offer	enum			accepted

A red circle highlights the 'Default' column of the table, which contains SQL query expressions for each form field.

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



The screenshot shows a web form with the following fields and values:

- quoteID***: A dropdown menu with a blue border and a downward arrow.
- customer**: A dropdown menu with a blue border and a downward arrow. The value **4** is selected and highlighted in blue.
- item**: A text input field containing the value **<unknown>**.
- price per item***: A text input field containing the value **<unknown>** and a red error icon.
- number of items***: A text input field containing the value **<unknown>** and a red error icon.
- offer***: A dropdown menu with a blue border and a downward arrow.

At the bottom of the form, there are two buttons: **Complete task** and **Reset form**.

A red arrow points from the text "from which the user can pick a value" in the list above to the dropdown menu for the "customer" field.

Populating Forms with DB Contents (3)

- 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**:
 - `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**

The screenshot shows a web form with the following fields and values:

Field	Value
quoteID *	4
customer	Mr. Smith
item	Hammer
price per item *	40.00
number of items *	3
offer *	

At the bottom of the form are two buttons: "Complete task" and "Reset form".

Populating Forms with DB Contents (4)

- a field with **Readable=True** and **Writable=False** will be **read-only** (values just displayed)
- a field with **Writable=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

Populating Forms with DB Contents (5)

- default expressions with `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 **before** the field that uses the query
 - → there must not be a cycle of references
 - use the **Up** and **Down** buttons to define a correct order of fields

Form properties:

Id	Name	Type	Expression	Variable	Default	Pa
quoteID	quoteID	enum			sql_ui:SELECT `id`,`cu	
customerName	customer				sql_ui:SELECT `custor	
item	item				sql_ui:SELECT `item`	
itemPrice	price per item	double			sql_ui:SELECT `itempr	
quantity	number of ite	long			sql_ui:SELECT `quant	
quoteOK	offer	enum			accepted	

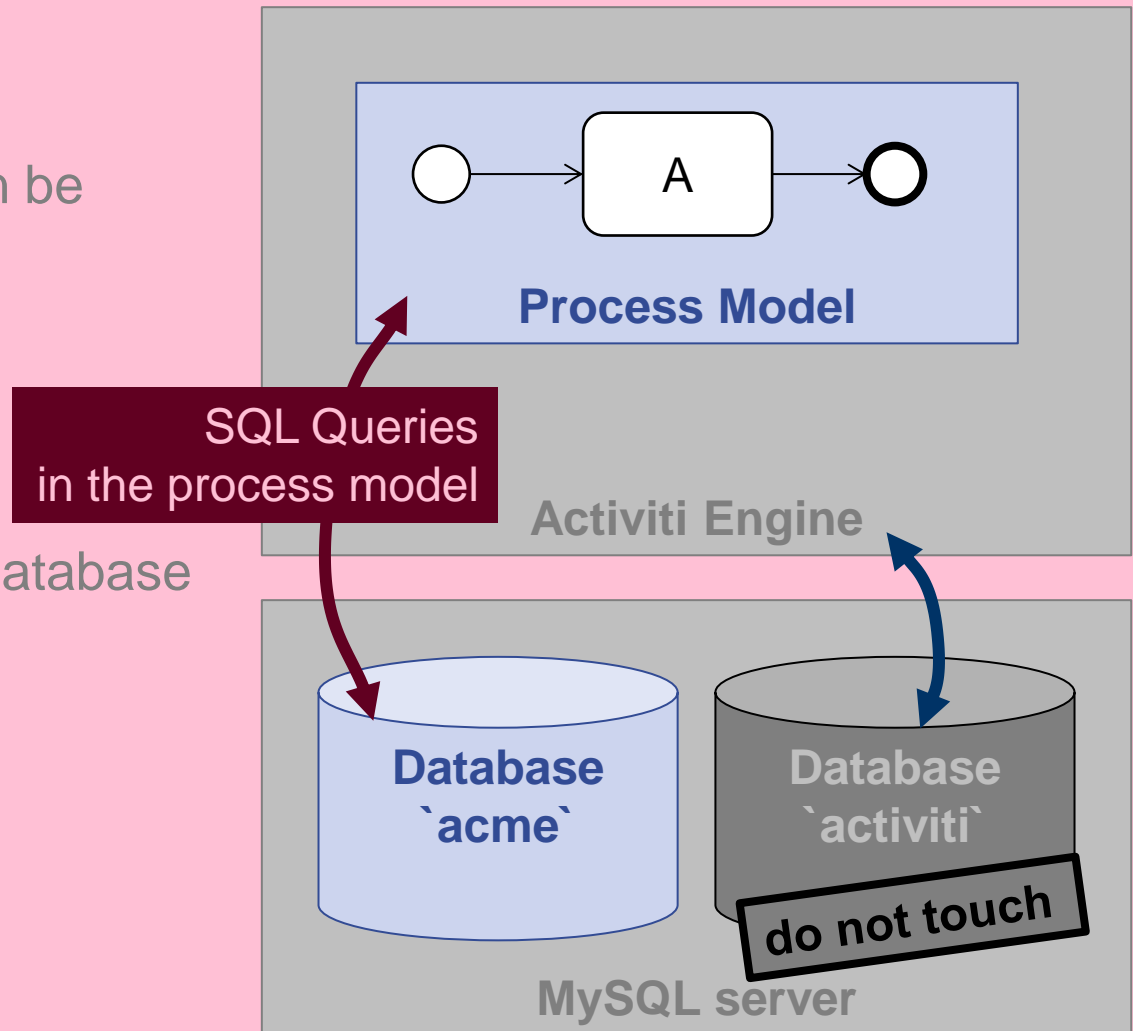
&{quoteID}

Up

Down

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



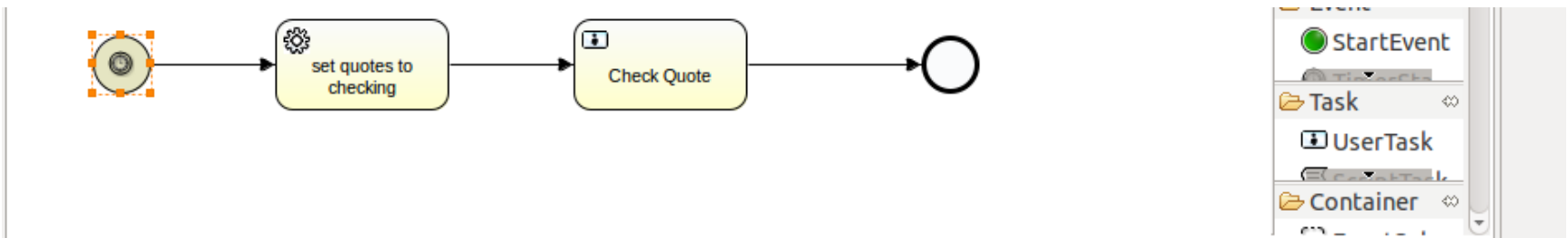
Trigger by Start Event (1)

- 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**

The screenshot displays the Eclipse IDE interface. At the top, a BPMN diagram is visible, featuring a start timer event (a circle with a gear icon) connected to a task labeled 'set quotes to checking', which is followed by another task labeled 'Check Quote', and finally an end event (a circle). Below the diagram, the 'Properties' window is open, showing the 'Main config' tab. The 'Time cycle' field is highlighted with a red oval and contains the value '0 0/1 * * * ?'. Other fields in the 'Main config' tab include 'Time duration' and 'Time date (ISO 8601)'. The 'Properties' window also shows a tree view on the right with categories like 'StartEvent', 'Task', 'UserTask', and 'Container'.

Trigger by Start Event (2)

- 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



The diagram illustrates a BPMN process flow. It starts with a StartEvent (circle with a green dot), followed by a Task (rounded rectangle with a gear icon) labeled 'set quotes to checking'. This is followed by another Task (rounded rectangle with a document icon) labeled 'Check Quote'. The process ends with an EndEvent (circle). To the right, a Properties window is open, showing a list of elements: StartEvent, Task, UserTask, and Container.

Properties window tabs: Problems, @ Javadoc, Declaration, Properties

Form properties:

Id	Name	Type	Ex	Va	Default	Pattern	Re
sql_trigger	SQL Trigger				sql:SELECT * FROM `quotes` WHERE		fa

Trigger by Start Event (3)

■ 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

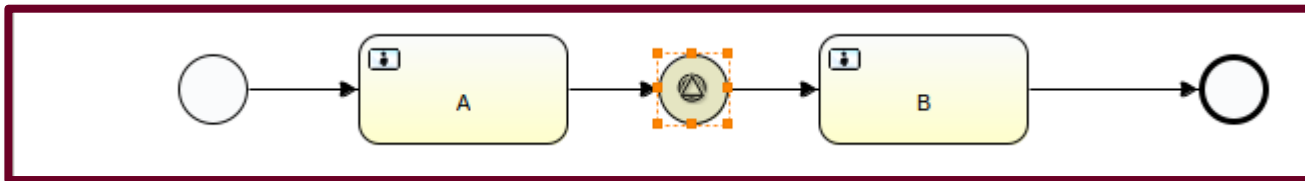
```
graph LR; StartEvent((Start Event)) --> SetQuotes[set quotes to checking]; SetQuotes --> CheckQuote[Check Quote]; CheckQuote --> EndEvent((End Event));
```

The diagram shows a BPMN process flow starting with a Start Event (timer icon), followed by a Task 'set quotes to checking', then a Task 'Check Quote', and finally an End Event (circle). A properties panel on the right shows the event type as 'StartEvent'. Below the diagram, the 'Properties' tab is active, showing 'Form properties:' and a table with the following data:

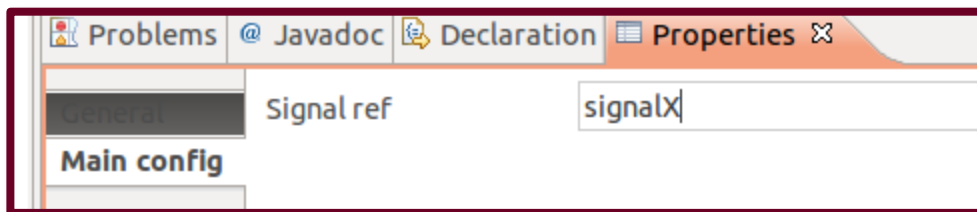
Id	Name	Type	Ex	Va	Default	Pattern	Re
sql_trigger	SQL Trigger				sql:SELECT * FROM `quotes` WHERE		fa

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**

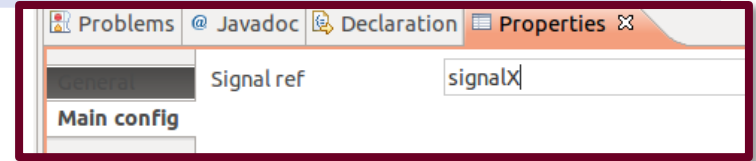
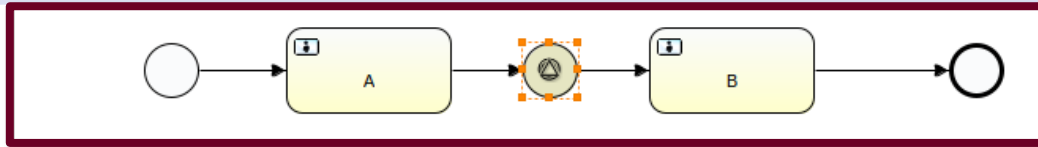


- the intermediate event pauses the process execution until a **signalX** is raised (**Properties > Main config > Signal ref**)

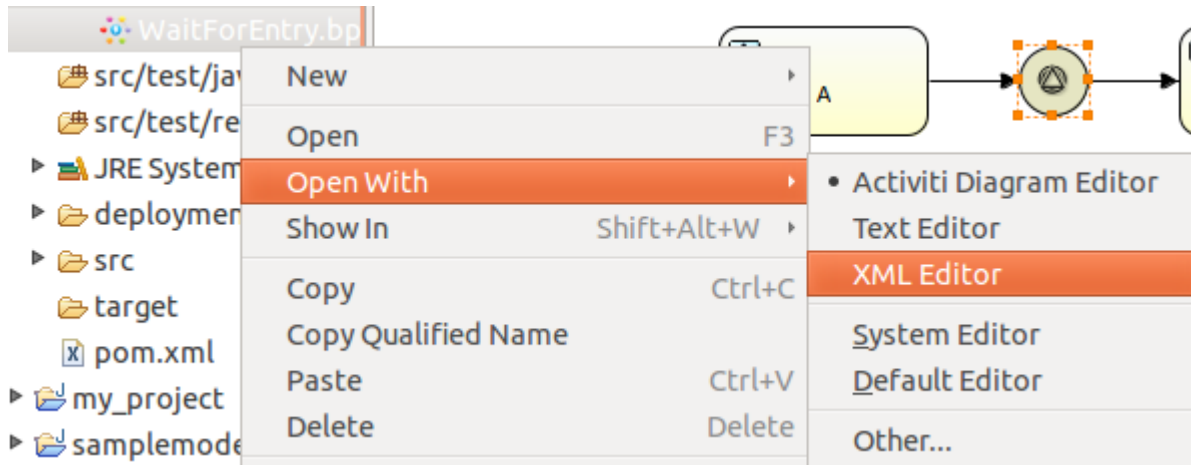


- signals can be raised in various ways
see <http://www.activiti.org/userguide/#bpmnEvents>

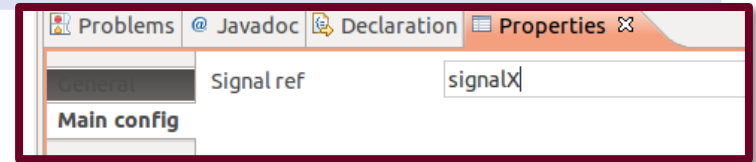
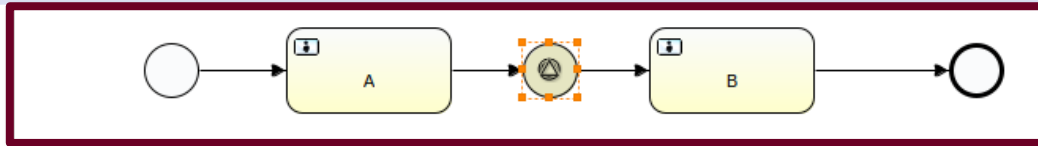
Trigger by Intermediate Event (2)



- 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)

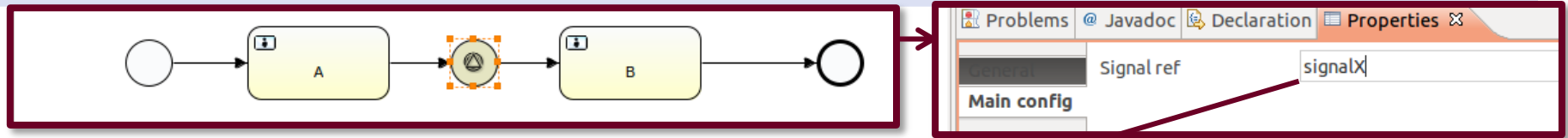


- to create/edit a signal definition in the **XML Editor**, select the **Source View**

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:xsi="http://www.w3.org/...
| <signal id="signalX" name="sql:SELECT * from `quotes` where state = 'accepted';"></signal>
<process id="waitForEntry" name="Wait For Entry" isExecutable="true">
  <startEvent id="startEvent1" name="Start"></startEvent>
```

- 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>

Trigger by Intermediate Event (4)



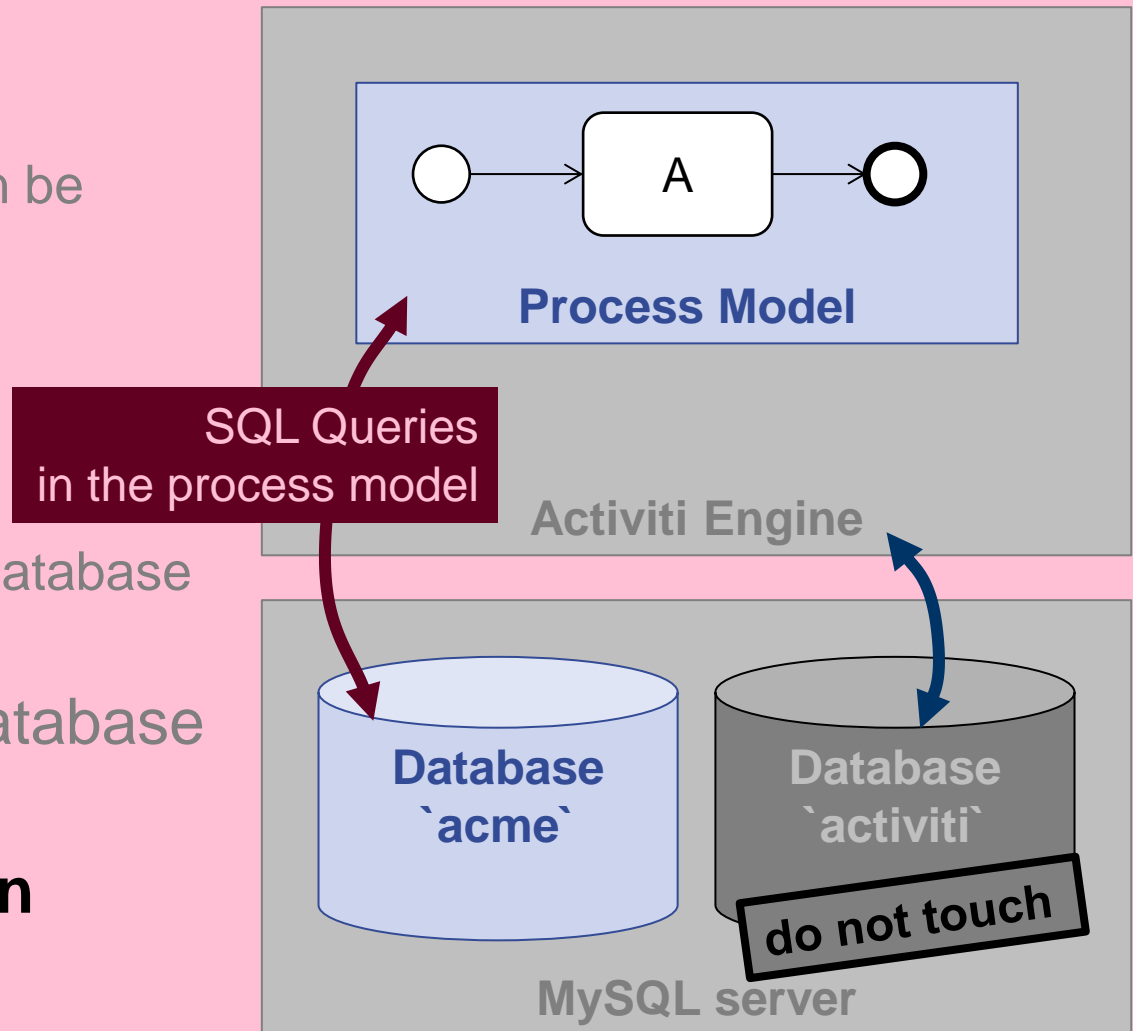
refers to

```
WaitForEntry WaitForEntry.bpmn
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://www.w3.org/2001/XMLSchema-instance http://www.w3.org/2001/XMLSchema-instance">
  <signal id="signalX" name="sql:SELECT * from `quotes` where state = 'accepted';"></signal>
  <process id="waitForEntry" name="Wait For Entry" isExecutable="true">
    <startEvent id="startEvent1" name="Start"></startEvent>
  </process>
</definitions>
```

- 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

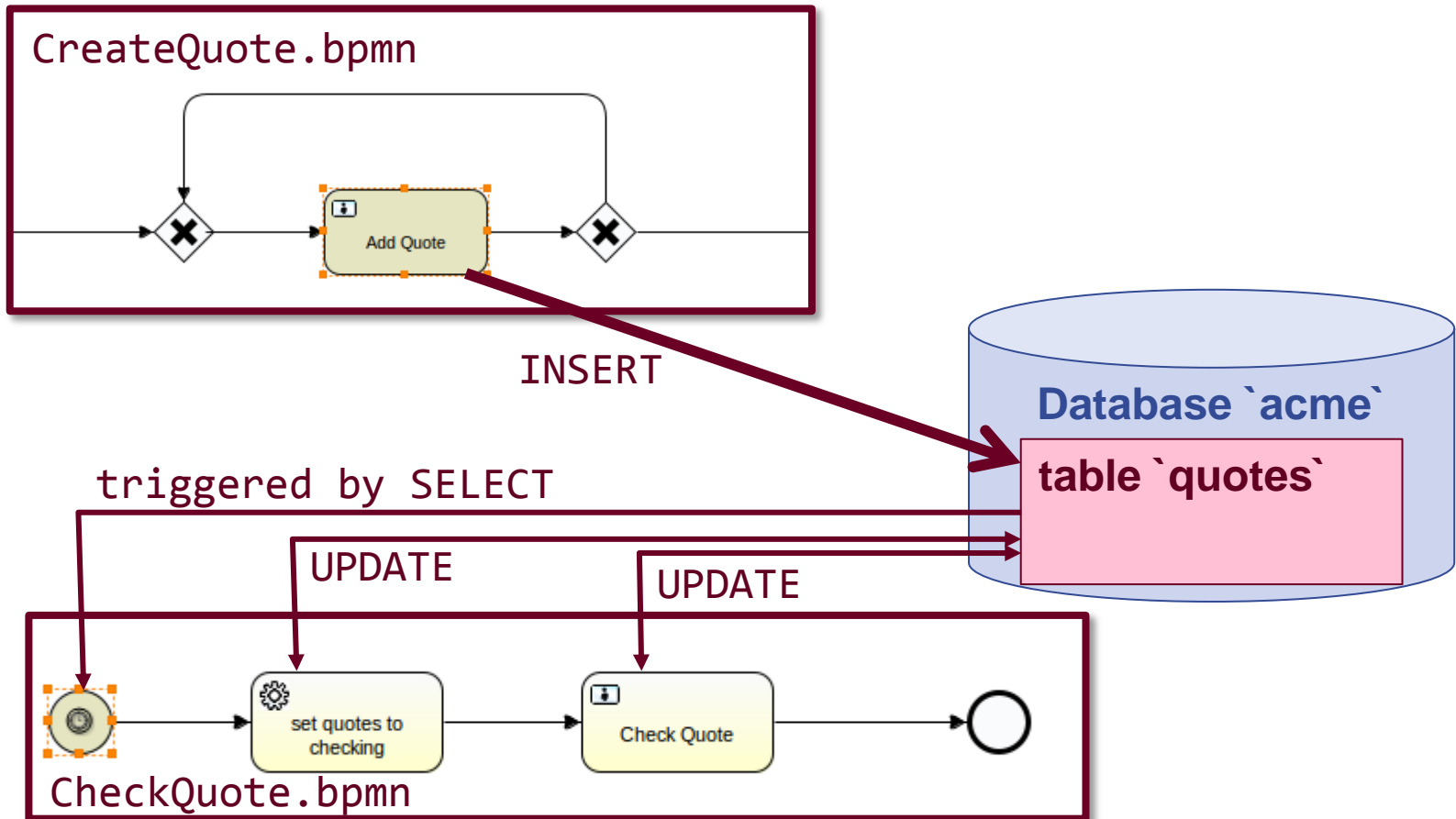
In this Tutorial

- 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





Dirk Fahland

2IO71 DBL Information Systems

Good Luck!

TU / **e**

Technische Universiteit
Eindhoven
University of Technology

Where innovation starts