Course manual

# Rave II

Version 22 of Crew Pairing, Crew Rostering and Tail Assignment

JEPPESEN®
A BOEING COMPANY

Göteborg Apr 2016

# Table of contents

# General

## About this document

This document contains course slides and exercises.

## About Rave II course

This course takes on where the Rave I course ended. Starting with a quick recap of basic Rave elements and then moving on to more advanced Rave features.

By learning more about how Rave works you may write more efficient code that will improve the performance of optimization and Studio even more.

There are some differences between the optimizers and Studio that you need to consider when writing Rave code.

## Rave

Rave is a global modeling language that describes legality, costs and quality constraints. Rave supports all Crew & Fleet products and is essential both for tuning everyday production and performing simulations. Rave is also used as a stand-alone product for in-house applications.

The Rave language allows modeling of complex rules and cost functions that can be checked quickly by optimizers. The performance is essential, since an optimization job may have to do billions of rule and cost evaluations.

Besides providing legality and cost functions for the optimizers, the definitions in Rave are also available for reports and GUI customization.

# *Slides*

# *Exercises*

## Introduction

### Variable names

Variable names are written in font courier: `variable_name`. Use the suggested variable names since they may be used in reports or in other exercises.

### Useful levels

| | |
|---|---|
| • Leg | atomic object |
| • Duty | at least X hours between arrival and departure |
| • Trip | new trip identifier |
| • Chain | complete chain/roster |

### Log on

***Step 1*** Double click the Academy Desktop icon
You are now logged in to windows as student## (see sticker on screen)

***Step 2*** Double click the Exceed on Demand  icon

***Step 3*** Double click the Launcher start icon RAVEII_SYSTEM
Just press **Yes/Run** in the certificate warning pop ups

***Step 4*** Type your unix **password**

***Step 5*** In the Launcher change Roles to **Developer** by right clicking in the background

***Step 6*** Start Studio by clicking the Studio icon

Since you have not done anything yet the system cannot find a rule set, and will produce a warning message: read, understand, and click **OK**.

***Note*** It is always important to read all messages and make sure you understand them.

If the message is unreadable, try to increase the size of the message window.

***Note*** In these exercises you will use a rule set where modules are used. This is the case for most clients but not all.

## Compile a rule set using DWS

Start DWS by selecting the command
**Admin Tools > Developer Workspace** or from **Launcher**.

Wait for DWS to initialize.

On the right hand side in the **Rule Set** tab click the ‘Choose active rule sets’ button (R with green tick) and select only Studio.

Select the only rule set, `default_ccr` and use the ‘**Rave compile…**’ button (R with hammer) to compile, select **Studio** with **Explorer** and **Optimize** options. Also click **Reload rule set** option.
Press **OK**.

# Rave I Test Recap

## ***Exercise 1*** Purpose

To review concepts from the Rave I course, via the Rave I Test.

This will be an interactive session.

# Tables

## *Exercise 2* Purpose

To learn more about tables.

## *Exercise 2.1* Row count

In the empty file `raveii_exercises`, create a variable which counts the number of exception rows for a particular crew member in the external table `SpLocal/RuleExceptions.etab`.

Test with plan: `Course/RaveII/june/ASSIGNED`

Use **Special > Rave > Rave Evaluator…** for crew 714933

## *Exercise 2.2* Clusters

Clusters are used when you first want to match a detailed condition then 'loosen' up the criteria if there is no match.

Create a table that evaluates the min connection time, depending on the current flight number and whether the leg is a deadhead.

Active, 2000-2900 ➔ 0:30

Deadhead, 2000-2598 ➔ 0:20

Active, 2901-3000 ➔ 0:45

Deadhead, 2599-3000 ➔ 0:30

Other active flights need at least 1:00 and the deadheads 0:40.

## *Exercise 2.3* Generic row count

Create a function that returns the number of rows in **any** external table.
Test with e.g. `SpLocal/RuleExceptions.etab`.

## *Exercise 2.4* Extra: External 'Wild Cards'

In internal tables it is possible to use '-' as a wildcard when the value in a certain column should not be checked. A common implementation for external tables has been to use several internal lookups and go through them one-by-one until a match is found. This is not ideal for Planners as it is not transparent in what order the table is searched.

After the normal crew need calculations have been done, we would like to add an extra service depending on ac-type, block time and departure airport. The criteria could be (in this order):

Everything departing from GOT always have +4 service crew!

73D always have +3 in crew need

73G,  >1:40 have +2

> 1:30 departing from PRG have +1

**Hint:** Use `From` and `To` columns and match all values within an interval.

**Hint:** 0 < A < z

**Hint:** Put the external table under **Default Sub-plan Tables**.

# Modules

Keep in mind that some modules are used by both Studio and the optimizer, whereas some modules are only used in one of them. This behaviour is controlled by the following construction, located in the top source file:

```
#if Product(<Matador|Studio|...>)
     use <module>;
end
```

## ***Exercise 3*** Purpose

To learn more about modules and inheritance.

You may work in the current modules; create new modules or new instances of modules.

## ***Exercise 3.1*** Inheritance

In the rule set `default_ccr` the planning period is set to one month. Create a new rule set that is identical to `default_ccr`, except for that the planning period lasts for a configurable number of weeks. Do this by redefining the planning period definitions in module `pp_basic`.

***Note*** The rule set file `default_ccr` may not be changed.

***Step 1*** Load sub-plan `Course/RaveII/june/ASSIGNED`.

***Step 2*** Create a new rule set `default_ccr_weeks`
(See Hint below.)

***Note*** The new rule set should only be used in this exercise. All other exercises should use rule set `default_ccr`.

***Step 3*** Change the planning period's end.
Let the number of weeks in the period be a parameter `num_weeks`.

***Step 4*** Add the string "`(n weeks)`" to the label/remark "`Start of planning period`" in the parameter form.

***Step 5*** Load the new rule set and reload the parameters from the assigned plan.

***Step 6*** Try different planning period lengths to see what happens with the rules.
(5 WEEKS should yield one illegal crew)

> **Hint:** The following step-by-step is a convenient way, in DWS, to create a new rule set which uses a new version of a module (using inheritance).

1. Find `default_ccr` in Project Explorer tab
2. Copy and Paste, and change names
3. Open the new rule set file

4. Go to the modules directory and create a new file only containing the line `module <new_module> inherits <old_module>`
5. Enter new file name (`crc/modules/) <new_module>`
6. Make sure the old module is a root module `root module <old_module>`
7. Modify the top source file:
   Change string :
   `"use <old_module>;"` to `"use <new_module>;"`
8. Analyze the rule set using the **Rule Set** tab
9. The new module should now be incorporated in the rule set. You should see it as a sub-module of the old root module

## *Exercise 3.2* 'use' and 'import' in the top file.

In the top file `default_ccr`, investigate what happens if you remove the following lines from the rule set. Make sure that you understand what is happening in each case.

- Remove the line "import levels;". Analyze. What happens? Why?
- Remove the line "use constraints;". Analyze. What happens? Why?
- Remove the line "use qualifications;". Analyze. What happens? Why?
- Remove the line "use leg;". Analyze. What happens? Why?

# Contexts

## ***Exercise 4*** Purpose

To learn more about contexts.

***Note*** The exercises have been divided into Pairing and Rostering parts; you may choose the part that best suits your needs.

***Note*** Use the **original rule set** `default_ccr` for all the following exercises.

## ***Exercise 4.1*** Contexts – Crew Pairing

You are interested in the distribution between long and short trips in the plan. The report `CountLongTrips` shows the number of long trips, and the percentage of long trips. Try the report in the trip window; you need to supply the correct values.

***Step 1*** In the current window, count the number of trips that are longer than three days. Do not consider trips that are ground duties.

```
crg_basic.crr_window_num_long_trips
```

> **Hint:** If `leg.%is_flight_duty%` is true on any leg in the trip, then the trip should be considered

***Step 2*** Calculate the percentage of long (> 3 days) trips. Do not consider trips that are ground duties.

```
crg_basic.crr_window_percentage_long_trips
```

***Step 3*** Run report `CountLongTrips`.

***Step 4*** Compare the distribution of long trips in the trip window with the trips in the crew window.
For the ASSIGNED plan with the environment loaded, there should be 5% long trips in the crew window and 2% in the trip window.

## ***Exercise 4.2*** Contexts – Crew Rostering

There is often a need to generate reports on objects that are currently displayed in the working window. This means that you first filter interesting objects, then generate the report. In this report (that is generated from the General Assignment pop-up menu) we want to count the number of crew and how much monthly block time they have together.

Use the variable `roster.%block_time%` to get a roster's total block time.

***Step 1*** Calculate the number of crew and total block time in the current window

```
[crg_basic.%crew_window_num_crew%,
 crg_basic.%crew_window_block_time%]
```

***Step 2*** Generate report `TotalBlockTime`.

For the ASSIGNED plan with the environment loaded, the total block time for the 5 crew having a surname starting with B should be 799:06

***Step 3*** How would you count all crew members in the sub-plan, not just the crew in the window?

# Transforms and Constraints

## *Exercise 5* Purpose

Learn about transforms, and how to implement vertical constraints for Studio and the optimizers.

This exercise is divided into Crew Pairing and Crew Rostering.

## Transforms

In these exercises you will model the problem by using transforms. The syntax is Studio specific since the optimizers do not support transforms as extensively as Studio does. Therefore you need to structure the code in a way so that the rules are only compiled for Studio.

## Constraints

Write the constraints in the module `constraints`.

**Hint:** Check your constraints with:

```
    Generate Report > check_constraints.py
```

**Hint:** Or use the report in the special menu.

## Crew Pairing

Load the plan `TRIPS`.

Show trips.

## *Exercise 5.1* Max number of deadheads on leg

In order to prevent deadheading crew from occupying seats, you want to limit the number of deadheading crew on a flight to two.

First create a rule, then a constraint.

***Step 1*** Count the number of crew deadheading on a flight across all trips in the plan. Use the `definition %assigned_crew_on_leg%.` It calculates the number of crew assigned to one single leg (in one trip).

```
studio_rules.%nr_dh_per_flight%
```

**Example**
2 trips with assignment vectors 0/2/1 and 0/1/0 use the same flight as a deadhead.
`studio_rules.%nr_dh_per_flight%` on that deadhead should then evaluate to 4.

***Step 2*** Write a rule that limits the number of crew deadheading on a flight. Define a parameter for the max value.
`studio_rules.max_dh_on_flight` **(rule)**
`rules.%max_dh_on_flight_param%`

> **Hint:** Put the parameter in the rule root module.
> You will use it in the next exercise.

## *Exercise 5.2* Max number of deadheads on flight

Write a vertical **constraint** that limits the number of crew deadheading on a flight.

Load the Trip plan and generate the `Trip General` report `check_constraints.py` to determine if there are any broken constraints. You should get the same number of trips as the number of illegal trips.

Define `failtext` so that it reports the `crr_name` of the trips that are violating the constraints.

> **Hint:** Try also the command:
> **Special** > **Reports**> **Global Constraints**
> It prints global constraints as the Crew Pairing Optimizer does.

## *Exercise 5.3* Limit number of long trips per day

Write a global constraint that limits the number of long trips that are allowed to start per day.

Define a parameter which sets the minimum length (in number of days) for a long trip.

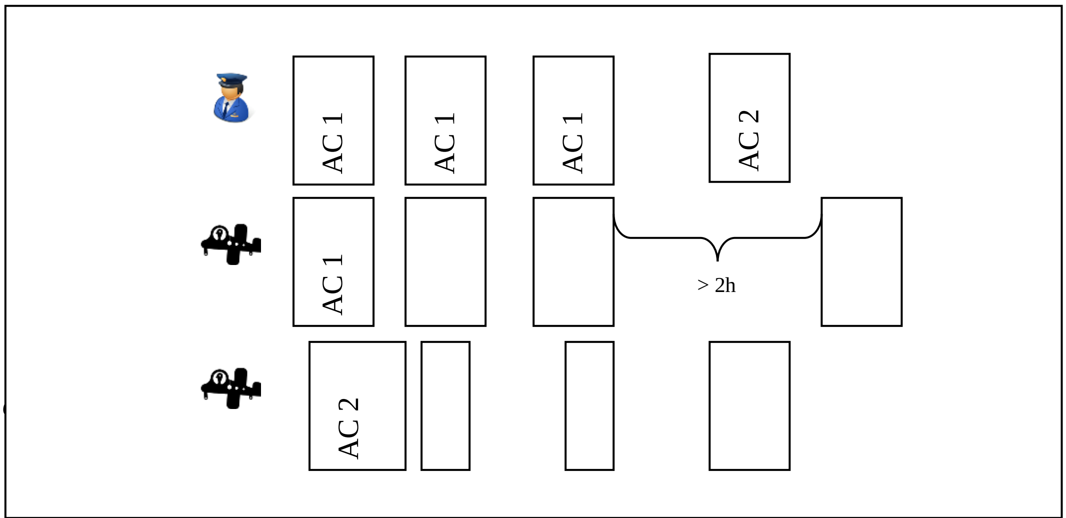
Use a `foreach` statement that iterates over the days in the planning month.

Define `failtext` so that it prints the current day. The overshoot should be printed if the constraint is violated.

***Note*** In a real implementation to be used by the Crew Pairing Optimizer a nonadditive definition would be needed too.

## *Exercise 5.4* Favourable aircraft connection

We have decided that it is favourable to let crew change aircraft when the aircraft on a leg has a long turn (>2:00). To calculate this you need to look at the aircraft rotation chain. Then you can see from the aircraft connection



time whether you have a favourable aircraft change after the current leg.

***Step 1*** Calculate the aircraft connection time for a leg
`crg_basic.%ac_cxn_time%`

***Step 2*** Implement the parameter and the variable
`crg_basic.%min_favourable_ac_cxn_time% (default 2:00),`
`crg_basic.%is_favourable_ac_change%`

***Step 3*** In Studio, find the legs that have a favourable aircraft (AC) change.

> **Hint:** Select with the Rave variable and verify with the command **Object > Leg > Show > Rotations**
>
> **Hint:** Also look at the turnin_ and turnout_ keywords.

# Crew Rostering

## *Exercise 5.5* Max inexperienced crew per trip

Create a rule and a constraint:

```
studio_rules.max_inexperienced_on_trip
constraint.max_inexperienced_on_trip
```

> **Hint:** Useful variables:
> `rules.%inexperienced%,`
> This is a trip-dependent variable which depends on which crew the trip is currently assigned to. Returns `TRUE` if the trip is assigned to an inexperienced crew
>
> `rules.%max_inexperienced%`

Define `failtext` to report how many inexperienced crew are allowed on a trip.

Load the `SMALL` plan with the new rule set.

Switch on the new rule and the new constraint.

Are there any broken constraints/rules?

What happens if you change the parameter `%max_inexperienced%`?

## *Exercise 5.6* Limit unassigned trips per day

Write a constraint that limits the number of unassigned trips starting per day.

Use a `foreach` construction that iterates over each day in the planning month.

Define `failtext` so that it prints the current day. The overshoot should be printed if the constraint is violated.

> **Hint:** Use the variable `crew_pos.%trip_assigned%` and note that it has different implementations for Studio and the Crew Rostering Optimizer.

## ***Exercise 5.7*** More senior First Officers?

The more senior you are, the lower your seniority number is. We are looking for legs where the First Officer’s seniority number is lower than the Captain’s seniority number.

***Step 1*** Write a leg variable that returns `TRUE` if the First Officer’s (CO) seniority number is lower than the Captain’s (CP).
`crg_basic.%leg_co_is_more_senior%`

***Step 2*** Select all legs in the sub-plan where `leg_co_is_more_senior` is `true`.

### Example

Crew IRVGM & JHNNK, trip on 5th of June. (plan `BIG`)

**Hint:** Use the keyword `assigned_crew_position_X`, to see if the leg is assigned to position X.

| | |
|---|---|
| Captain (CP) | = position 1 |
| First Officer (CO) | = position 2 |

Remember to protect from legs in unassigned trips.
(The definition `fundamental.%is_roster%` returns `true` if the trip is assigned to a crew.)

Remember to protect from `void` values.

Use one variable for the CO seniority number of the leg,
and one variable for the CP seniority number of the leg.

# Performance and Caching

## *Exercise 6* Purpose

- To be able to improve Studio performance by finding slow or unnecessary definitions.
- To develop a familiarity with Rave profiler.

## *Exercise 6.1* Profiler analysis

***Step 1*** Select the Rave Profiler option and recompile the rule set.

***Step 2*** When the rule set is compiled, open the `BIG` plan with environment plan `ENVIRONMENT` in Studio.

To take measurements with Rave Profiler, follow these steps:

1. **Admin Tools >Rave Profiler >Start Rave Profiling**
2. Perform the operations that you want to measure
   3. Run a report (several times)
   4. Show illegal rosters (several times)
   5. Scroll all rows (up and down)
6. **Admin Tools >Rave Profiler >Stop Rave Profiling**
7. Save profile
   **Admin Tools >Rave Profiler >Save Profile**
8. In DWS:
   9. Click **Rave Profiler** perspective (blue stopwatch) and enlarge the window
   10.press **Load**, and find your saved profiler run
   11.Set **Threshold** to find a usable detail level (e.g. 0.1)

***Step 3*** Use Rave profiler to analyse where most of the time is spent.

***Step 4*** Try to turn off rudobs, rules and constraints. Does it make a difference?

***Step 5*** Modify the Rave code to improve the scrolling speed.

## *Exercise 6.2* Rewrite definition

What is wrong with the definition `roster.%scaled_block_time%`?

***Step 1*** Rewrite it to improve its performance.

## *Exercise 6.3* Extra: performance

In the online help, read this section:

*Development > Rave Reference > Appendix: Modelling information > Improving the performance of Rave code*

# Costs and Rules

## Costs

***Exercise 7***

### Purpose

To practice some cost functions and get familiar with rules, failtexts and rule exceptions.

***Exercise 7.1***

### Cost of roster

Study the variable `cost.%roster%`.

**Hint:** Use **Special** > **Optimization & KPIs** > **Generate and Display KPIs** to get detailed information about the cost.

***Exercise 7.2***

### Cost of deadheads

***Step 1*** Write a quadratic penalty for deadheads in a trip. There should be no penalty for only one deadhead.

***Step 2*** Add a use parameter

***Note*** This is a bad cost (not additive in the duty network) for the Crew Pairing Optimizer.

## Rules

For the following exercises:

- Work in the plan `Course/RaveII/june/ASSIGNED`.
- Load plan `../ENVIRONMENT` as environment if nothing else is stated.

***Exercise 7.3***

### Valid

Rewrite the following rule (on paper) using a `valid` statement. The behaviour of the rule should be unchanged.

```
rule r =

  not (%a% and %b%)
  or %x% <= %max%;

end
```

***Exercise 7.4***

### Valid, part II

Examine the valid statements in the rules file.

Are there any reoccurring variables?

Considering lazy evaluation, how should the reoccurring variables be sorted?

## ***Exercise 7.5*** Fail text

For the rule `acc_duty_time_14days` add a remark and a fail text.

***Note*** When using the Alert Monitor, the fail text should never be longer than 40 characters. You should try to create a text that is possible to understand “stand alone”.

> **Hint:** The report generated by the command “check legality” in the roster window displays the fail text in the first column.

## ***Exercise 7.6*** Rule exceptions

The rule exception mechanism is already implemented in the course user. However, all rules do not yet allow for rule exceptions.

Make it possible for Rave to consider rule exceptions for the following rule:

`rules.max_month_credit_block_time`

Before you start, make sure that there are crew that break this rules,
e.g. you can change the parameter settings.

> **Hint:** Turn on ‘Show Children’ in the properties form to view rule parameters in the parameters form.

### Create a rule exception

In the left margin in Studio, click an illegal crew and select the command **Rule Exceptions** > **Create** in the Crew (margin) pop-up menu.

### View the rule exception external table

Select the command **Planning Tools** > **External Table Manager**.

Select
**Directory -** `Sub-plan tables (SpLocal)`
**File** `RuleExceptions.etab`
and click **View** or (**Text View**).

## ***Exercise 7.7*** Extra: Deadhead Quality

Add your cost definition (7.2) as a new Quality element in the existing `cost.%roster%` variable.

## ***Exercise 7.8*** Extra: Deadhead Quality, part II

Update the variable names and structure to fit the existing patterns.

# Accumulators

## *Exercise 8* Purpose

To learn more about accumulators.

## *Exercise 8.1* Accumulators

The rule `acc_duty_time_14days` restricts the accumulated duty time over 14 days. The rule does not use accumulators, so the rule only works if the loaded plan contains data for at least 14 days before the planning period.

**Create a new rule which uses an accumulator in order to remove the need of historical data in the plan**.

**`rule`**`.acc_duty_time_14_days_b`

***Step 1*** Create an accumulator for accumulated duty time. (You are welcome to reuse existing code such as the `rules.%accumulated_duty_time%` function.)

***Step 2*** Create the new rule, using the accumulator.

***Step 3*** Accumulate the data in the `ENVIRONMENT` plan.

This step is easiest done by loading the `ASSIGNED` plan, loading the `ENVIRONMENT` as Env, and then accumulating the time period 01Apr2004 to 01June2004.
(Make sure the `acc_start` and `acc_end` attributes are set correctly).

An empty accumulator table `accumulator_rel` has already been prepared, and is located in the `CARMDATA/ETABLES` in the Etable Manager.

***Step 4*** Compare the new and the old rule. Are there any differences?
In that case, why?

This step is easiest done by using the reference plan functionality:

1. Load ASSIGNED (the environment will be included)
2. Unload the environment plan
3. Load ASSIGNED as Ref. (the environment will be included)
4. You now have 2 rows per crew.
   One with the 2-month history and one without.
5. Turn on both rules, and show illegal crew
   e.g. Look at crew: LANRE 771748
6. Experiment with turning on and off the rules.

# Solutions

## Exercise 1 Rave I Test Recap

## Exercise 2 Tables

### Exercise 2.1 Row count

```
table bid_rows_tab(string crew_id) =
    crew_id -> int %bid_rows%;
    external "SpLocal/RuleExceptions.etab";
    "CrewId" -> count(row_number);
end
```

**Note** You have to count an integer column.

### Exercise 2.2 Clusters

```
table min_connection_tab =
    flight_number, deadhead
        -> %min_connection_time%;
    (2000, 2900), False -> 0:30;
    (2901, 3000), False -> 0:45;
    (2000, 2598), True  -> 0:20;
    (2599, 3000), True  -> 0:30;
    &
    -, False -> 1:00;
    -, True  -> 0:40;
end
```

### Exercise 2.3 Generic row count

```
 table num_rows_in_tab(string table_path) =
    0 -> global export Int %num_rows_in_table%;
    external table_path;
    < row_number -> count(row_number);
end
```

## Exercise 2.4 Extra: External ‘Wild Cards’

```
import leg;
  …

table wild_card_extra_crew =
    aircraft_type,
    leg.%block_time%,
    departure_airport_name
    -> Int %extra_service_crew%;
    external "SpLocal/ExtraServiceCrew.etab";
    (ac_from, ac_to),
    (block_from, block_to),
    (dep_from, dep_to)
    -> extra_crew;
    -, -, - -> 0;
end
```

ExtraServiceCrew.etab:

```
7
Sac_from,
Sac_to,
Rblock_from,
Rblock_to,
Sdep_from,
Sdep_to,
Iextra_crew,

“000”, ”ZZZ”, 0:00, 24:00, “GOT”, “GOT”, 4,
“73D”, ”73D”, 0:00, 24:00, “AAA”, “ZZZ”, 3,
“73G”, ”73G”, 1:41, 24:00, “AAA”, ”ZZZ”, 2,
“000”, ”ZZZ”, 1:31, 24:00, “PRG”, “PRG”, 1,
```

**Note** The planner now has the choice to decide that row 1 is more important than row 2 (i.e. if a 73D takes off from Gothenburg) by switching them around.

# Exercise 3 Modules

## Exercise 3.1 Inheritance

1. Copy `source/default_ccr` to `source/default_ccr_weeks`
2. Change the file `modules/pp_basic` to become a root module.
   Just add `root` in before `module`.
3. Create a new file `modules/pp_basic_weeks`.
4. Add the new definitions in `pp_basic_weeks`:

```
module pp_basic_weeks inherits pp_basic
redefine %start_para% =

    _
    remark "Start of planning period (n weeks)";
%num_weeks% = parameter 4
    remark "Number of weeks in planning period";
redefine export %end% =
    %start% + %num_weeks% * 7 * 24:00;
```

5. In `source/default_ccr_weeks`:
   change the line “`use pp_basic;`” to “`use_pp_basic_weeks;`”
6. With `%num_weeks% = 5` or `6` one roster should become illegal.

**Note** Remember to load the `ASSIGNED` sub-plan, your new rule set and the parameters from the plan.

## Exercise 3.2 ‘use’ and ‘import’ in the top file.

Discuss the results in class.

- `levels` is used in the `_topmodule` so it needs to be imported
- `constraints` is just a basic module (and not used anywhere else in Rave, although it could be used in reports/scripts)
- `qualifications` is imported by other modules and is also a basic module
- `leg` is a root module and needs a use to define which version to include.

# Exercise 4 Contexts

## Exercise 4.1 Contexts – Crew Pairing

In module `crg_basic` write the following code:

```
%crr_window_num_long_trips%=
     context(default_context,
            count(trip_set)
             where (trip.%days% > 3
                    and %is_flight_duty%));
%crr_window_num_all_trips%=
     context(default_context,
             count(trip_set)
              where (%is_flight_duty%));
%crr_window_percentage_long_trips% =
     100
     * %crr_window_num_long_trips%
     / %crr_window_num_all_trips%;
%is_flight_duty% =
     any(leg(trip), leg.%is_flight_duty%);
```

## Exercise 4.2 Contexts – Crew Rostering

In module `crg_basic` write the following code:

```
import roster;


[...]

%crew_window_num_crew% =
     context(default_context,
             count(roster_set));
%crew_window_block_time% =
   context(default_context,
           sum(roster_set,
               roster.%block_time%));


/* Step 3. count all the crew in the plan: */
%sp_num_crew% =
    context(sp_crew,
           count(roster_set));


*) the iterators are globally exported
```

## Exercise 5 Transforms and Constraints

**Note** When writing constraints, always be aware of the impact on performance. Sometimes it is enough to just add a simple cost or even disregard since it will not be part of the solution anyway.

## Exercise 5.1 Max number of deadheads on leg

```
/* in studio_rules */

rule (off) max_dh_on_flight =

    valid leg.%is_deadhead%;

    %nr_dh_per_flight% <= %max_dh_on_flight_param%;

    remark "Max crew deadheading on flight";

end


%nr_dh_per_flight% =
    sum(equal_legs, %assigned_crew_on_leg% )
    where (leg.%is_deadhead%);

/* in rules */

export %max_dh_on_flight_param% = parameter 2
    remark "Max number of deadheads on a flight: ";
```

## Exercise 5.2 Max number of deadheads per flight

```
/*pairing MAX deadhead*/


import leg;
import rules;

constraint (off) max_dh_on_flight_constr =
   sum(equal_legs, rules.%assigned_crew_on_leg% )
      where (leg.%is_deadhead%)
   <= rules.%max_dh_on_flight_param%;

   failtext(int lhs, int rhs) =
       concat(format_int(lhs, "Too many DHs %i "),
            format_int(rhs, "(%i) on leg"),
            " trip: ",
            crr_name);

   cost = 1;

   remark "Max number of deadheads on a flight: ";
end
```

**Note** The active slice is used in APC when the valid statement is calculated. Therefore you cannot check that the leg slice is active in the valid statement.

## Exercise 5.3 Limit number of long trips per day

```
/* Pairing max_long_trips */

import pp;

import levels;

import trip;

constraint (off) max_long_trips =
   foreach day in set(pp.%start%, pp.%end% - 24:00,
24:00);
   sum(default_context, %slots_on_day_chain%(day))
     where (first(trip(chain), trip.%num_duties%)
             >= %long_trip_is%)
   <= %long_trips_allowed%;
   cost = 1;
   failtext(int lhs, int rhs) =
      if lhs > rhs then
         concat("Long Trips: ",
                format_time(day, "%02d %b"),
                " Overshoot ",
                format_int(lhs - rhs, "%i"))
      else
         concat("Long Trips: ",
                format_time(day, "%02d %b"),
                " ok");
   remark "Max Long Trips per day";
end

%long_trip_is% = parameter 4;
%long_trips_allowed% = parameter 2;

%slots_on_day_chain%(Abstime day)=
   first(trip(chain), %slots_on_day%(day));

%slots_on_day%(Abstime day)=
   if trip.%start_day% = day then
      first(leg(trip), %assigned_crew_on_leg%)
   else
      0;
```

*) default_context is on ‘chain’ level and in our Roster system the level is called ‘chain’. We do this to level cast since %slots_on_day% is a Trip variable.

## Exercise 5.4 Favourable aircraft connection

```
%ac_cxn_time% =
   min(ac_rotations_ref,
      next(leg(chain), departure)- arrival);
%min_favourable_ac_cxn_time% = parameter 2:00;
%is_favourable_ac_change% =
   default( %ac_cxn_time% >=
         %min_favourable_ac_cxn_time%,
         true);
```

```
/* ac_cxn_time will be void for last leg in ac
chain, which should evaluate to a favourable ac
change */
```

## Exercise 5.5 Roster trip qualification

Rule:

```
rule (off) inexperienced_on_trip =

  valid %inexperienced%;

  count(equal_trips) where (%inexperienced%)
  <= %max_inexperienced%;

  failtext format_int(%max_inexperienced%,
              "Max %d inexperienced crew on trip");

  remark "Rule: max inexp on trip";

end
```

Constraint:

```
import rules;

constraint (off) max_inexperienced =

   count(equal_trips)
     where (rules.%inexperienced%)

   <= rules.%max_inexperienced%;

   failtext format_int(rules.%max_inexperienced%,

      "Max %d inexperienced crew on trip");

   remark "Constraint: max inexp on trip";

end
```

## Exercise 5.6 Limit unassigned trips per day

```
import fundamental;

import crew_pos;

%max_unassigned_per_day% = parameter 5;


constraint (off) max_unassigned_slots_per_day=

   foreach day in set(pp.%start%,

                      pp.%end% - 24:00,

                      24:00);

   sum(sp_crew_chains,
      first(trip(chain),
%slots_in_trip_on_day%(day)))
      where (not fundamental.%is_roster%)
   <= %max_unassigned_per_day%;

   cost = 5000;
   failtext(int lhs, int rhs)=
      if lhs > rhs then
          concat("Max unassigned ",
                 format_time(day, "%02d%b"),
                 " Overshoot ",
                 format_int(lhs - rhs, "%i"))
       else
           concat("Max Unassigned: ",
                  format_time(day, "%02d%b"),
                  " ok");
end
%slots_in_trip_on_day%(abstime day) =
    if trip.%start_day% <= day
       and trip.%start_day%
           + 24:00 * trip.%days%
           > day then
        crew_pos.%trip_assigned%
    else
        0;
```

## ***Exercise 5.7*** More senior First Officers?

```
import crew;
%leg_co_is_more_senior% =
    default(%leg_co_seniority%
            < %leg_cp_seniority%,
        false);
%leg_cp_seniority% =
    max(equal_legs, crew.%seniority%)
    where(assigned_crew_position_1 = 1
          and fundamental.%is_roster%);
%leg_co_seniority% =
    min(equal_legs,crew.%seniority%)
    where(assigned_crew_position_2 = 1
          and fundamental.%is_roster%);
```

# ***Exercise 6*** Performance and Caching

## ***Exercise 6.1*** Profiler analysis

The `%rudob_underbooked_len%` calculation is one example of where performance improvements can be made. One way to improve the definition is the following:

```
-----------------------
```

```
%rudob_underbooked_len%=
    if not %rudob_display_underbooked_legs%
       or (not leg.%is_flight_duty%)
       or crg_crew_pos.%unassigned_slots%=0
       /* order was switched.
        * leg.%is_flight_duty% is easier to
        * calculate than
        * crg_crew_pos.%unassigned_slots%
        */
    then
        0:00
    else
        arrival-departure;
-----------------------
export %unassigned_slots%=
    /* only check on legs that are in
     * the planning period: */
    if arrival < pp.%start% then 0 else
        sum(equal_legs, %assigned_position_sum%)
        where (void(crr_crew_id)));
-----------------------
/* Only sum the 7 first positions, because more are
   not used in this CARMUSR.
   */
export %assigned_position_sum% =
    sum(times(7),
        assigned_crew_position(times_index(0)))
-----------------------
```

**Another thing to change is the duty definition. This change gives a small but measurable improvement:**

Change:

```
levels.%levels_leg_is_last_in_duty% =

    next(leg(chain), departure) - arrival
    > %min_duty_connection_time%
    or
    (next(leg(chain), trip_id) <> -1)
    and (next(leg(chain),trip_id) <> trip_id);
```

to

```
levels.%levels_leg_is_last_in_duty% =
    next(leg(chain), departure) - arrival
    > %min_duty_connection_time%
    or
    (next(leg(chain), trip_id) <> trip_id)
    and (next(leg(chain), trip_id) <> -1);
```

***Note*** Here we use lazy evaluation to make a really small, but noticeable, change.

**However, the biggest problem is the** `equal_legs` **transform**. The usage means that Rave's level structure must be built for MANY chains.
**Can we get rid of it?**
Yes in this case we can, there are keywords to use that give the needed values without building any level structures for Rave.

```
export %unassigned_slots% =

   if arrival < pp.%start% then
      0
   else
      sum(times(7),

          booked_crew_position(times_index(0)) -

          rostered_crew_position(times_index(0)));
```

## *Exercise 6.2* Rewrite definition

Replace `roster.%scaled_block_time%` with:

```
export %scaled_block_time% =

    sum(wop(roster), wop.%scaled_block_time%);
```

Replace `trip.%scaled_block_time%` with:
`wop.%scaled_block_time%`

(in module wop:)

```
%scaled_block_time% =
    if %is_preassigned% then
        (80 * %block_time%) / 100
    else
        %block_time%;
```

## *Exercise 6.3* Extra: performance

First use the Rave Profiler to find the bottlenecks and where improvements will benefit the most. Then use these techniques to update the code.

# *Exercise 7* Costs and Rules

## *Exercise 7.1* Cost of roster

Talk in class.

## *Exercise 7.2* Cost of deadheads

```
%deadhead_cost_use% = parameter True
    remark "Use cost of extra deadheads: ";
```

```
%deadhead_cost_p% =
    parameter 100
    remark "Cost of extra deadheads (quadratic): ";
```

```
%num_deadheads% =
    sum(duty(trip), duty.%num_deadheads%);

%deadhead_cost% =
    let diff = nmax(%num_deadheads% - 1, 0);

    if %deadhead_cost_use% then
        diff * diff * %deadhead_cost_p%
    else
        0;
```

***Note*** This can be accomplished in many different ways.

## *Exercise 7.3* Valid

One example:

```
rule r =

    valid %a% and %b%;
    %x% <= %max%;
end
```

***Note*** This example is just for understanding. You should always aim for comparing an actual value to a limit in the main statement of a rule.

## *Exercise 7.4* Valid, part II

There are several variables that show up on most rules, e.g.

`wop.%in_pp% and wop.%is_on_duty%`

As all the rules are often evaluated together, these variables will be cached and are very fast to reuse. Thus they should be sorted early in the valid statements. The valid statement for `rule min_wop_rest_days` could be slightly changed.

```
valid
wop.%is_on_duty%
and next(wop(roster), wop.%is_on_duty%)

and wop.%in_pp%
and next(wop(roster), wop.%in_pp%);
```

```
valid
wop.%in_pp% and wop.%is_on_duty%
and next(wop(roster),
         wop.%in_pp% and wop.%is_on_duty%)
```

## *Exercise 7.5* Fail text

```
%accumulated_duty_time_this_duty% =
    %accumulated_duty_time%(duty.%end_hb% - 14 * 24:00,
                            duty.%end_hb%);
```

```
rule acc_duty_time_14days =
    %accumulated_duty_time_this_duty%
    <= %max_accumulated_duty_time%;
    remark "Max duty time in 14 days";
    failtext concat("Max duty time in 14 days: ",
                    format_time(%max_accumulated_duty_time%,
                                "%h:%02M" ),
                    format_time(%accumulated_duty_time_this_duty%,
                                "(%h:%02M)"));
end
```

## Exercise 7.6 Rule exceptions

```
export rule max_month_credit_block_time =
    month.%credit_block_time%
    <= %max_month_credit_block_time
       + rule_exceptions.%overshoot_rel%(
           pp.%start%);
    startdate = pp.%start%;
    enddate = pp.%end%;
    remark "Max credit block time in month";
end
```

## Exercise 7.7 Extra: Deadhead Quality

```
table cost_rq(int element) =
    ...
    5 -> "2.5 Cost for single days off",
       %rq_single_days_off_cost%;
    6 -> "2.6 Cost for deadheads in roster",
        %rq_deadheads_in_roster%;
    - -> "", 0;
end
%rq_deadheads_in_roster% =
    sum(trip(roster), %deadhead_cost%);
```

## Exercise 7.8 Extra: Deadhead Quality, part II

```
%trip_num_deadheads% =
    sum(duty(trip), duty.%num_deadheads%);

%wop_num_deadheads% =
    sum(trip(wop), %trip_num_deadheads%);

%rq_deadheads_in_roster_use% = parameter True
    remark "2.6.1  Use ost of extra deadheads: ";

%rq_deadheads_in_roster_weight% = parameter 100
    remark "2.6.2  Cost of extra deadheads (quad)";
```

```
%rq_deadheads_in_roster_cost% =
  if %rq_deadheads_in_roster_use% then
       %rq_deadheads_in_roster_weight%
       * sum(wop(roster), %wop_num_deadheads%)
         where(wop.%in_pp%)
  else
       0;

group quality =

  %cost_quality_header%,
  ...

    %rq_deadheads_in_roster_use%,
    %rq_deadheads_in_roster_weight%;

table cost_rq(int element) =
    ...
    6 -> "2.6 Cost for deadheads in roster",
        %rq_deadheads_in_roster%;

    7 -> "2.6 Cost for deadheads in roster",
        %rq_deadheads_in_roster_cost%;

    - -> "", 0;

end
```

## Exercise 8 Accumulators

## Exercise 8.1 Accumulators

In module rules:

```
/* This solution uses separate parameters for
acc_start and acc_end in order to make it easy for
the planner to change the range that should be
accumulated. In a real installation, we would
define acc_start=pp.%start% and
acc_end=pp.%end%+0:01;

*/

accumulator duty_time_acc(Abstime s,Abstime e) =

  %accumulated_duty_time%(s, e);

  key = crew.%id%;

  plan_start = pp.%start%;

  plan_end = pp.%end%;

  acc_start = %acc_start%;

  acc_end = %acc_end% + 0:01;

  acc_next(Abstime t) = t + 24:00;
```

```
end


%acc_start% = parameter 01Apr2004;

%acc_end% = parameter 01Jun2004;


rule acc_duty_time_14days_b =
  duty_time_acc(duty.%end_hb% - 14 * 24:00,
duty.%end_hb%)
  <= %max_accumulated_duty_time%;
end
```

# *Quick reference*

### Variables

```
%block_time% = arrival - departure;
```

### Constants

```
%briefing% = 0:45;
```

### Parameters

```
%briefing% = parameter 0:45
   remark “Length of briefing: ”;
```

### Functions

```
%work_start_with_offset%(Reltime offset) =
   %start_time% + offset;
```

### Built-in functions

```
%rest_time_after_duty% =
   default( next(duty(trip), %duty_start% )
         - %duty_end%,
      0:00 );

%trip_number_of_days% =
   (round_up(%trip_end%, 24:00)
   - round_down(%trip_start%, 24:00) ) / 24:00 ;

%required_rest_after_leg% =
   nmax( %block_time%, %min_rest_allowed% );
```

### Traversers

```
%trip_block_time% = sum( leg(trip), %block_time% )

%total_time_away% =
   last( leg (trip), arrival )
   - first( leg (trip), deparure );

%accumulated_block_time% =
   let this_dep = departure;

   sum( leg(trip), %block_time% )
   while departure <= this_dep);
```

## If-Then-Else

```
%time_between_legs% =
   if is_last( leg(trip) ) then
      0:00
   else
      next( leg(trip), departure ) – arrival;
```

## Tables

```
table hotel_costs_tab =
   %hotel% -> %hotel_cost%;

   “jerrys_inn” -> 250;
   “holliday_inn” -> 350;
   “sheraton” -> 500;
   “plaza” -> 650;
   - -> 999;
end
```

## External tables

```
table aircraft_family =
   aircraft_type -> String %aircraft_family%;

   external “aircraft_family”;

   ac_type -> ac_family;
   - -> ”no family”;
end

2
Sac_type,
Sac_family,

“747”, “747”,
“74E”, “747”,
“727”, “727”,
```

## Set

```
set asian_airports = “BKK”,”SIN”,”HKG”,”PEK”,”NRT”
   remark “Airports in Asia: ”;

Filters:

filter active_legs = leg( not deadhead );
```

## Rule

```
rule min_rest_time_in_trip =
   %trip_rest_time% >= %min_rest_time_in_trip_p%;
   remark “Minimum rest in Trip: ”;
end
```

## ( Penalty )

```
rule max_consecutive_deadheads =
   delta = 3;
   max_penalty = 100;
   min_penalty = 50;

   %nr_consecutive_deadheads%
   <= %max_consecutive_deadheads_p%;
end
```

### Levels:

```
level trip =
   is_last( duty )
   when( %duty_arrival% = homebase );
end
```

### Iterators:

```
iterator flight =
   partition(leg)
   by(departure_airport_name, deadhead );
end
```

### Context:

```
context( sp_crrs, count( chain_set ) );
```

### Transforms:

```
   sum(equal_legs, %tot_crew_leg%);
```

…