

# **BRVCE Tutorial**

Jalal Kawash

# Contents

<b>1 Preliminaries</b>	<b>2</b>
1.1 Running Example . . . . .	2
1.2 BR $\forall$ CE Interface . . . . .	2
<b>2 Relational Algebra Tutorial</b>	<b>6</b>
2.1 Projection and selection . . . . .	7
2.2 Aggregate functions . . . . .	10
2.3 Set operations . . . . .	12
2.4 Joins . . . . .	14
2.5 Division . . . . .	16
<b>3 Relational Calculus Tutorial</b>	<b>19</b>
3.1 Simple RC queries . . . . .	19
3.2 Existential quantifiers . . . . .	22
3.3 Universal quantifiers . . . . .	24

# Chapter 1

## Preliminaries

### 1.1 Running Example

In this tutorial, we will use the database schema depicted in Figure 1.1. This database keeps information about *employees*, who are described by their numbers, names (first and last), date of birth (DOB), gender (we assume the gender values are M for male, F for female, X for LGPTQ, and O for other), salary, and the department (number) they work for. A *department* is described by its number and name. Employees may have *dependents*, who are described by their name, gender, and relation to the employee. The *job* title of an employee is recorded with its effective start date (effDate).

The data set used in our tutorial is shown in Figure 1.2. It is contrived, and each table contains only a handful of rows. We used famous cartoon character names for employees, but we largely made up the rest of the information.

### 1.2 BR $\forall$ CE Interface

The BR $\forall$ CE interface was shown in Figure ???. In BR $\forall$ CE, the schema is loaded to the tool as an XML file. The XML representation of the schema of Figure 1.1 is given in Figure 1.3. It follows a very simple Document Type Definition (DTD) that lists the relations and their attribute names. To use your own schema, code in XML and load it to BR $\forall$ CE through the **File** menu choosing the *Load Database Schema* option.

Next, click the **Relational Algebra** or the **Relational Calculus** tab to start composing your RA or RC query, respectively. A query can be composed by se-

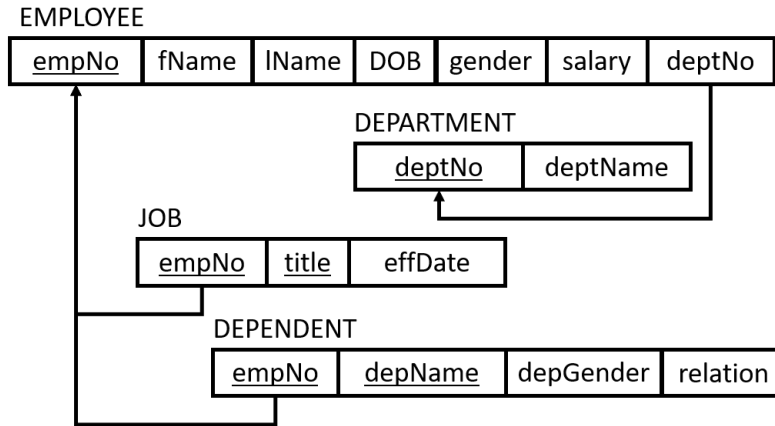


Figure 1.1: A simple company database schema

lecting a *Tile Group* and then dragging and dropping the required tile to the *query composition area*. Some tiles require a relation or an attribute name. BR $\forall$ CE populates the fields that require relation or attribute names in these tiles with drop-down lists. You can scroll down these lists and select the required relations or attributes for your query. Some tiles require a constant value or a quantifier variable name (only in RC). For these tiles, use the keyboard to enter the required value into the appropriate tile.

Once your visual query is composed and is complete, click the **Generate Code** button. The equivalent theoretical expression to your visual query and the equivalent SQL code will be shown in their respective areas. You can copy each of these expressions to the clipboard, using  $\square$ , or download them as a file, using  $\downarrow$ . You can save your query from the **File** menu. BR $\forall$ CE queries are saved with the extensions *raq* and *rcq* for RA and RC, respectively. Note that the schema representation is piggybacked to the query since queries are schema-specific. When you load a query from a file, the database schema is also loaded to BR $\forall$ CE.

**EMPLOYEE**

<b>empNo</b>	<b>fName</b>	<b>lName</b>	<b>DOB</b>	<b>gender</b>	<b>salary</b>	<b>deptNo</b>
18	Bugs	Bunny	1971-6-3	X	53000	3
19	Jessica	Rabbit	1985-6-14	F	65000	1
22	Daffy	Duck	1977-9-19	M	35000	6
25	Willma	Flinstone	1966-4-30	F	55000	3
33	Donald	Duck	1970-2-20	M	52000	3
38	Minnie	Mouse	1988-3-19	F	67000	1
40	Mickey	Mouse	1980-8-2	M	37000	6
41	Road	Runner	1992-11-22	O	40000	6

**DEPARTMENT**

<b>deptNo</b>	<b>deptName</b>
1	Human Resources
3	Information Technology
6	Sales

**DEPENDENT**

<b>empNo</b>	<b>depName</b>	<b>depGender</b>	<b>relation</b>
18	Hugs Bunny	X	Son
25	Fred Flinstone	M	Spouse
25	Pebbles Flinstone	F	Daughter
41	Hill Runner	O	Spouse

**JOB**

<b>empNo</b>	<b>title</b>	<b>effDate</b>
18	Developer	2000-1-1
18	Software Engineer	2006-1-1
19	HR Specialist	2002-3-1
22	Sales Manager	2002-1-1
25	Developer	2000-1-1
33	Developer	2001-1-1
33	Software Engineer	2004-1-1
38	HR Consultant	2003-4-1
40	Sales Person	2003-4-1
41	Sales Person	2005-7-1

Figure 1.2: A simple data set for the company database schema

```
<database name="Company Database">
  <relation name="Employee">
    <attribute name="empNo"/>
    <attribute name="fname"/>
    <attribute name="lname"/>
    <attribute name="DOB"/>
    <attribute name="gender"/>
    <attribute name="salary"/>
    <attribute name="deptNo"/>
  </relation>

  <relation name="Department">
    <attribute name="deptNo"/>
    <attribute name="deptName"/>
  </relation>

  <relation name="Dependent">
    <attribute name="empNo"/>
    <attribute name="depName"/>
    <attribute name="depGender"/>
    <attribute name="relation"/>
  </relation>

  <relation name="Job">
    <attribute name="empNo"/>
    <attribute name="title"/>
    <attribute name="effDate"/>
  </relation>
</database>
```

Figure 1.3: XML representation for the Company schema

## Chapter 2


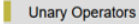
# Relational Algebra Tutorial

We will present queries of increasing complexity, formulate these in BR $\forall$ CE, and generate the equivalent RA and SQL expressions. We will then, use the Company database of Figure 1.2 to show the result of the query in that database. The operands in the RA tiles are labeled as follows:

1.  $R, R_1,$  and  $R_2$  are relation operands
2.  $a$  is an attribute name list operand
3.  $c, c_1,$  and  $c_2$  are logical conditions
4.  $e_1,$  and  $e_2$  are expressions which can be an attribute name or a constant value entered by the user.

An operand can be left blank if the label is enclosed in square brackets, such as  $[a]$ . We do not intend nor we have space to cover every possible RA operation supported by BR $\forall$ CE. However, we will demonstrate at least one RA operation from each each tile group.

The RA tile groups are:

1. The  Relations group contains the *relation* and *attribute* tiles to be used in the next two groups.
2. The  Unary Operators group contains the tiles for the RA unary operators (they have one relation operand): *select*, *project*, *aggregate functions*, and *aggregate functions with grouping*.

3. The **Binary Operators** group contains the tiles for the RA binary operators (they require two relation operands). Many of these operators share the same tile. The required operator is chosen from a drop-down list in the tile. There are four tiles in this group: (i) joins that do not require conditions (namely, *natural join* and *cross join*), (ii) joins that require conditions (all forms of *theta joins* and *outer joins*), (iii) set operations (*intersection*, *union*, and *difference*, and (iv) *division*.
4. The **Query Condition** group contains the tiles for formulating logic conditions. There are six tiles in this group: (i) the logical *and* or *or* tile, (ii) the logical *negation* tile, (iii) the comparison operators tile ( $=$ ,  $\neq$ ,  $<$ ,  $>$ ,  $\leq$ , and  $\geq$ ), (iv) the *attribute* tile needed for attribute names in conditions, (v) the *number literal* tile, and (vi) the *string literal* tile.

Any of the tiles in the unary and binary operators groups can serve as a *container* for the RA query.

## 2.1 Projection and selection

The *project* unary operator, denoted by the Greek symbol  $\pi$ , filters a relation vertically. That is, it can eliminate some of its columns. We start by formulating a projection query. Because this is our first RA query, we will go through its construction step by step. The query is *retrieve the first name and last names of employees*. The steps are:

1. Drag the project ( $\pi$ ) tile from the Unary Operators group and drop it in the query composition area:



2. The project tile requires a relation ( $R$ ) operand. From the Relations group, drag the relation tile and snap it into the project tile. Then, choose from the drop-down list the required relation:

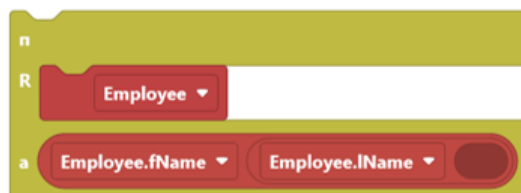




3. The project tile also requires an attribute list operand (*a*). From the Relations group, drag the attribute tile and snap into the project tile. Then choose from the drop-down list the required attribute:



4. More attributes can be added to the attribute list by snapping more attribute tiles onto the last added attribute tile:



This completes our query. Click the **Generate Code** button to generate the RA and SQL expressions that are equivalent to the BR $\forall$ CE query. The equivalent RA expression generated by BR $\forall$ CE is:

$\pi$  Employee.fName, Employee.lName (Employee).

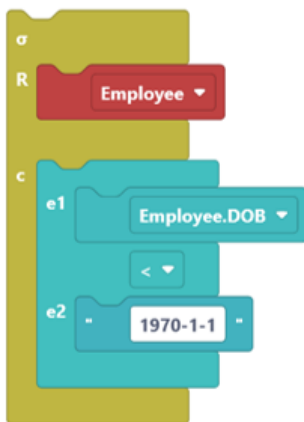
The equivalent SQL expression generated by BR $\forall$ CE is:

select Employee.fName, Employee.lName from Employee.

The result of this query when applied to our data set is:

fName	lName
Bugs	Bunny
Jessica	Rabbit
Daffy	Duck
Willma	Flinstone
Donald	Duck
Minnie	Mouse
Mickey	Mouse
Road	Runner

The *select* operation, denoted by the Greek symbol  $\sigma$ , filters a relation horizontally. That is, it can eliminate some of the rows. The following is a selection query that *retrieves the employees who were born before 1970-1-1*:



The equivalent RA expression generated by BR $\forall$ CE is:

$\sigma$  Employee.DOB < "1970-1-1" (Employee).

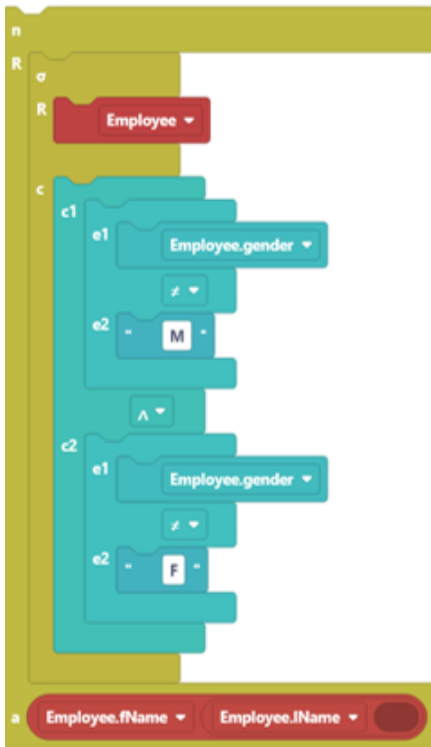
The equivalent SQL expression generated by BR $\forall$ CE is:

```
select *
from Employee
where Employee.DOB < "1970-1-1".
```

The result of this query is:

empNo	fName	lName	DOB	...
25	Willma	Flinstone	1966-4-30	...

The next query combines both projection and selection. It retrieves *the first and last names of employees who neither identify as males nor females*:



The equivalent RA expression generated by BRVCE is:

$\pi$  Employee.fName, Employee.lName ( $\sigma$  (Employee.gender  $\neq$  "M"  $\wedge$  Employee.gender  $\neq$  "F") (Employee)).

The equivalent SQL expression generated by BRVCE is:

```
select Employee.fName, Employee.lName
from Employee
where ( Employee.gender != "M"
and Employee.gender != "F" ).
```

The result of this query is:

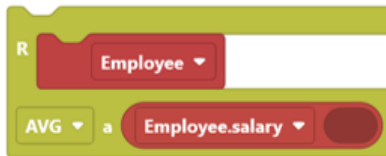
fName	lName
Bugs	Bunny
Road	Runner

## 2.2 Aggregate functions

The aggregate functions in RA are *min*, *max*, *sum*, *count*, and *average*. Calculations can be performed to compute a single value, such as the average salary in

the company, or to compute a single value for a group of rows, such as the average value per gender.

The following BR $\forall$ CE query *retrieves the average salary of all employees*:



The equivalent RA expression generated by BR $\forall$ CE is:

```
AVG (Employee.salary) (Employee).
```

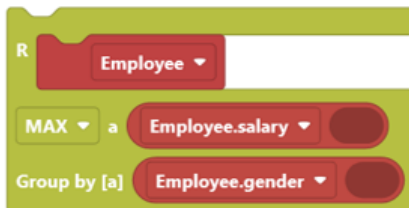
The equivalent SQL expression BR $\forall$ CE generated is:

```
select AVG (Employee.salary)
from Employee.
```

The result of this query is:

AVG(salary)
50500

To calculate any other function, such as the min, max, or sum, it is only necessary to choose that function from the drop-down list in the aggregate function tile. To *retrieve the maximum salary for each gender group*, the query is written as:



The equivalent RA expression generated by BR $\forall$ CE is:

```
MAX (Employee.salary) (Employee) (Employee.gender).
```

The equivalent SQL expression generated by BR $\forall$ CE is:

```
select Employee.gender,
       MAX (Employee.salary)
from Employee
group by Employee.gender.
```

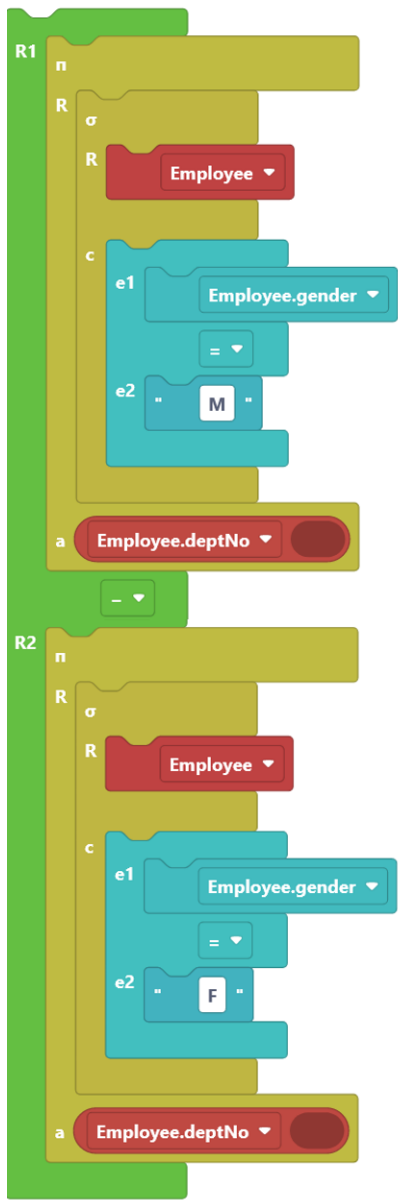
The result of this query is:

gender	MAX(salary)
X	53000
F	67000
M	52000
O	40000

## 2.3 Set operations

There are three set operations in RA: *union*, denoted by  $\cup$ , *intersection*, denoted by  $\cap$ , and *minus*, denoted by  $-$ . These are binary RA operators requiring two relations as operands. To retrieve the *depNos* for departments that have male employees but do not have female employees, we formulate a *minus* query. The first set ( $R_1$ ) contains departments that have male employees, and the second set ( $R_2$ ) contains departments that have female employees. The result is the first set *minus* the second set.

The BR $\forall$ CE query is:



The equivalent RA expression generated by BRVCE is:

$$(\pi \text{ Employee.deptNo } (\sigma \text{ Employee.gender} = \text{"M"} (\text{Employee})) - \pi \text{ Employee.deptNo } (\sigma \text{ Employee.gender} = \text{"F"} (\text{Employee}))).$$

The equivalent SQL expression generated by BRVCE is:

```
( select Employee.deptNo
from Employee
```

```

where Employee.gender = "M"
except
select Employee.deptNo
from Employee
where Employee.gender = "F" ).

```

The result of this query is:

deptNo	-	deptNo	=	deptNo
3		1		6
6		3		

Note that the query that *retrieves the depNos for departments that have both male employees and female employees* would only require changing the “-” to “∩” in the above query. The query that *retrieves the depNos for departments that have male employees or female employees* require using “∪” instead of the “-”.

## 2.4 Joins

Join queries cross reference two relations against each other by pairing each row in the first relation with each row in the second relation. For all the joins, except for the *cross-join* (denoted by  $\times$ ), a selection condition is applied to eliminate some of the irrelevant rows. A *cross-join* between Employee and Dependent lists every employee with every dependent:



The equivalent RA expression generated by BR $\forall$ CE is:

(Employee  $\times$  Dependent).

The equivalent SQL expression generated by BR $\forall$ CE is:

```

select *
from Employee cross join Dependent.

```

The *natural-join* (denoted by  $*$ ) eliminates the rows where empNo from Employee is not equal to empNo from Dependent in the above listing. The  $*$  must be chosen from the drop-down list in the tile and the equivalent RA expression is:

(Employee  $*$  Dependent).

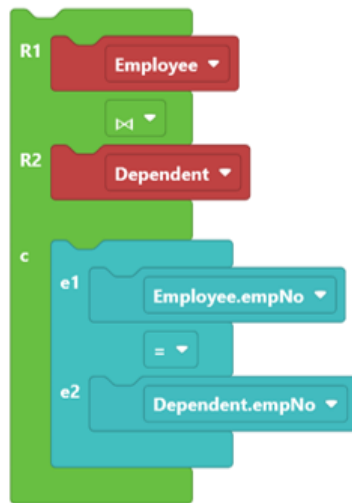
The equivalent SQL expression generated by BRVCE is:

```
select *
from Employee natural join Dependent .
```

The result of this query is:

EMPLOYEE			DEPENDENT		
empNo	fName	...	empNo	depName	...
18	Bugs	...	18	Hugs Bunny	...
25	Willma	...	25	Fred Flinstone	...
25	Willma	...	25	Pebbles Flinstone	...
41	Road	...	41	Hill Runner	...

This can also be expressed as a *inner-join* query:



Note that the natural and inner joins eliminate the employees who do not have any dependents. To include such employees in the result with the dependent information left blank when it is not applicable (for employees who have no dependents), an *outer-join* is needed. In the above query, it is sufficient to replace  $\bowtie$  by  $\bowtie\llcorner$  in the tile's drop-down list to create a right (Employee) outer join.

The equivalent RA expression generated by BRVCE is:

$(Employee \bowtie\llcorner Employee.empNo = Dependent.empNo Dependent)$ .

The equivalent SQL expression generated by BRVCE is:

```
select *
from Employee right outer join Dependent on Employee.empNo = Dependent.empNo.
```

The result of this query is:

EMPLOYEE			DEPENDENT		
----------	--	--	-----------	--	--



empNo	fName	...	empNo	depName	...
18	Bugs	...	18	Hugs Bunny	...
19	Jessica	...	19	NULL	...
22	Daffy	...	22	NULL	...
25	Willma	...	25	Fred Flinstone	...
25	Willma	...	25	Pebbles Flinstone	...
33	Donald	...	33	NULL	...
38	Minnie	...	38	NULL	...
40	Mickey	...	40	NULL	...
41	Road	...	41	Hill Runner	...

## 2.5 Division

Division requires a collection of attribute values to be “related” in some relation to every attribute value in another relation. For example, given the relations  $A$  and  $B$ :

$A$	
$a_1$	$a_2$
$x_1$	$y_1$
$x_2$	$y_1$
$x_3$	$y_1$
$x_4$	$y_1$
$x_1$	$y_2$
$x_2$	$y_2$

$B$
$a_1$
$x_1$
$x_2$
$x_3$

$A \div_{a_1} B$
$a_2$
$y_1$

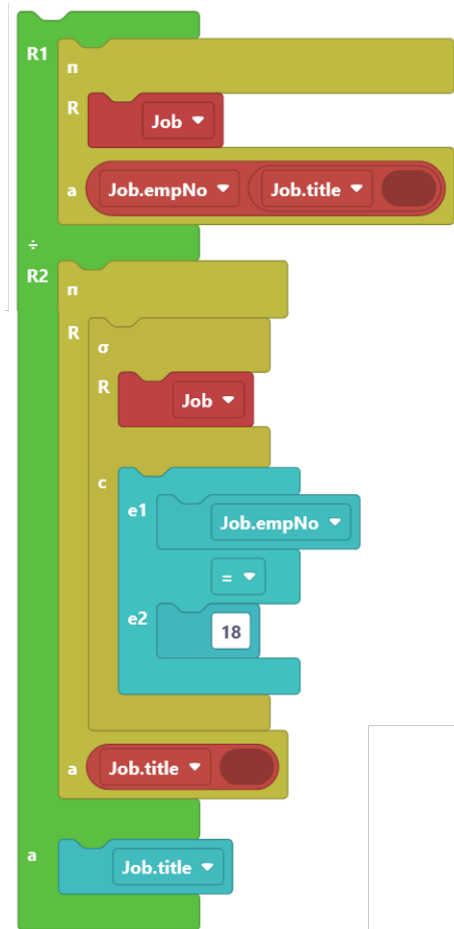
since  $y_1$  is “related” in  $A$  to every attribute value of  $a_1$  in  $B$ .

There is no direct support for the division operation in SQL. One way to express division queries in SQL is as follows. Let  $R_1(Z)$  and  $R_2(X)$  be relations and let  $Z$  and  $X$  be sets of attributes such that  $Y \subseteq Z - X$ .  $R_1(Z) \div_Y R_2(X)$  is written using other RA operations as follows:

$$\pi_Y(R_1) - \pi_Y((R_2 \times \pi_Y(R_1)) - R_1).$$

Note that our definition for division is more general than the classical division definition which assumes  $Y = Z - X$ .

To retrieve the numbers for those employees who held the same job titles as all the ones held by employee with empNo 18, requires the following division query:



The equivalent RA expression generated by BRVCE is:

$(\pi_{\text{Job.title}, \text{Job.empNo}}(\text{Job}) \div \text{Job.title } \pi_{\text{Job.title}}(\sigma_{\text{Job.empNo} = 18}(\text{Job})))$

The equivalent SQL expression generated by BRVCE is:

```
( select Job.title from ( select Job.title, Job.empNo from Job )
as temp0 except select Job.title from ( ( select * from ( select
Job.title from ( select Job.title, Job.empNo from Job ) as temp1
) as temp2 cross join ( select Job.title from Job where Job.empNo
= 18 ) as temp3 except select Job.title, Job.empNo from Job ) ) as
temp4 )
```

The result of this query is:

<b>empNo</b>	<b>title</b>
18	Developer
18	Software Engineer
19	HR Specialist
22	Sales Manager
25	Developer
33	Developer
33	Software Engineer
38	HR Consultant
40	Sales Person
41	Sales Person

$\div_{title}$

<b>title</b>
Developer
Software Engineer

=

<b>empNo</b>
18
33

# Chapter 3

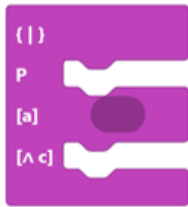
## Relational Calculus Tutorial

We will present four RC example queries in increasing complexity. There are four groups of tiles for RC in BR $\forall$ CE:

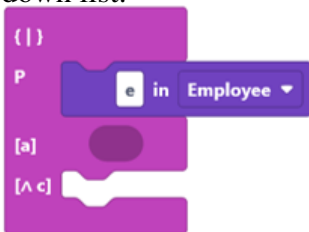
1. The **Main** group contains two tiles. The *main* tile is a container for all RC queries. The *attribute* tile is used to specify attribute names in RC queries. Similarly to RA tiles, optional operands are enclosed in square brackets.
2. The **Predicates** group contains two *predicate* tiles. The first tile represents the predicate  $P(x)$  and the second represents  $P(x) \wedge c$ , where  $c$  is a logical condition.
3. The **Quantifiers** group has two tiles. The *exists* tile corresponds to the predicate  $\exists x(P(x) \wedge c)$  and the *forall* tile corresponds to the predicate  $\forall x(P(x) \rightarrow c)$ , where  $c$  is a condition.
4. The **Query Condition** group has the same tiles as the same group in the RA tab.

### 3.1 Simple RC queries

We start with two queries that do not require the use of quantifiers. To *retrieve the employee names (first and last) who do not identify as male or female*, requires the following. First, the *main* tile is required as a container:



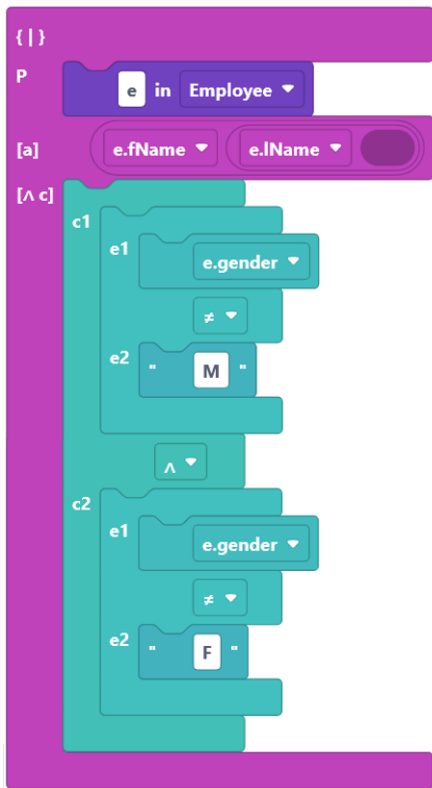
This tile requires a predicate ( $P$ ). From the Predicates group, drag and snap the simple predicate tile. Then change  $Var$  to  $e$  and choose Employee from the drop-down list:



Drag and snap two attribute tiles and select  $fname$  and  $lname$  from the drop-down lists:



Formulate the condition:  $(e.gender \neq "M") \wedge (e.gender \neq "F")$  using condition tiles, and the resulting query is:



The equivalent RC expression generated by BR $\forall$ CE is:

$\{e.fName, e.lName | Employee(e) \wedge ((e.gender \neq "M") \wedge (e.gender \neq "F"))\}$ .

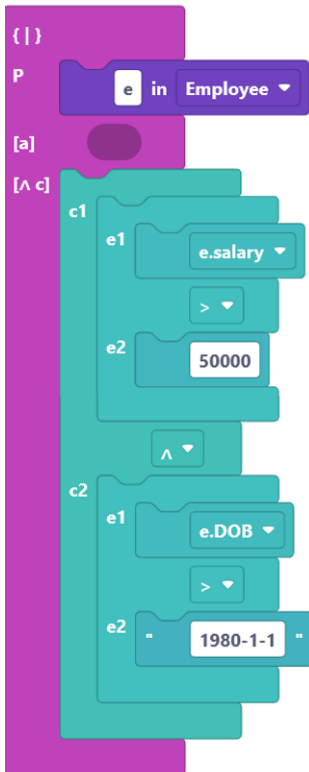
The equivalent SQL expression generated by BR $\forall$ CE is:

```
select e.lName, e.fName
from Employee as e
where ( e.gender != "M"
and e.gender != "F" ).
```

The result of this query is:

fName	lName
Bugs	Bunny
Road	Runner

The next query *retrieves the employees who earn more than 50000 and who were born after January 1, 1980* (note that the attribute list in the *main* tile is left empty):



The equivalent RC expression generated by BR $\forall$ CE is:

$\{e \mid Employee(e) \wedge ((e.salary > 50000) \wedge (e.DOB > "1980-1-1"))\}$ .

The equivalent SQL expression generated by BR $\forall$ CE is:

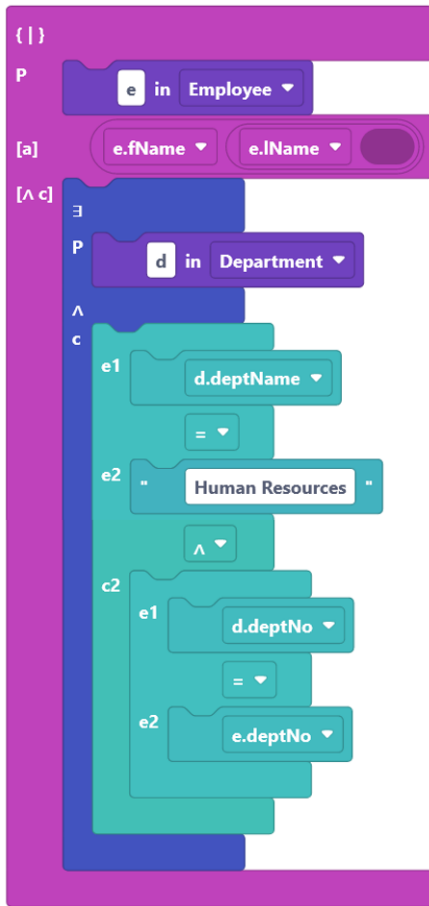
```
select *
from Employee as e
where ( e.salary > 50000
and e.DOB > "1980-1-1" ).
```

The result of this query is:

empNo	fName	lName	DOB	salary	...
19	Jessica	Rabbit	1985-6-14	65000	...
38	Minnie	Mouse	1988-3-19	67000	...

## 3.2 Existential quantifiers

Joins in RC require the use of the existential quantifier. To *list employee names who work for the Human Resources department*:



The equivalent RC expression generated by BR $\forall$ CE is:  
 $\{e.fName, e.lName | Employee(e) \wedge \exists d (Department(d) \wedge (d.deptName = "HumanResources") \wedge (d.deptNo = e.deptNo))\}$ .

The equivalent SQL expression generated by BR $\forall$ CE is:

```
select e.fName, e.lName
from Employee as e
where exists (
  select *
  from Department as d
  where (
    d.deptName = "Human Resources"
    and d.deptNo = e.deptNo )
).
```

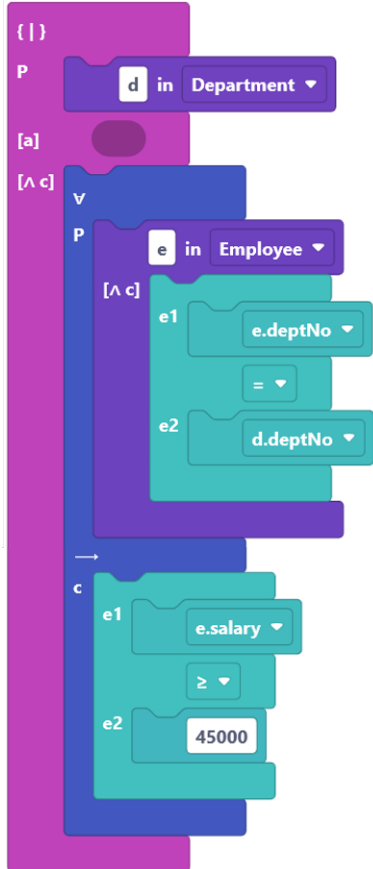


The result of this query is:

fName	lName
Jessica	Rabbit
Minnie	Mouse

### 3.3 Universal quantifiers

The following query retrieves the departments that have every employee earning at least 45000:



The equivalent RC expression generated by BR $\forall$ CE is:

$\{d \mid \text{Department}(d) \wedge \forall e((\text{Employee}(e) \wedge (e.\text{deptNo} = d.\text{deptNo})) \rightarrow (e.\text{salary} \geq 45000))\}$ .

The equivalent SQL expression generated by BR $\forall$ CE is:

select \*

```
from Department as d
where not (exists (
  select *
  from Employee as e
  where (
    e.deptNo = d.deptNo
    and e.salary < 45000 )
  ))).
```

The result of this query is:

<b>deptNo</b>	<b>deptName</b>
1	Human Resources
3	Information Technology

Note that *forall* queries can be represented in BR $\forall$ CE using the negation of the *exists* quantifier. For instance, the previous query can be also formulated as:

