Université Catholique de Louvain

LINGI2172 - Databases

Mission 3 - Database Design

Authors :
Baugnies Benjamin (6020-10-00)
Colson Olivier (5039-10-00)
Vanwelde Romain (3143-10-00)

Group 3

Supervisers :
Pr. Bernard Lambeau
Antoine Cailliau

16 mai 2014

1 Introduction

Telling a story is challenging. Indeed, in order to build a “good" scenario, one must think of a lot of different aspects and ensure coherence between all these points.

Some can manage it using diagrams, other can rely on tons of paper sheets referencing each other or, more reasonably, use a computer to store all their documents. However, with all these ways of working, the same problem arises : as the storyline and background get denser, it becomes more and more difficult to ensure that no contradiction appears. This is a big problem, since contradictions ruin the feeling of reality that must always be given by a good scenario. How about asking the computer to gather, interpret and display all this information in a clean and understandable way ?

Our project can be defined as a “narration manager". Its goal is to make it easier for people to write coherent and complex scenarios without either becoming mad or cancelling their project because of its increasing complexity. It is intended for all "story makers" (videogame makers, film makers, roleplayers, writers, ...), and is thus meant to be generic and conveniently adapt to various situations, as well as user preferences and priorities in the story (for example, some users could want to define a precise date for each event happening in their story, as others could prefer to focus on the relations binding all the characters together).

Possibilities of telling a story are infinite, yet time and coordination constraints often limit what is actually possible to achieve. It is now time to push these limits away.

2 Elementary Facts

Bellow are some elementary facts we wrote to better understand what to do, which relation exists between all the entities.

2.1 About characters

Pierre is from Bruxelles.
Pierre is born on 28/12/1992
Jean is born on 01/04/1992
Benjamin is born on 03/06/1992
Jean is melancholic
Benjamin is member of the association "Les Petits Riens"

2.2 Characters relations

Pierre liked Benjamin from 9/12/2002 to 13/7/2007.
Benjamin liked Pierre from 10/11/2003 to 12/8/2009.
Jean doesn’t like Benjamin from 10/11/2003 to 12/8/2009.
Paul is Pierre’s father.

Pierre is Paul's son since 28/12/1992.

2.3 About events

Jean attended the event "The beer festival"
The "beer festival" took place at LLN
The "beer festival" lasted from 08/03/2014 to 18/03/2014.
The "beer festival" is "blablablablablablablabla" as description.

2.4 About places

Intel room is a sub-location of Réaumur's Map and is represented on square number 10.
Réaumur is a sub-location of LLN's Map and is represented on square number 5.
The LLN's Map represents the location "LLN"
The Réaumur's Map represents the location "Réaumur"
LLNMap has 10 square width, and represents a 5km distance.
LLNMAP has 20 square length, and represents a 10km distance.

3 ORM schema

The ORM schema is shown on the Figure 1. If some relations are too difficult to read, the numeric version is available in the Annexes directory of the zip file.
The notations conventions used are those one :

- Entities are circled with straight lines
- Attributes are linked to entities with dashed lines.
- When the relation arity equals 2, we indicate the multiplicity/compulsory of the relation on the link.
- When the relation arity is bigger, we draw a line above the square representing the relation (If no lines are drawn, it is equivalent of having drawn the line on all squares). And this means that combination of all the attributes concerned by the line will always be different, whatever the relation expressed.
- If a relation is not clear enough, we could add some text in the squares to clarify the meaning of the relation.
- An underlined attribute means that it is a unique key.

As we can see on the schema, there are 4 main entities which are `"Characters"`, `"Events"`, `"Place"`, and `"Map"`.

We will explain here above three specific case of the diagram :

Character - Relation relations involving time range, time or timeless notion.
The relations explain the relationships between different characters. They are uni-directional and the different kinds of relations can be defined by the user. We split these relations into three types to represent the fact that some relations are time-independent (e.g "... is my father"), some can start at a given time and be permanent and/or open-ended (e.g "... is my godfather since ... "), and finally some can last for only a while (e.g "... was

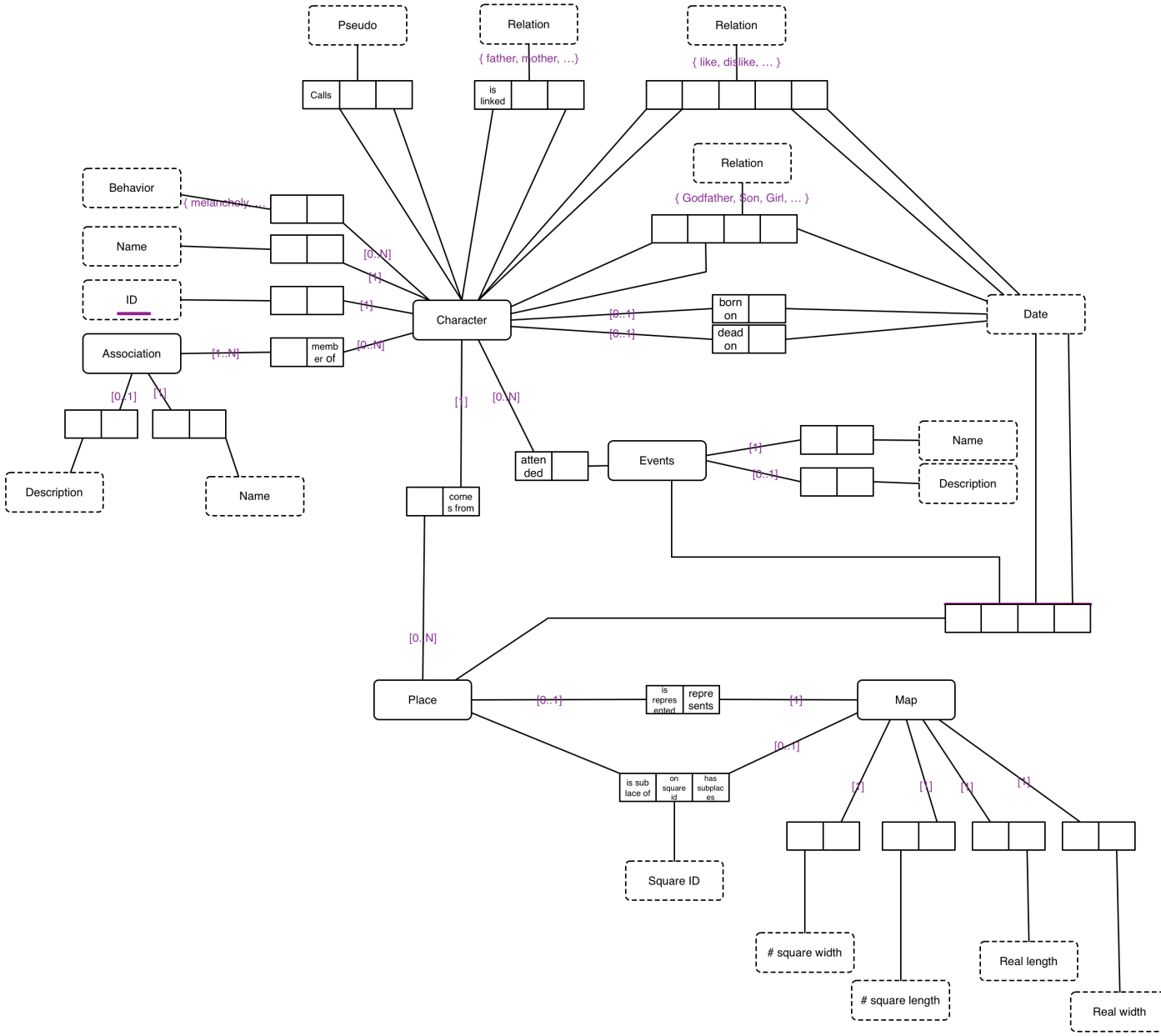

Figure 1 – ORM Schema

my friend from ... to ... ").

Pseudo relation.
This relation involves two characters and a pseudonym. It describes the name used by the first character for the second one. This represents the fact that during the story, a given character might not know another's real name. We added this relation since this can have an impact on the story (he wouldn't realize others were talking about someone he knows for example). The corresponding table will also allow us the find all the pseudonyms a given person might go by.

Place - Map relation.
This is a rather complex relation that we introduced to keep track of a story's geography at different levels. The first use is to allow users to situate events or characters. We can also define sub-places to refine locations. We can see that a place's map is optional. However, since a place's sub-places are linked to its map, it implicitly becomes required when we want to add levels. This structure allows us to chain an arbitrary number of levels with a "place - map - place - map - ... " hierarchy. This relation has two constraints that are not expressed in the database and will have to be verified in the software implementation. Firstly, the map square on which the sub-place is located must belong to the map's domain of possible squares (between 1 and (# square width)*(# square length)). Secondly, it is required that two places of the same level do not overlap.

4 Translating ORM in relvar

In this section , we will explain the way we convert our ORM diagram into relvar predicates.

First, we identify, for each entity, the compulsory binaries relations. All those are groupped in the same relvar.
Second, we create a new relvar for all the optionals binaries relations.
Last,we create a new relvar for all relations with an arrity bigger than 2.

5 Constraints

Among all the foreign key contraints, unique keys, ... we fond a more special and interesting constraint to express. We impose that the start date of a relation is lower than the end date. In tutorial-D and PostgreSQL, since we used custom types for our date representation, we implemented a function that allows us to know which date came before the other.

We realized that we could add other constraints (like oblige the month number to be lower than 12, the day number to be lower than 31, ...), but since while creating histories, we like having freedom (we can easily imagine a story where there are more than 12 months, 31 days, ...).

6 Tutorial-D script

In the zip file, you can find files `structure_rel.d` and `data_rel.d`.

structure_rel.d has three differents parts. The **first** one is the definition of special types (All the entities ID, and one for some names too). Then, the **second** part is all the relvars, with all primary keys, and also unique keys (e.g. `MAPPEDPLACE` which has one key on the PLACEID and another on the MAPID). The third part is all the constraints which are foreign keys. We express first all the constraints implying CHARID, then implying

ASSOCIATIONID, and so on.

data_rel.d fills the database with all elementary facts expressed before in the report.

7 Relvar predicates

In this section, we will give the relvar predicates that our database represents. The attributes of a relation are in bold, and the attributes that form the key are underlined.

About characters :

- Relvar [CHARACTER]
  The character **CHARACTERID** is named **NAME**.
- Relvar [BEHAVIOR]
  The character **CHARACTERID** has a behavior that is **BEHAVIOR**.
- Relvar [PSEUDO]
  The character **CALLERID** knows the charactert **CALLEID** by the pseudonym **PSEUDONYME**.

About time :

- Relvar [BIRTH]
  The character **CHARACTERID** was born on **BIRTH**.
- Relvar [DEATH]
  The character **CHARACTERID** died on **DEATH**.

About Associations :

- Relvar [ASSOCIATIONNAME]
  The association **ASSOCIATIONID** is named **NAME**.
- Relvar [ASSOCIATIONDESCRIPTION]
  The association **ASSOCIATIONID** is described as **DESCRIPTION**.
- Relvar [ASSOCIATION]
  The character **CHARACTERID** is a member of the association **ASSOCIATIONID**.

About character relations :

- Relvar [RELATIONLIST]
  The relation **RELATIONID** is of the type **RELATIONSHIP**.
- Relvar [TIMELESSRELATION]
  The character **SOURCE** has a timeless relation with **TARGET** of the type **RELATIONID**.
- Relvar [DATERELATION]
  The character **SOURCE** started a permanent relation with **TARGET** of the type **RELATIONID** at the time **DATE**.

- Relvar [RANGERELATION]
  The character **SOURCE** had a relation with **TARGET** of the type **RELATIONID** that started on **START** and ended on **ENDDATE**.

About places :

- Relvar [PLACE]
  The place **PLACEID** is called **PLACENAME**.
- Relvar [MAP]
  The map **MAPID** has a width of **WIDTH** split into **NUMWIDTH** sections, and a length **LENGTH** split into **NUMLENGTH** sections.
- Relvar [MAPPEDPLACE]
  The place **PLACEID** is represented by the map **MAPID**.
- Relvar [SUBPLACE]
  The place **PLACEID** is locate on the square **SQUAREID** of the **MAPID** map.
- Relvar [ORIGINATES]
  The character **CHARACTERID** originates from **PLACEID**.

About events :

- Relvar [EVENTNAME]
  The event **EVENTID** is named **NAME**.
- Relvar [EVENTDESCRIPTION]
  The event **EVENTID** is described as **DESCRIPTION**.
- Relvar [ATTENDS]
  The character **CHARACTERID** attended **EVENTID**.
- Relvar [EVENT]
  The event **EVENTID** happened at **PLACEID**, started on **BEGINNING**, and ended on **ENDDATE**.

8 General Architecture

Our program uses an Model, View, Controller (MVC) template.

8.1 Models

The model files are classes used to have an in-memory representation of the DB data. Most of our models extend the DBModel abstract class which contains all the mandatory information for every model type, (most notably the ID) and allows certain methods to apply to any type of Model. The exceptions are CharacterPseudoData and RelationData, which do not use IDs. Our current models are :

- AssociationModel (non implemented)
- CharacterModel
- CharacterPseudoData
- MapModel
- PlaceModel

– RelationData

What these classes represent is fairly obvious. It is important to note that the models are strongly connected. Indeed, an event will contain information about the characters that participated. These characters will in turn contain information about the other events they attended. From there, it is not difficult to imagine scenarios where every model is somehow related to every other one, resulting in the whole database being loaded into memory when accessing a single character. In order to avoid this, a number of models do not contain instances of other models, but references to them (in this case, the String representation of the database ID). For example, a character model contains a LinkedList of the IDs of the events he attended. It is also usefull to note that EditionTableModel is not a model in this context, despite its name. Indeed, it is an implementation of the TableModel interface which is used for building the GUI.

8.2 Views

The View classes (named Windows in our program) dictate how the models are represented in the GUI. All our Windows are aimed at creating or editing a model while viewing it at the same time. To facilitate this, they extend an abstract EditionWindow class which provides the default methods for building the GUI, as well as the facilities to communicate with the Controller The one exception is MainMenu, which of course does not manipulate any data. The implemented Views are :

– CharacterEditionWindow
– EventEditionWindow
– MainMenu
– MapChoiceWindow
– MapCreateWindow
– PlaceEditionWindow
– RelationEditionTable

In almost every case, the same view is used to both create and update/view a model. The exception is for the map. Indeed, viewing a map involves displaying the Events and subplaces on it, while creating it only involves collecting its dimensions.

8.3 Controller

The controller is a single class that acts as a bridge between the Models and Views. Its tasks are to give the Views access to the data from the Models to fill the GUI with the relevant information. It is also in charge of controlling what happens when one View is closed. Depending on what happened, it will launch the next View and/or perform the corresponding database manipulations though not directly, as we will explain in the next part.

8.4 Other files

Some files do not belong strictly to the MVC pattern but are used for convenience. The most important of them is the DatabaseCoordinator. This class provides a layer between the Database and the MVC. The main utility of this class is to group all the SQL queries into a single class, in order to abstract away from the SQL language in the rest of the program.

The other files are :

- NarrationManager : acts as Main class, starts the Controller
- NarrationDate : simple class to represent a date without our calendar's restrictions
- EditionTableModel : implementation of java AbstractTableModel (which implements TableModel), used for tabular representations in a GUI as a model for JTable.
- EditionTable : an extension of JTable to provide common functionalities between its different extensions.
- EditionTablePanel : extension of JPanel to contain an EditionTable and add default buttons (apply and cancel).
- EventEditionTable/RelationEditionTable : extensions of EditionTable to provide a table layout of the corresponding model.

9 SQL script

In the zip file, you can find files `structure.sql` which builds the entire structure of the database in SQL. This script is idempotent, and is build with the same structure as the tutorial-D script. Indeed, we first create all the tables with the primary keys/unique keys, then we alter them to add the foreign key constraints.

The file `data.sql` is also available in the zip file. This one is also idempotent. It builds the structure of the database as the previous file, then it fills the database with all the elementary facts presented above in the report.

10 What we did

In this section, we explain which functionalities we managed to implement before the project's deadline. As the requirements set is pretty large, we had to choose among them which ones were more important in order to get a working program as soon and efficiently as possible. For clarity purposes, we group them here by concerned database model :

10.1 Events

Let us begin with events. Events can be created using the main menu's "Create new event" button, which opens a new edition window (actually, a JDialog, so it blocks the calling window until it has been closed). In that window, we can see that several default

data have already been put into the required fields, and of course that it is possible to modify them. To modify an element in the table, simply double-click on it, and then press enter to validate your caption. Note that in the case of date fields, the user cannot enter other values than string in the form year-month-day (the programs will refuse other input formats and surround the corresponding cell in red). These dates do however not need to represent "real-world dates", as it is possible to enter any number for the day, month, or year. We did this choice to better stick to the requirements of the program telling that the user should be able to model as many imaginary universes as possible in the program (which may imply the use of a non-Gregorian calendar). Once you have done all the modifications you needed, simply click the "Apply" button to save them into database and close the window. Clicking "Cancel" would also close the window, but would discard your changes, so that nothing is written in database. Events can also be edited with the same kind of menu (actually, the same class is called), using the "Edit event" option in the main menu, and then selecting the event to edit into the lists it displays. If no event is available to be edited in the database, an error message will be displayed to the user to kindly inform him that he should first create an event. Links can also be created between characters and events, as we will explain in the characters –related point.

Relations

Relations can be created or edited in order to link characters. Their creation and edition is entirely managed in the menus used for characters, so we will explain them in the "Characters" section.

10.2 Characters

Characters can be created or edited in the same manner than events. The characters edition window contains two tabs, one containing the basic data about the character and his relations with other characters in a table, the other containing a table with all the events linked to the character. These two tables give the possibility to add or remove elements to them using the "Insert…" and "Remove selected lines" buttons lying below them. The relations table displays both relations targeting the opened character and the relations from which this character is the source. To achieve this, it simply appends the tag " [INVERSE]" (with a space before the '[') to the type of the relation when the character is the target of the relation. Note that the relations table allows editing table cells in each of its columns, except the one containing the target name (as this is more new relation creation than edition in that case). This table also allows sorting relations by type or character name with the JComboBox below it. The second type lists the events linked to the character being edited. It is not editable (to edit an event, one needs to open the dedicated window through the main menu), but allows adding or removing relations, and thus modifying the "peerings" between events and characters.

11 Technical specifications

11.1 About the database

For this project, we use PostgreSQL 9.3.4. It can be downloaded here : `http://www.enterprisedb.com/products-services-training/pgdownload#windows` In order to get the project working, you need to install it, and then create a new database with it on localhost (on the default PostgreSQL port number), naming it lingi2172, with password "ben" for user "postgres".

11.2 About the code

Our project makes use of the latest Java version, that is, 1.8.0_05. Make sure you have it (or a later version) installed to use our software, as we used some of the new constructs it defines (as lambda functions). Java 8 is available here on Oracle's website : `http://www.oracle.com/technetwork/java/javase/downloads/index-jsp-138363.html`

We connect the database to Java using JDBC and directly coding the requests with it (in DatabaseCoordinator.java). To be able to use PostgreSQL with JDBC, you need a specific database driver, defined in the .jar file available here : `http://jdbc.postgresql.org/download.html` (make sure you choose a driver compatible with Java 8). This file has been included in the submitted archive for convenience purposes, so you are not obliged to download it. The section about the code philosophy will explain you what to do with it.

11.3 Code philosophy

We did not use any IDE for this project, only text editor and command line. This choice was first proposed by one of our team members (a jEdit-fundamentalist who strongly believes that coding without IDE leads to better code quality and higher mastery of the overall language and has been programming this way in Java since the very beginning of his studies). The two other ones saw there an opportunity to try something different and decided to follow him. However, even our fundamentalist (actually, especially him) must admit that finding how to run Java in the command line, despite the great wisdom of Internet, sometimes may proof pretty uneasy (and it is a pity). For this reason, here is a small guide explaining how to compile and execute our program, so that you don't lose time searching if you don't already know how to do :

11.3.1 First steps

- Open a terminal into the repository containing the source code
- Create a new directory where the program will be compiled. We will refer to it as "cpld" (stands for "compiled"), but you can of course give any name you like.
- Copy the database driver obtained as stated in the section about Java into your cpld directory.

– Check you java and javac installation, using both “java –verion” and “javac –version” commands. Both should return that you are using Java 1.8. If it is not the case, check your environment variables.

11.3.2 Compilation

In the directory containing the code, type :

```
java -d cpld *.java
```

This command compiles everything, creating the package directories into the cpld directory

11.3.3 Execution

Go to your cpld directory and type :

```
java -cp .;<name_of_driver_jar>  narrationmanager.NarrationManager
```

Note that this command seems to be different on Mac : you must then replace the ‘;’ by a ‘ :’. We did not test it on other UNIX systems, but it is possible that they also require this modification.

`<name_of_driver_jar>` must of course be replaced by the name of the database driver file (actually, you can also place it in another repository and give the path to it here).

-cp specifies the classpath to use for the JVM

Narrationmanager.NarrationManager is the full package path toward the class containing the main method.

11.3.4 Javadoc

– The code contains few comments, but each class has a brief Javadoc description that can be nicer to see browsing a generated Javadoc. This point explains how to do this
– In your source code repository, create a new directory, say “doc”
– Type the following command (as previously, replacing doc by the name of your doc repository) :
`javadoc -d doc *.java`
(don’t worry about all the stuff this command prints in the terminal ; for once, if means everything is going fine)
– The Javadoc is now in the doc repository !

12 Implementation issues

Due to various factors, we did not achieve all the things we would have wanted to do. In this section we will go the main features would have liked to add to the program.

12.1 DB coherence

Our current implementation uses queries directly through the JDBC driver. However, certain manipulations of the database require queries to multiple tables. I the event that one of the queries is invalid (wrong parameters, system crash ...), the others are not stopped and no recovery system is used. Additionaly, the users is not warned when one or more queries fail. This can result in situations where data might not have been added to the DB while the user thinks it is, or where some of the data is written in the DB while the user thinks the whole modification was dropped. Given more time, we would have wished to add mechanisms to warn the user in the event that he provided an invalid input before executing the query. Additionaly, we would have modified the project to use transactions to add data to the DB in a safe and reliable way.

12.2 Design oversights

When the moment came for us to gather statistics for the user, we realized we had forgottent a simple but important field for characters : their gender. This had the unfortunate consequence of making one of our big objectives (giving the user a quick overview of information over the whole text) a lot more complicated. We did consider some workaround solutions, such as representing gender by a self to self "Relation" (character relation, not DB relation), but this would cause gender statistics to be indistinguishable from other relations.

12.3 Missing features

Many features were not implemented due to the size of the project and the time given. As mentioned in the previous paragraph, statistics on gender were made difficult because of an oversight, and the whole feature ended up being dropped. We also did not have time to create a satisfactory Place viewer like those for Characters and Events. Ultimately, this viewer would have contained an interactive map representation of the place. This map would would show the Events and Subplaces of the Place, and would change depending on the time. Another feature that was not implemented is the handling of Character behaviour. This is simply another characteristic of a character, on which statistics would also have been collected. Similarly, Associations were not implemented beyond creating the empty model file.

Annexes

A Code Rel

```
TYPE NAME  POSSREP {NAME CHAR};
TYPE CHARACTER#  POSSREP {CHARACTERNUM CHAR};
TYPE EVENT#  POSSREP {EVENTNUM CHAR};
TYPE MAP#  POSSREP {MAPNUM CHAR};
TYPE PLACE# POSSREP {PLACENUM CHAR};
TYPE ASSOCIATION#  POSSREP {ASSOCIATIONNUM CHAR};
TYPE RELATION#  POSSREP {RELATIONNUM CHAR};
TYPE ENHANCEDDATE  POSSREP {YEAR INTEGER, MONTH INTEGER, DAY INTEGER};

OPERATOR DATE_CMP (START ENHANCEDDATE,ENDDATE ENHANCEDDATE) RETURNS INTEGER
    ;
IF THE_YEAR(START) > THE_YEAR(ENDDATE) THEN RETURN 1;
ELSE IF THE_YEAR(START) < THE_YEAR(ENDDATE) THEN RETURN -1;
ELSE IF THE_MONTH(START) > THE_MONTH(ENDDATE) THEN RETURN 1;
ELSE IF THE_MONTH(START) < THE_MONTH(ENDDATE) THEN RETURN -1;
ELSE IF THE_DAY(START) > THE_DAY(ENDDATE) THEN RETURN 1;
ELSE IF THE_DAY(START) < THE_DAY(ENDDATE) THEN RETURN -1;
ELSE RETURN 0;
END IF;
END IF;
END IF;
END IF;
END IF;
END IF;
END OPERATOR;


VAR CHARACTER  BASE RELATION {CHARACTERID CHARACTER#, NAME NAME} KEY {
    CHARACTERID};

VAR ASSOCIATION BASE RELATION {CHARACTERID CHARACTER#, ASSOCIATIONID
    ASSOCIATION#} KEY {CHARACTERID , ASSOCIATIONID};
VAR ORIGINATES  BASE RELATION {CHARACTERID CHARACTER#, PLACEID PLACE#} KEY
    {CHARACTERID};
VAR BEHAVIOR  BASE RELATION {CHARACTERID CHARACTER#, BEHAVIOR CHAR} KEY {
    CHARACTERID, BEHAVIOR};
VAR PSEUDO  BASE RELATION {CALLERID CHARACTER#, CALLEDID CHARACTER#,
    PSEUDONYME NAME} KEY {CALLERID, CALLEDID, PSEUDONYME};

VAR TIMELESSRELATION  BASE RELATION {SOURCE CHARACTER#, TARGET CHARACTER
    #, RELATIONID RELATION#} KEY {SOURCE, TARGET, RELATIONID};
VAR DATERELATION  BASE RELATION {SOURCE CHARACTER#, TARGET CHARACTER#,
    RELATIONID RELATION#, DATE ENHANCEDDATE} KEY {SOURCE, TARGET, RELATIONID
    , DATE};
VAR RANGERELATION  BASE RELATION {SOURCE CHARACTER#, TARGET CHARACTER#,
    RELATIONID RELATION#, START ENHANCEDDATE, ENDDATE ENHANCEDDATE} KEY {
    SOURCE, TARGET, RELATIONID, START, ENDDATE};


VAR BIRTH  BASE RELATION {CHARACTERID CHARACTER#, BIRTH ENHANCEDDATE} KEY
    {CHARACTERID};
VAR DEATH  BASE RELATION {CHARACTERID CHARACTER#, DEATH ENHANCEDDATE} KEY
    {CHARACTERID};

```

```
42 VAR ATTENDS BASE RELATION {CHARACTERID CHARACTER#, EVENTID EVENT#} KEY {
       CHARACTERID, EVENTID};
43
44 VAR RELATIONLIST      BASE RELATION {RELATIONID RELATION#, RELATIONTYPE CHAR}
       KEY {RELATIONID};
45
46 VAR EVENTNAME    BASE RELATION {EVENTID EVENT#, NAME CHAR} KEY {EVENTID};
47 VAR EVENTDESCRIPTION      BASE RELATION {EVENTID EVENT#, DESCRIPTION CHAR}
       KEY {EVENTID};
48
49 VAR ASSOCIATIONNAME BASE RELATION {ASSOCIATIONID ASSOCIATION#, NAME CHAR}
       KEY {ASSOCIATIONID};
50 VAR ASSOCIATIONDESCRIPTION  BASE RELATION {ASSOCIATIONID ASSOCIATION#,
       DESCRIPTION CHAR} KEY {ASSOCIATIONID};
51
52
53 VAR MAP       BASE RELATION {MAPID MAP#, NUMWIDTH INTEGER, NUMLENGTH INTEGER,
       WIDTH RATIONAL, LENGTH RATIONAL} KEY {MAPID};
54
55 VAR PLACE    BASE RELATION {PLACEID PLACE#, PLACENAME NAME} KEY {PLACEID};
56
57
58 VAR SUBPLACE      BASE RELATION {PLACEID PLACE#, SQUAREID INTEGER, MAPID MAP
       #} KEY {PLACEID, SQUAREID, MAPID};
59
60
61 VAR MAPPEDPLACE BASE RELATION {PLACEID PLACE#, MAPID MAP#} KEY {PLACEID}
       KEY {MAPID};
62
63 VAR EVENT    BASE RELATION {EVENTID EVENT#, PLACEID PLACE#, BEGINNING
       ENHANCEDDATE, ENDDATE ENHANCEDDATE} KEY {EVENTID, PLACEID, BEGINNING,
       ENDDATE};
64
65
66
67
68 CONSTRAINT C1 ASSOCIATION {CHARACTERID} <= CHARACTER {CHARACTERID};
69
70 CONSTRAINT C2 ORIGINATES {CHARACTERID} <= CHARACTER {CHARACTERID};
71 CONSTRAINT C3 BEHAVIOR {CHARACTERID} <= CHARACTER {CHARACTERID};
72 CONSTRAINT C4 (PSEUDO RENAME {CALLERID AS CHARACTERID}){CHARACTERID} <=
       CHARACTER {CHARACTERID};
73 CONSTRAINT C5 (PSEUDO RENAME {CALLEDID AS CHARACTERID}){CHARACTERID} <=
       CHARACTER {CHARACTERID};
74
75 CONSTRAINT C6 (TIMELESSRELATION RENAME {SOURCE AS CHARACTERID}) {
       CHARACTERID} <= CHARACTER {CHARACTERID};
76 CONSTRAINT C7 (DATERELATION RENAME {SOURCE AS CHARACTERID}) {CHARACTERID}
       <= CHARACTER {CHARACTERID};
77 CONSTRAINT C8 (RANGERELATION RENAME {SOURCE AS CHARACTERID}) {CHARACTERID}
       <= CHARACTER {CHARACTERID};
78
79 CONSTRAINT C9 (TIMELESSRELATION RENAME {TARGET AS CHARACTERID}){CHARACTERID
       } <= CHARACTER {CHARACTERID};
80 CONSTRAINT C10 (DATERELATION RENAME {TARGET AS CHARACTERID}){CHARACTERID}
       <= CHARACTER {CHARACTERID};
```

```
81 CONSTRAINT C11 (RANGERELATION RENAME {TARGET AS CHARACTERID}){CHARACTERID}
       <= CHARACTER {CHARACTERID};
82
83 CONSTRAINT C12 BIRTH {CHARACTERID} <= CHARACTER {CHARACTERID};
84 CONSTRAINT C13 DEATH {CHARACTERID} <= CHARACTER {CHARACTERID};
85
86 CONSTRAINT C14 ATTENDS {CHARACTERID} <= CHARACTER {CHARACTERID};
87
88
89
90
91 CONSTRAINT C15 ASSOCIATION {ASSOCIATIONID} <= ASSOCIATIONNAME {
       ASSOCIATIONID};
92 CONSTRAINT C16 ASSOCIATIONDESCRIPTION {ASSOCIATIONID} <= ASSOCIATIONNAME {
       ASSOCIATIONID};
93
94
95 CONSTRAINT C17 ORIGINATES {PLACEID} <= PLACE {PLACEID};
96 CONSTRAINT C18 SUBPLACE {PLACEID} <= PLACE {PLACEID};
97 CONSTRAINT C19 MAPPEDPLACE {PLACEID} <= PLACE {PLACEID};
98 CONSTRAINT C20 EVENT {PLACEID} <= PLACE {PLACEID};
99
100
101 CONSTRAINT C21 TIMELESSRELATION {RELATIONID} <= RELATIONLIST {RELATIONID};
102 CONSTRAINT C22 DATERELATION {RELATIONID} <= RELATIONLIST {RELATIONID};
103 CONSTRAINT C23 RANGERELATION {RELATIONID} <= RELATIONLIST {RELATIONID};
104
105
106 CONSTRAINT C31 ATTENDS {EVENTID} <= EVENTNAME {EVENTID};
107 CONSTRAINT C32 EVENTDESCRIPTION {EVENTID} <= EVENTNAME {EVENTID};
108 CONSTRAINT C33 EVENT {EVENTID} <= EVENTNAME {EVENTID};
109
110 CONSTRAINT C34 MAPPEDPLACE {MAPID} <= MAP {MAPID};
111 CONSTRAINT C35 SUBPLACE {MAPID} <= MAP {MAPID};
112
113 CONSTRAINT C36 DATE_CMP (RANGERELATION {START}, RANGERELATION {ENDDATE}) <=
       0;
114 CONSTRAINT C37 DATE_CMP (EVENT {BEGINNING}, EVENT {ENDDATE}) <= 0;
115
116
117
118
119 CONSTRAINT C36 DATE_CMP(ENHANCEDDATE(2014,4,4), ENHANCEDDATE(2015,4,4)) <=
       0;
120
121 CONSTRAINT C37 DATE_CMP(RANGERELATION{START}, RANGERELATION{ENDDATE}) <= 0;
```

structure_rel.d

B Code SQL

```sql
DROP TABLE IF EXISTS CHARACTER CASCADE;
DROP TABLE IF EXISTS ASSOCIATION CASCADE;
DROP TABLE IF EXISTS ORIGINATES CASCADE;
DROP TABLE IF EXISTS BEHAVIOR CASCADE;
DROP TABLE IF EXISTS PSEUDO CASCADE;
DROP TABLE IF EXISTS TIMELESSRELATION CASCADE;
DROP TABLE IF EXISTS DATERELATION CASCADE;
DROP TABLE IF EXISTS RANGERELATION CASCADE;
DROP TABLE IF EXISTS BIRTH CASCADE;
DROP TABLE IF EXISTS DEATH CASCADE;
DROP TABLE IF EXISTS ATTENDS CASCADE;
DROP TABLE IF EXISTS RELATIONLIST CASCADE;
DROP TABLE IF EXISTS EVENTNAME CASCADE;
DROP TABLE IF EXISTS EVENTDESCRIPTION CASCADE;
DROP TABLE IF EXISTS ASSOCIATIONNAME CASCADE;
DROP TABLE IF EXISTS ASSOCIATIONDESCRIPTION CASCADE;
DROP TABLE IF EXISTS MAP CASCADE;
DROP TABLE IF EXISTS PLACE CASCADE;
DROP TABLE IF EXISTS SUBPLACE CASCADE;
DROP TABLE IF EXISTS SQUAREID CASCADE;
DROP TABLE IF EXISTS MAPPEDPLACE CASCADE;
DROP TABLE IF EXISTS EVENT CASCADE;
DROP TYPE IF EXISTS ENHANCEDDATE;

CREATE TYPE ENHANCEDDATE as (YEAR INTEGER, MONTH INTEGER, DAY INTEGER);

CREATE TABLE CHARACTER(
  CHARACTERID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
  NAME TEXT NOT NULL
);

CREATE TABLE ASSOCIATION(
  CHARACTERID CHARACTER VARYING(10) NOT NULL,
  ASSOCIATIONID CHARACTER VARYING(10) NOT NULL,
  PRIMARY KEY (CHARACTERID,ASSOCIATIONID)
);

CREATE TABLE ORIGINATES(
  CHARACTERID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
  PLACEID CHARACTER VARYING(10) NOT NULL
);

CREATE TABLE BEHAVIOR(
  CHARACTERID CHARACTER VARYING(10) NOT NULL,
  BEHAVIOR CHARACTER VARYING(42) NOT NULL,
  PRIMARY KEY (CHARACTERID,BEHAVIOR)
);

CREATE TABLE PSEUDO(
  CALLERID CHARACTER VARYING(10) NOT NULL,
  CALLEDID CHARACTER VARYING(10) NOT NULL,
  PSEUDONYME CHARACTER VARYING(42) NOT NULL,
  PRIMARY KEY (CALLERID,CALLEDID,PSEUDONYME)
```

```
);

CREATE TABLE TIMELESSRELATION(
   SOURCE CHARACTER VARYING(10) NOT NULL,
   TARGET CHARACTER VARYING(10) NOT NULL,
   RELATIONID CHARACTER VARYING(10) NOT NULL,
   PRIMARY KEY (SOURCE,TARGET,RELATIONID)
);

CREATE TABLE DATERELATION(
   SOURCE CHARACTER VARYING(10) NOT NULL,
   TARGET CHARACTER VARYING(10) NOT NULL,
   RELATIONID CHARACTER VARYING(10) NOT NULL,
   DATE ENHANCEDDATE NOT NULL,
   PRIMARY KEY (SOURCE,TARGET,RELATIONID,DATE)
);

CREATE TABLE RANGERELATION(
   SOURCE CHARACTER VARYING(10) NOT NULL,
   TARGET CHARACTER VARYING(10) NOT NULL,
   RELATIONID CHARACTER VARYING(10) NOT NULL,
   START ENHANCEDDATE NOT NULL,
   ENDDATE ENHANCEDDATE NOT NULL,
   PRIMARY KEY (SOURCE,TARGET,RELATIONID,START,ENDDATE)
);

CREATE TABLE BIRTH(
   CHARACTERID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
   BIRTH ENHANCEDDATE NOT NULL
);

CREATE TABLE DEATH(
   CHARACTERID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
   DEATH ENHANCEDDATE NOT NULL
);

CREATE TABLE ATTENDS(
   CHARACTERID CHARACTER VARYING(10) NOT NULL,
   EVENTID CHARACTER VARYING(10) NOT NULL,
   PRIMARY KEY (CHARACTERID,EVENTID)
);

CREATE TABLE RELATIONLIST(
   RELATIONID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
   RELATIONTYPE CHARACTER VARYING(42) NOT NULL
);

CREATE TABLE EVENTNAME(
   EVENTID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
   NAME CHARACTER VARYING(42) NOT NULL
);

CREATE TABLE EVENTDESCRIPTION(
   EVENTID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
   DESCRIPTION CHARACTER VARYING(42) NOT NULL
);
```

```
110
111 CREATE TABLE ASSOCIATIONNAME(
112    ASSOCIATIONID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
113    NAME CHARACTER VARYING(42) NOT NULL
114 );
115
116 CREATE TABLE ASSOCIATIONDESCRIPTION(
117    ASSOCIATIONID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
118    DESCRIPTION TEXT NOT NULL
119 );
120
121 CREATE TABLE MAP(
122    MAPID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
123    NUMWIDTH INTEGER NOT NULL,
124    NUMLENGTH INTEGER NOT NULL,
125    WIDTH DOUBLE PRECISION NOT NULL,
126    LENGTH DOUBLE PRECISION NOT NULL
127 );
128
129 CREATE TABLE PLACE(
130    PLACEID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
131    PLACENAME CHARACTER VARYING(42) NOT NULL
132 );
133
134 CREATE TABLE SUBPLACE(
135    PLACEID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
136    SQUAREID INTEGER NOT NULL,
137    MAPID CHARACTER VARYING(10) NOT NULL
138 );
139
140 CREATE TABLE SQUAREID(
141    PLACEID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
142    SQUAREID CHARACTER VARYING(10) NOT NULL
143 );
144
145 CREATE TABLE MAPPEDPLACE(
146    PLACEID CHARACTER VARYING(10) PRIMARY KEY NOT NULL,
147    MAPID CHARACTER VARYING(10) UNIQUE NOT NULL
148 );
149
150 CREATE TABLE EVENT(
151    EVENTID CHARACTER VARYING(10) NOT NULL,
152    PLACEID CHARACTER VARYING(10) NOT NULL,
153    BEGINNING ENHANCEDDATE NOT NULL,
154    ENDDATE ENHANCEDDATE NOT NULL,
155    PRIMARY KEY (EVENTID,PLACEID,BEGINNING,ENDDATE)
156 );
157
158 ALTER TABLE ONLY ASSOCIATION
159      ADD CONSTRAINT C1 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
           CHARACTERID);
160
161 ALTER TABLE ONLY ORIGINATES
162      ADD CONSTRAINT C2 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
           CHARACTERID);
163
```

```
164 ALTER TABLE ONLY BEHAVIOR
165     ADD CONSTRAINT C3 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
        CHARACTERID);
166
167 ALTER TABLE ONLY PSEUDO
168     ADD CONSTRAINT C4 FOREIGN KEY (CALLERID) REFERENCES CHARACTER(
        CHARACTERID);
169
170 ALTER TABLE ONLY PSEUDO
171     ADD CONSTRAINT C5 FOREIGN KEY (CALLEDID) REFERENCES CHARACTER(
        CHARACTERID);
172
173 ALTER TABLE ONLY TIMELESSRELATION
174     ADD CONSTRAINT C6 FOREIGN KEY (SOURCE) REFERENCES CHARACTER(CHARACTERID
        );
175
176 ALTER TABLE ONLY DATERELATION
177     ADD CONSTRAINT C7 FOREIGN KEY (SOURCE) REFERENCES CHARACTER(CHARACTERID
        );
178
179 ALTER TABLE ONLY RANGERELATION
180     ADD CONSTRAINT C8 FOREIGN KEY (SOURCE) REFERENCES CHARACTER(CHARACTERID
        );
181
182 ALTER TABLE ONLY TIMELESSRELATION
183     ADD CONSTRAINT C9 FOREIGN KEY (TARGET) REFERENCES CHARACTER(CHARACTERID
        );
184
185 ALTER TABLE ONLY DATERELATION
186     ADD CONSTRAINT C10 FOREIGN KEY (TARGET) REFERENCES CHARACTER(
        CHARACTERID);
187
188 ALTER TABLE ONLY RANGERELATION
189     ADD CONSTRAINT C11 FOREIGN KEY (TARGET) REFERENCES CHARACTER(
        CHARACTERID);
190
191 ALTER TABLE ONLY BIRTH
192     ADD CONSTRAINT C12 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
        CHARACTERID);
193
194 ALTER TABLE ONLY DEATH
195     ADD CONSTRAINT C13 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
        CHARACTERID);
196
197 ALTER TABLE ONLY ATTENDS
198     ADD CONSTRAINT C14 FOREIGN KEY (CHARACTERID) REFERENCES CHARACTER(
        CHARACTERID);
199
200 ALTER TABLE ONLY ASSOCIATION
201     ADD CONSTRAINT C15 FOREIGN KEY (ASSOCIATIONID) REFERENCES
        ASSOCIATIONNAME(ASSOCIATIONID);
202
203 ALTER TABLE ONLY ASSOCIATIONDESCRIPTION
204     ADD CONSTRAINT C16 FOREIGN KEY (ASSOCIATIONID) REFERENCES
        ASSOCIATIONNAME(ASSOCIATIONID);
205
```

```
206 ALTER TABLE ONLY ORIGINATES
207     ADD CONSTRAINT C17 FOREIGN KEY (PLACEID) REFERENCES PLACE(PLACEID);
208
209 ALTER TABLE ONLY SUBPLACE
210     ADD CONSTRAINT C18 FOREIGN KEY (PLACEID) REFERENCES PLACE(PLACEID);
211
212 ALTER TABLE ONLY MAPPEDPLACE
213     ADD CONSTRAINT C19 FOREIGN KEY (PLACEID) REFERENCES PLACE(PLACEID);
214
215 ALTER TABLE ONLY EVENT
216     ADD CONSTRAINT C20 FOREIGN KEY (PLACEID) REFERENCES PLACE(PLACEID);
217
218 ALTER TABLE ONLY TIMELESSRELATION
219     ADD CONSTRAINT C21 FOREIGN KEY (RELATIONID) REFERENCES RELATIONLIST(
        RELATIONID);
220
221 ALTER TABLE ONLY DATERELATION
222     ADD CONSTRAINT C22 FOREIGN KEY (RELATIONID) REFERENCES RELATIONLIST(
        RELATIONID);
223
224 ALTER TABLE ONLY RANGERELATION
225     ADD CONSTRAINT C23 FOREIGN KEY (RELATIONID) REFERENCES RELATIONLIST(
        RELATIONID);
226
227 ALTER TABLE ONLY ATTENDS
228     ADD CONSTRAINT C31 FOREIGN KEY (EVENTID) REFERENCES EVENTNAME(EVENTID);
229
230 ALTER TABLE ONLY EVENTDESCRIPTION
231     ADD CONSTRAINT C32 FOREIGN KEY (EVENTID) REFERENCES EVENTNAME(EVENTID);
232
233 ALTER TABLE ONLY EVENT
234     ADD CONSTRAINT C33 FOREIGN KEY (EVENTID) REFERENCES EVENTNAME(EVENTID);
235
236 ALTER TABLE ONLY MAPPEDPLACE
237     ADD CONSTRAINT C34 FOREIGN KEY (MAPID) REFERENCES MAP(MAPID);
238
239 ALTER TABLE ONLY SUBPLACE
240     ADD CONSTRAINT C35 FOREIGN KEY (MAPID) REFERENCES MAP(MAPID);
```

structure.sql