Chapter 3
Extending the Database Data Model: Animals and Sensors

Ferdinando Urbano

Abstract GPS positions are used to describe animal movements and to derive a large set of information, for example, about animals' behaviour, social interactions and environmental preferences. GPS data are related to (and must be integrated with) many other sources of information that together can be used to describe the complexity of movement ecology. This can be achieved through proper database data modelling, which depends on a clear definition of the biological context of a study. Particularly, data modelling becomes a key step when database systems manage many connected data sets that grow in size and complexity: it permits easy updates of the database structure to accommodate the changing goals, constraints and spatial scales of studies. In this chapter's exercise, you will extend your database (see Chap. 2) with two new tables to integrate ancillary information useful to interpreting GPS data: one for GPS sensors and the other for animals.

Keywords Data modelling · GPS tracking · Data management · Spatial database

Introduction

GPS positions are used to describe animal movements and to derive a large set of information, for example, on animals' behaviour, social interactions and environmental preferences. GPS data are related to (and must be integrated with) many other information that together can be used to describe the complexity of movement ecology. This can be achieved through proper database data modelling. A data model describes what types of data are stored and how they are organised.

F. Urbano (✉)
Università Iuav di Venezia, Santa Croce 191 Tolentini, 30135 Venice, Italy
e-mail: ferdi.urbano@gmail.com

F. Urbano and F. Cagnacci (eds.), *Spatial Database for GPS Wildlife Tracking Data*,
DOI: 10.1007/978-3-319-03743-1_3, © Springer International Publishing Switzerland 2014

It can be seen as the conceptual representation of the real world in the database structures that include data objects (i.e. tables) and their mutual relationships. In particular, data modelling becomes a key step when database systems grow in size and complexity, and user requirements become more sophisticated: it permits easy updates of the database structure to accommodate the changing goals, constraints, and spatial scales of studies and the evolution of wildlife tracking systems. Without a rigorous data modelling approach, an information system might lose the flexibility to manage data efficiently in the long term, reducing its utility to a simple storage device for raw data, and thus failing to address many of the necessary requirements.

To model data properly, you have to clearly state the biological context of your study. A logical way to proceed is to define (1) very basic questions on the sample unit, i.e. individual animals and (2) basic questions about data collection.

1. Typically, individuals are the sampling units of an ecological study based on wildlife tracking. Therefore, the first question to be asked while modelling the data is: 'What basic biological information is needed to characterise individuals as part of a sample?' Species, sex and age (or age class) at capture are the main factors which are relevant in all studies. Age classes typically depend on the species[1]. Other information used to characterise individuals could be specific to a study, for example in a study on spatial behaviour of translocated animals, 'resident' or 'translocated' is an essential piece of information linked to individual animals. All these elements should be described in specific tables.
2. A single individual becomes a 'studied unit' when it is fitted with a sensor, in this case to collect position information. First of all, GPS sensors should be described by a dedicated table containing the technical characteristics of each device (e.g. vendor, model). Capture time, or 'device-fitting' time, together with the time and a description of the end of the deployment (e.g. drop-off of the tag, death of the animal), are also essential to the model. The link between the sensors and the animals should be described in a table that states unequivocally when the data collected from a sensor 'become' (and cease to be) bio-logged data, i.e. the period during which they refer to an individual's behaviour. The start of the deployment usually coincides with the moment of capture, but it is not the same thing. Indeed, moment of capture can be the 'end' of one relationship between a sensor and an animal (i.e. when a device is taken off an animal) and at the same time the 'beginning' of another (i.e. another device is fitted instead).

[1] Age class of an animal is not constant for all the GPS positions. The correct age class at any given moment can be derived from the age class at capture and by defining rules that specify when the individual changes from one class to another (for roe deer, you might assume that on 1st April of every year each individual that was a fawn becomes a yearling, and each yearling becomes an adult).

Thanks to the tables 'animals', 'sensors', and 'sensors to animals', and the relationships built among them, GPS data can be linked unequivocally to individuals, i.e. the sampling units.

Some information related to animals can change over time. Therefore, they must be marked with the reference time that they refer to. Examples of typical parameters assessed at capture are age and positivity of association to a disease. Translocation may also coincide with the capture/release time. If this information changes over time according to well-defined rules (e.g. transition from age classes), their value can be dynamically calculated in the database at different moments in time (e.g. using database functions). You will see an example of a function to calculate age class from the information on the age class at capture and the acquisition time of GPS positions for roe deer in Chap. 9.

The basic structure 'animals', 'sensors', 'sensors to animals', and, of course, 'position data', can be extended to take into account the specific goals of each project, the complexity of the real-world problems faced, the technical environment and the available data. Examples of data that can be integrated are capture methodology, handling procedure, use of tranquilizers and so forth, that should be described in a 'captures' table linked to the specific individual (in the table 'animals'). Finally, data referring to individuals may come from several sources, e.g. several sensors or visual observations. In all these cases, the link between data and sample units (individuals) should also be clearly stated by appropriate relationships.

At the moment, there is a single table in the test database that represents raw data from GPS sensors. Now, you can start including more information in new tables to represent other important elements involved in wildlife tracking. This process will continue throughout all the following chapters.

In this chapter's exercise, you will include two new tables: one for GPS sensors and one for animals, with some ancillary tables (age classes, species).

Import Information on GPS Sensors and Add Constraints to the Table

In the subfolder \tracking_db\data\animals and \tracking_db\data\sensors of the test data set[2], you will find two files: animals.csv and gps_sensors.csv. Let us start with data on GPS sensors. First, you have to create a table in the database with the same attributes as the .csv file and then import the data into it. Here is the code of the table structure:

[2] The file with the test data set trackingDB_datasets.zip is part of the Extra Material of the book available at http://extras.springer.com.

```
CREATE TABLE main.gps_sensors(
  gps_sensors_id integer,
  gps_sensors_code character varying NOT NULL,
  purchase_date date,
  frequency double precision,
  vendor character varying,
  model character varying,
  sim character varying,
  CONSTRAINT gps_sensors_pkey
    PRIMARY KEY (gps_sensors_id ),
  CONSTRAINT gps_sensor_code_unique
    UNIQUE (gps_sensors_code)
);
COMMENT ON TABLE main.gps_sensors
IS 'GPS sensors catalog.';
```

The only field that is not present in the original file is *gps_sensors_id*. This is an integer[3] used as primary key. You could also use *gps_sensors_code* as primary key, but in many practical situations it is handy to use an integer field.

You add a field to keep track of the timestamp of record insertion:

```
ALTER TABLE main.gps_sensors
  ADD COLUMN insert_timestamp timestamp with time zone DEFAULT now();
```

Now, you can import data using the *COPY* command:

```
COPY main.gps_sensors(
  gps_sensors_id, gps_sensors_code, purchase_date, frequency, vendor, model,
  sim)
FROM
  'C:\tracking_db\data\sensors\gps_sensors.csv'
  WITH (FORMAT csv, DELIMITER ';');
```

At this stage, you have defined the list of GPS sensors that exist in your database. To be sure that you will never have GPS data that come from a GPS sensor that does not exist in the database, you apply a foreign key[4] between *main.gps_data* and *main.gps_sensors*. Foreign keys physically translate the concept of relations among tables.

[3] In some cases, a good recommendation is to use a 'serial' number as primary key to let the database generate a unique code (integer) every time that a new record is inserted. In this exercise, we use an integer data type because the values of the *gps_sensors_id* field are defined in order to be correctly referenced in the exercises of the next chapters.

[4] http://www.postgresql.org/docs/9.2/static/tutorial-fk.html.

```
ALTER TABLE main.gps_data
  ADD CONSTRAINT gps_data_gps_sensors_fkey
  FOREIGN KEY (gps_sensors_code)
  REFERENCES main.gps_sensors (gps_sensors_code)
  MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION;
```

This setting says that in order to delete a record in *main.gps_sensors*, you first have to delete all the associated records in *main.gps_data*. From now on, before importing GPS data from a sensor, you have to create the sensor's record in the *main.gps_sensors* table.

You can add other kinds of constraints to control the consistency of your database. As an example, you check that the date of purchase is after 2000-01-01. If this condition is not met, the database will refuse to insert (or modify) the record and will return an error message.

```
ALTER TABLE main.gps_sensors
  ADD CONSTRAINT purchase_date_check
  CHECK (purchase_date > '2000-01-01'::date);
```

Import Information on Animals and Add Constraints to the Table

Now, you repeat the same process for data on animals. Analysing the animals' source file (animals.csv), you can derive the fields of the new *main.animals* table:

```
CREATE TABLE main.animals(
  animals_id integer,
  animals_code character varying(20) NOT NULL,
  name character varying(40),
  sex character(1),
  age_class_code integer,
  species_code integer,
  note character varying,
  CONSTRAINT animals_pkey PRIMARY KEY (animals_id)
);
COMMENT ON TABLE main.animals
IS 'Animals catalog with the main information on individuals.';
```

As for *main.gps_sensors*, in your operational database, you can use the *serial* data type for the *animals_id* field. Age class (at capture) and species are attributes that can only have defined values. To enforce consistency in the database, in these cases, you can use lookup tables. Lookup tables store the list and the description of all possible values referenced by specific fields in different tables and constitute the

definition of the valid domain. It is recommended to keep them in a separated schema to give the database a more readable and clear data structure. Therefore, you create a *lu_tables* schema:

```
CREATE SCHEMA lu_tables;
  GRANT USAGE ON SCHEMA lu_tables TO basic_user;
COMMENT ON SCHEMA lu_tables
IS 'Schema that stores look up tables.';
```

You set as default that the user *basic_user* will be able to run *SELECT* queries on all the tables that will be created in this schema:

```
ALTER DEFAULT PRIVILEGES
  IN SCHEMA lu_tables
  GRANT SELECT ON TABLES
  TO basic_user;
```

Now, you create a lookup table for species:

```
CREATE TABLE lu_tables.lu_species(
  species_code integer,
  species_description character varying,
  CONSTRAINT lu_species_pkey
  PRIMARY KEY (species_code)
);
COMMENT ON TABLE lu_tables.lu_species
IS 'Look up table for species.';
```

You populate it with some values (just roe deer code will be used in our test data set):

```
INSERT INTO lu_tables.lu_species
  VALUES (1, 'roe deer');
INSERT INTO lu_tables.lu_species
  VALUES (2, 'rein deer');
INSERT INTO lu_tables.lu_species
  VALUES (3, 'moose');
```

You can do the same for age classes:

```
CREATE TABLE lu_tables.lu_age_class(
  age_class_code integer,
  age_class_description character varying,
  CONSTRAINT lage_class_pkey
  PRIMARY KEY (age_class_code)
);
COMMENT ON TABLE lu_tables.lu_age_class
IS 'Look up table for age classes.';
```

You populate it with some values[5]:

```
INSERT INTO lu_tables.lu_age_class
  VALUES (1, 'fawn');
INSERT INTO lu_tables.lu_age_class
  VALUES (2, 'yearling');
INSERT INTO lu_tables.lu_age_class
  VALUES (3, 'adult');
```

At this stage, you can create the foreign keys between the *main.animals* table and the two lookup tables:

```
ALTER TABLE main.animals
  ADD CONSTRAINT animals_lu_species
  FOREIGN KEY (species_code)
  REFERENCES lu_tables.lu_species (species_code)
  MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION;
ALTER TABLE main.animals
  ADD CONSTRAINT animals_lu_age_class
  FOREIGN KEY (age_class_code)
  REFERENCES lu_tables.lu_age_class (age_class_code)
  MATCH SIMPLE ON UPDATE NO ACTION ON DELETE NO ACTION;
```

For sex class of deer, you do not expect to have more than the two possible values: female and male (stored in the database as ‘f’ and ‘m’ to simplify data input). In this case, instead of a lookup table you can set a check on the field:

```
ALTER TABLE main.animals
  ADD CONSTRAINT sex_check
  CHECK (sex = 'm' OR sex = 'f');
```

Whether it is better to use a lookup table or a check must be evaluated case by case, mainly according to the number of admitted values and the possibility that you will want to add new values in the future.

You should also add a field to keep track of the timestamp of record insertion:

```
ALTER TABLE main.animals
  ADD COLUMN insert_timestamp timestamp with time zone DEFAULT now();
```

[5] These categories are based on roe deer; other species might need a different approach.

As a last step, you import the values from the file:

```
COPY main.animals(
  animals_id,animals_code, name, sex, age_class_code, species_code)
FROM
  'C:\tracking_db\data\animals\animals.csv'
  WITH (FORMAT csv, DELIMITER ';');
```

To test the result, you can retrieve the animals' data with the extended species and age class description:

```
SELECT
  animals.animals_id AS id,
  animals.animals_code AS code,
  animals.name,
  animals.sex,
  lu_age_class.age_class_description AS age_class,
  lu_species.species_description AS species
FROM
  lu_tables.lu_age_class,
  lu_tables.lu_species,
  main.animals
WHERE
  lu_age_class.age_class_code = animals.age_class_code
  AND
  lu_species.species_code = animals.species_code;
```

The result of the query is:

```
 id | code |    name    | sex | age_class | species
----+------+------------+-----+-----------+----------
  1 | F09  | Daniela    | f   | adult     | roe deer
  2 | M03  | Agostino   | m   | adult     | roe deer
  3 | M06  | Sandro     | m   | adult     | roe deer
  4 | F10  | Alessandra | f   | adult     | roe deer
  5 | M10  | Decimo     | m   | adult     | roe deer
```

You can also create this query with the pgAdmin tool 'Graphical Query Builder' (Fig. 3.1).

First Elements of the Database Data Model

In Fig. 3.2, you have a schematic representation of the tables created so far in the database, and their relationships. As you can see, the table *animals* is linked with foreign keys to two tables in the schema where the lookup tables are stored. In fact, the tables *lu_species* and *lu_age_class* contain the admitted values for the related fields in the *animals* table. The table *gps_data*, which contains the raw data

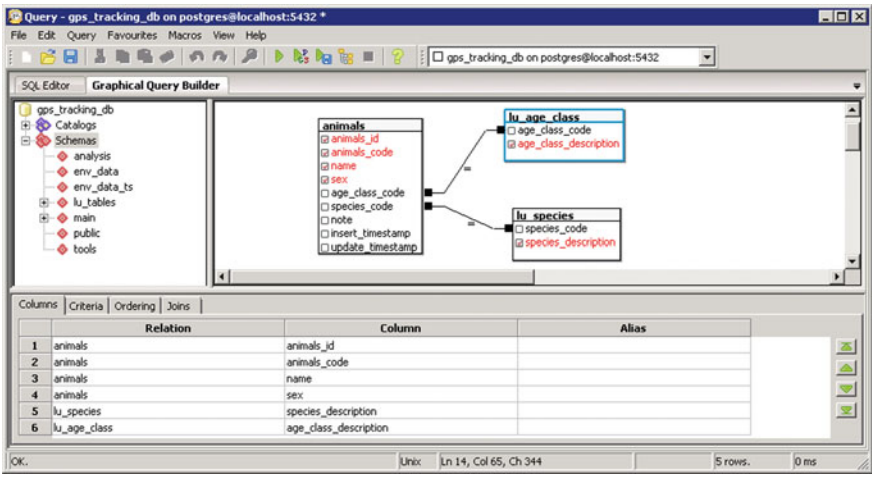

Fig. 3.1 pgAdmin GUI interface to create queries

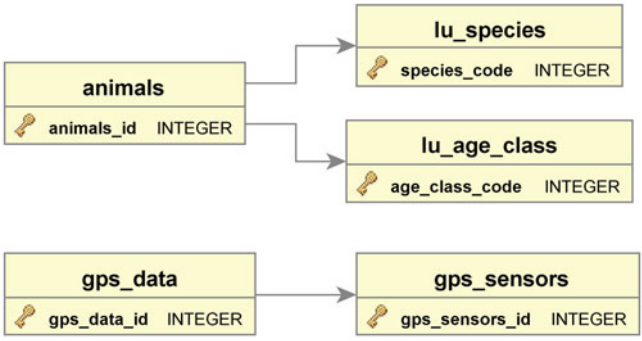

Fig. 3.2 Tables stored in the database at the end of the exercise for this chapter. The arrows identify links between tables connected by foreign keys

coming from GPS sensors, is linked to the table *gps_sensors*, where all the existing GPS sensors are stored with a set of ancillary information. A foreign key creates a dependency, for example, the table *gps_data* cannot store data from a GPS sensor if the sensor is not included in the *gps_sensors* table.

Note that, at this point, there is no relation between animals and the GPS data. In other words, it is impossible to retrieve positions of a given animal, but only of a given collar. Moreover, you cannot distinguish between GPS positions recorded when the sensors were deployed on the animals and those that were recorded for example, in the researcher's office before the deployment. You will see in the following chapter how to associate GPS positions with animals.