CHAPTER 4

Systematic data management

Cameron J. Nordell and Alastair Franke

Nordell, C. J., and A. Franke. 2017. Systematic data management. Pages 75–90 *in* D.L. Anderson, C.J.W. McClure, and A. Franke, editors. Applied raptor ecology: essentials from Gyrfalcon research. The Peregrine Fund, Boise, Idaho, USA.
https://doi.org/10.4080/are.2017/004

4.1 Introduction

4.1.1 Systematic data management for raptor biologists

Systematic data management has become essential for the collection, storage, and use of "big data" common to banking, healthcare, and insurance industries. However, projects of all sizes can benefit from structured data management. The collection and storage of information using appropriate organizational structure, data entry, and metadata promotes error free and efficient manipulation, reporting, analysis, sharing, and congruity over time (Léonard 1992). In this chapter we discuss from the perspective of raptor biology how to organize information so that it can be accessed, managed, and updated easily and without error (Ritchie 2002). Recent literature has highlighted the significant benefits accrued by wildlife research programs using systematic data management (Rüegg et al. 2014, Applegate 2015, Reynolds et al. 2016). Despite the practical benefits, systematic data management remains underused by wildlife researchers (Applegate 2015).

One barrier to the implementation of systematic data management is that there are few resources designed specifically to assist wildlife biologists, and virtually none for raptor researchers. Researchers developing a data management system can use Chamanara and König-Ries (2014), Briney (2015), and Reynolds et al. (2016) for examples of broad conceptual ideas for building and managing data. Further, in-depth resources are available

that discuss the challenges of running long-term and large-scale research programs (Applegate 2015, Hunt et al. 2015, Sólymos et al. 2015, and Sutter et al. 2015). However, the conceptual nature and comprehensive scope of these articles precludes them as a simple introduction to the topic. Our intention here, using Arctic raptor ecology as a model, is to provide a practical introduction to data management for raptor field research. We provide justification, instruction, and examples of systematic data management using real-world examples of raptor data. Most raptor research programs have similar data needs, including the identification of distinct nesting sites and territories, collection of reproductive data at nesting sites, and banding and re-sighting of individual raptors within and across years. These commonalities promote the creation of raptor-specific data management guidelines. Using basic principles of relational data-basing (Ritchie 2002) we provide specific examples that raptor researchers can replicate directly, while also illustrating data management concepts that are flexible enough to be implemented according to project-specific constraints. Researchers interested in using the analyses in subsequent chapters will benefit from systematic data management practices.

4.2 Relational databases

The most common conceptual framework to manage data is the relational database (Ritchie 2002), a structured data format that permits attribution of formal, rule-based relationships among discrete sets of data. Database Management Systems (DBMS) are typically software programs that can be used to create and use a database. This chapter does not endorse any particular DBMS, but we will use Microsoft Access 2016 to illustrate how to build and use a database (see GYRF Growth.accdb in Chapter 4 online materials for examples), as it is widely available, user friendly, and well supported. Based on project needs and prior experience, readers may prefer different platforms to manage their databases, but the underlying principles of data management do not change. The structured query language (SQL) code provided to demonstrate the manipulation of data in relational tables should be compatible across platforms. Additional information regarding SQL is beyond the scope of this chapter, but is widely available.

4.2.1 What is a relational database

To avoid confusion, we use the following nomenclature: **Bold** text will be used to identify first use of a precisely defined data-basing concept or term (e.g., **Table**). Subsequent use of the term will be identified using *italics* (e.g., *table*). When referring to a specific component of the database, we use the font `Courier New` (e.g., `table BreedingAttempts` to refer to a

specific component of the database, or *tables* within). When referring to an in-text table (e.g., Table 4.1.), we do not apply any special formatting.

A relational database is the arrangement of information into a set of two or more **Tables** that are linked to each other (via key fields) and allow the data to be accessed or reassembled in many different ways without reorganization or modification. Data are grouped together in *tables,* which are comprised of a series of **Records** (in rows), and the distinct attributes of those *records* are stored as **Fields** (in columns). One *table* should be comprised of many *records,* each corresponding to an individual entity, such as a single nesting site. Many *records* (e.g., known nesting sites) arranged together would, for example, comprise the `table NestingSites.` Data (e.g., the geographical coordinates of a nesting site) about each *record* are stored in *fields,* which increase in number to accommodate more data to describe each entity.

Tables, one type of database **Object,** are the essential components of relational databases. Three additional *object* types common to many Microsoft Access databases, are: **Forms** (user interface for inputting or manipulating data), **Queries** (tools to select data contained in one or more tables), and **Reports** (stylized summaries of the data, often displaying selected data). Data *tables* are linked using formalized **Relationships** (an association between mutually shared information in two or more *tables*).

In contrast to relational databases, flat-file databases are an arrangement of *records* that have no structured relationships and usually consist of a series of rows and columns in tabular format (e.g., a spreadsheet). Information in a well-organized flat-file database can be sorted, filtered, and transformed. Although flat-file databases are an intuitive and simple way to store data, they are error prone, inflexible, and cumbersome to use compared to relational databases. The main advantages of a relational database over a flat-file database are easy access and re-assembly of data, prevention of data duplication and inconsistency and, straightforward formatting, editing, adding, and deleting data.

4.2.2 Starting the data-basing process

A valuable approach to initiate the construction of a relational database is to generate a list of all the data that will be collected, and itemize the attributes associated with the data. Eventually each datum will be stored in a *field.* For each item, catalog the *field* name, the type of data, the value that a typical datum could take on, the size of a typical datum, and describe each type of datum (see Table 4.1). Data-basing software (including Microsoft Access, in the "Design View" of a *table*) will store descriptions of data, constrain the types and sizes of data that can be entered into a *field* (e.g., text, integers, real numbers), identify specific values that can be

entered (e.g., "0" or "1"), and specify the format in which the data must be entered (e.g., YYYY–MM–DD for dates). *Fields* that describe one set of *records* will eventually become part of a *table* within the database (e.g., `table NestingSites`). The exercise of listing all of your data also creates metadata that describe the data in your database, and initiates the process of dividing data into *tables.* Before generating our *tables,* we will discuss a number of principles users should follow to ensure data quality is assured.

4.2.3 Principles of data tables

Here, we discuss a series of principles that promote appropriate data management in the context of relational database structure (referred to as **Normalization**; Ritchie 2002). The three goals are to: 1) minimize errors and inconsistency, 2) maximize flexibility, and 3) maximize usage efficiency. There are many types of errors to avoid. A datum may be entered or altered erroneously, information in one location can disagree with information stored elsewhere, or data may be incorrectly formatted and become unusable. Maximum flexibility is intended to give the greatest ability to change, update and expand the database, export data in formats required by funding agencies, data banks (e.g., eBird, USFWS Bird Banding Laboratory) and statistical software. Finally, by maximizing efficiency, all data entry, manipulation, reporting, and analysis can proceed without slow and error prone data processing (e.g., copying and pasting).

1. **Primary Keys** are unique identifiers for each *record* within a *table*. They must not be repeated and are crucial for establishing *relationships* among *tables*.
2. Each *field* within a *record* should contain information about the entity (i.e., about the *primary key*).
3. Data should not be stored redundantly. Do not store a datum in more than one *table*, or *field*.
4. *Fields* should be atomic (i.e., one datum per *field*, per entity)
5. Each *field* should have a unique name that is not duplicated across the database.
6. Similar data are not stored in unintuitive places, termed "inconsistent dependency" (e.g., it is illogical to store data about banded adults in a `table NestingSites`, because the adults present at a given Nesting Site change through time).
7. Similar data are not stored across multiple *fields* (e.g., it is correct to store all data about nestling weights in one `field NestlingWeight`, rather than as a series of `fields: Nestling1Weight, Nestling2Weight`).

4.3 An example database

4.3.1 The research objective: estimating nestling growth

A DBMS is a natural extension of the activities carried out in a field program. Here we provide an exemplar database using the simulated nestling growth data from Chapter 6 (Modeling nestling growth) to help the reader visualize the link between activities carried out in the field and activities carried out to curate data. In Chapter 6 the authors compared Gyrfalcon nestling growth at 33 nesting territories among years with varying rainfall using simulated data generated for four years (2012 to 2015). The simulated data assumed nesting territories were visited regularly until occupancy was confirmed or until the breeding season was sufficiently advanced to conclude that the site was unoccupied. Researchers visited occupied sites weekly to record the number of eggs laid and hatched, number of nestlings, and nestling weights. Cameras at nest sites quantified laying dates, hatching dates, and causes of nestling mortality.

To effectively address this research objective, it is important to know the weight of nestlings at various intervals, the age of the nestlings, the nest site at which they hatched (to be incorporated into models as a random intercept), as well as the hatch order (asynchronous hatch), and sex of each nestling. The exemplar database we construct, called `GYRF Growth.accdb`, will incorporate and store all data collected from the initiation of this study and facilitate the addition of new data as they are collected. Here we will discuss the Construction of the database (Section 4.3.2), Data Entry (Section 4.3.3), and the extraction of the required growth data from the database using a SQL Query outlined in Section 4.3.4 (Manipulating data).

4.3.2 Database construction

4.3.2.1 Which tables are needed

The *Fields* presented in Table 4.1 can be logically grouped into tables, using the principles in Section 4.2.3. In Table 4.1 the `field Category` indicates the logical sets that comprise our dataset. `Latitude` and `Longitude` values describe the location of individual nesting sites, and are thusly grouped. However, the species occupying a nesting site may change year-to-year, and it is therefore illogical to store the `field OccupantSpecies` in the `table NestSites.` Thus, another set in our data is a `table BreedingAttempts`, where information about a breeding attempt, such as year and occupying species are stored. The `NestSites` and `BreedingAttempts` tables are linked via the `field NestSitePKEY.` By grouping our data in this way, it follows that we need four different tables to store these data. Tables 4.2 – 4.5 give 10 *records* from each *table.* Readers can execute the SQL code in Section 4.3.4 to generate the dataset used for statistical analysis in Chapter 6.

Table 4.1 A first step in the construction of a relational database is to generate a list of all the data that will be collected, and identify the attributes associated with the data. Each row corresponds to different data that were collected to study Gyrfalcon nestling growth rates. The FIELD NAME, DATA TYPE, SIZE and FIELD DESCRIPTION comprise Meta-data that are important for data-basing.

FIELD NAME	EXAMPLE VALUE	DATA TYPE	SIZE (Characters)	FIELD DESCRIPTION	CATEGORY
NestingSite	10	integer	1-3	Nesting Site Identication number	NestingSites
Latitude	35.12345	real number	8	Decimal degrees, 5 places	NestingSites
Longitude	-115.12345	real number	8-10	Decimal degrees, 5 places	NestingSites
BreedingSeason	2013	integer	4	Year of study	BreedingAttempt
OccupantSpecies	GYRF	categorical	4	AOU 4-letter species code. Can be UNK (Unknown) or NONE (Not occupied)	BreedingAttempt
WeatherCondition[a]	0	binary	1	1 (Treatment) or 0 (Control) or Null, indicating Supplemental Feeding Experiment	BreedingAttempt
HatchDate	2016-07-15	date	10	Date Hatched from Egg. Input format must be YYYY-MM-DD	Nestlings
Sex	m	categorical	1	M (Male), F (Female) or, U (Unknown)	Nestlings
Nestling	red	categorical	3-5	nestling hatch order from oldest to youngest (red -> blue -> green -> white)	Nestlings
Weight	750	integer	3-4	Weight in grams (g)	Weights
DateWeighed	2016-08-09	date	10	Date individual was weighed. Input format must be YYYY-MM-DD	Weights[a]

[a] This field corresponds to the field "COND" in Chapter 6

Table 4.2 Truncated `NestSites` table from `GYRF Growth database` showing nesting site identification number and geographic coordinates. (Note: the geographic coordinates are not required to conduct the growth analysis described in Chapter 6, and will not be included in the query output.)

NestSitePKEY	NEST	Latitude	Longitude
1	7	69.54901	-82.56129
2	8	69.77236	-82.18895
3	23	69.46888	-82.46216
4	31	69.03081	-82.26486
5	35	69.60018	-82.31784
6	72	69.00331	-82.76241
7	75	69.78712	-82.24231
8	4	69.70351	-82.95098
9	19	69.66146	-82.96091
10	28	69.81256	-82.08394

Table 4.3 Truncated `BreedingAttempts table` from `GYRF Growth database` showing breeding attempts (occupancy) that occurred at 33 nest sites from 2012–2015. `COND` indicates whether it was a heavy rain year (1) or regular rain levels (0). `NestSiteFKEY` defines the relationship between the `NestSites table` and the `BreedingAttempts table.`

BreedingAttemptPKEY	NestSiteFKEY	YEAR	COND	OccupantSpecies
1	26	2015	0	GYRF
2	8	2013	1	GYRF
3	8	2015	1	GYRF
4	1	2013	0	GYRF
5	2	2013	0	GYRF
6	2	2015	1	GYRF
7	29	2015	1	GYRF
8	9	2013	1	GYRF
9	9	2015	0	GYRF
10	16	2014	0	GYRF

Table 4.4 Truncated `Nestlings table` from `GYRF Growth database` showing 1–4 nestlings hatch at each breeding attempt. Each nestling is given a unique ID in the `field ID`. `CHICK` indicates hatch order, such that red hatched first, followed by blue, green and black. `BreediingAttemptFKEY` defines the relationship between the `BreedingAttempts table` and the `Nestling table.`

NestlingPKEY	BreedingAttemptFKEY	ID	CHICK	SEX	HATCHDATE
1	4	7black2013	black	f	7-14-2013
2	4	7blue2013	blue	f	7-11-2013
3	4	7green2013	green	m	7-10-2013
4	4	7red2013	red	m	7-10-2013
5	5	8black2013	black	f	7-17-2013
6	5	8blue2013	blue	m	7-16-2013
7	5	8green2013	green	m	7-14-2013
8	5	8red2013	red	f	7-13-2013
9	12	23green2013	green	m	7-7-2013
10	12	23red2013	red	f	7-7-2013

Table 4.5 Truncated `Weights table` from `GYRF Growth database` showing mass (g) of each nestling at each site visit at approximately 5-day intervals. The `NestlingsFKEY` defines the relationships between the `Nestlings table` and the `Weights table.`

WeightsPKEY	NestlingsFKEY	WEIGHT	VISITDATE
1	1	1836	8-13-2013
2	1	1756	8-7-2013
3	1	1276	8-1-2013
4	1	792	7-27-2013
5	1	438	7-23-2013
6	1	202	7-19-2013
7	2	1746	8-13-2013
8	2	1738	8-7-2013
9	2	1408	8-1-2013
10	2	856	7-27-2013

4.3.2.2 Which Fields should be primary keys

As highlighted in Section 4.2.3, researchers must select a *field*, or combination of *fields*, that uniquely specify each *record* in a *table*, called *primary keys.* These keys are critical for establishing *relationships* between *tables*. *Primary keys* are effective if: 1) each value is unique, 2) the datum is never empty or null, and 3) the value is permanent. In Table 4.2 this is accomplished using the `field NestSitePKEY`, a special integer called an Auto-Number, which is an incrementally increasing integer that will not repeat (i.e., the *field* has the property "Indexed" in the Microsoft Access "Design View"). The `field SiteID` could also be used as a *primary key* (alternative keys are called **Candidate Keys**), but only if the researchers ensure that `SiteIDs` are never repeated. However, lack of permanency may preclude `SiteID` as a good *primary key* choice if any site names (i.e., identification numbers) change over the project's history. A combination of *fields* can also produce a *primary key* (called a **Composite Key**), for example, the combination of `SiteID` and `Year` is a suitable *primary key* for a given breeding attempt (Table 4.3). Researchers must choose between, Auto-Number *primary keys,* and other *candidate keys. Primary keys* appear in the *table* for which they identify unique entities, and they also appear in *tables* to which they are related, as **Foreign Keys**. This link between *primary keys* and *foreign keys* is the foundation of the *relationships* between tables.

4.3.2.3 How will the Tables be related?

In Microsoft Access, after your *tables* are created, *relationships* are created in the relationships manager (Fig. 4.1). *Relationships* instruct the manner in which tables are related to one another, which is important for both data manipulation and data entry. There are three types of *relationships*.

One to One — The *primary key* in one *table* matches one or zero *primary keys* in a second *table.* As an example, we may use the USFW Band number as a *primary key* to identify individual raptors (assuming that bands are never replaced), and also use the USFW Band number as a *primary key* for a second *table* where we keep track of all of our bands, deployed or not.

One to Many — This *relationship* can be visualized as a hierarchy, where "child" *tables* are nested beneath a "parent" *table.* The *primary key* in the parent *table* is related to one or more *records* in the child *table.* This is the most common type of *relationship.* In our example the parent `table NestSites` is related to the child `table BreedingAttempts`, where multiple breeding attempts occur at a given nesting site across years.

Many to Many — Multiple *records* in one *table* can be associated with multiple *records* in another *table*. This *relationship* is powerful for contributing to error free use of complex datasets. For example, a database could relate any number of banded raptors (in one *table*) to any number of nesting sites (a second *table*) over time. Plainly stated, an individual raptor can

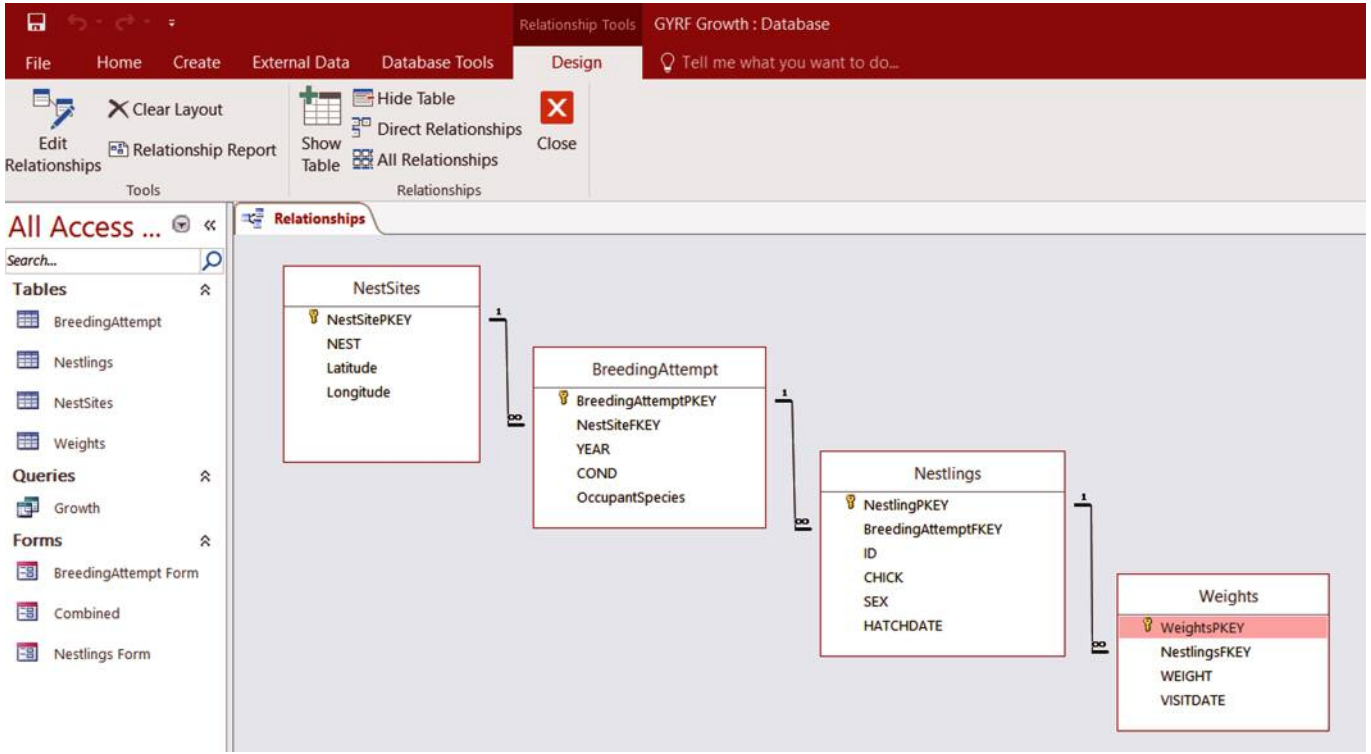

Figure 4.1 A tool in Microsoft Access to visualize and manipulate relationships between tables. In this instance four tables are related hierarchically using one-to-many relationships.

have multiple nesting sites over time, whereas an individual nest site may be occupied by multiple individual raptors over time (i.e., many nest sites can be occupied by many individuals).

4.3.3 Data entry

Once the *tables* have been created, data *fields* defined, and *relationships* among tables established, the next step will likely be data input. Data entry can be problematic when inputting data into multiple *tables* (some databases will require many *tables* with different *relationships* among them), because of the need to maintain *primary* and *foreign keys*, and the *relationships* that depend on them. Although it is possible to input values directly into *tables*, this method may increase the likelihood of error. Databasing software, including Microsoft Access, offers two solutions for users: 1) **Subdatasheets**, and 2) *Forms* that facilitate data input. In Microsoft Access, *subdatasheets* are automatically generated when one to many *relationships* are established between two *tables*. *Subdatasheets* allow users to input data via an arrangement of related *tables* into a collapsing and expanding interface (Fig. 4.2A). In the `database GYRF Growth`, *subdatasheets* can be viewed by opening `table NestSites`, the highest tier *table* comprising this dataset. *Forms* can perform tasks similar to *subdatasheets*, but offer greater visual and structural flexibility for complex data. In Microsoft Access several smaller *forms* can be generated, each of which typically reflect the *tables* in the database. We can then merge the various *forms* into one or more large form(s) (Fig. 4.2B). Examples of *forms* can be found in `GYRF Growth.` In Microsoft Access, simple *forms* that display data for one *record* (e.g., `BreedingAttempt Form`) can be created using the "Form" button in the "Create" tab. Records in simple *forms* are navigated using the arrows at the bottom of the *form*. "Multiple Items" *forms*, displaying multiple records (e.g., `Nestlings Form`) can be created using the "Multiple Items" button. Two forms can be combined to show data in each form simultaneously, such as `form Combined`, which shows `Nestling Form` nested within `BreedingAttempts Form`. Ultimately, the user's ability to customize forms is immense, and detailed instruction is beyond the scope of this chapter.

4.3.4 Manipulating data using queries

There are many different types of *queries* available for viewing, rearranging, and summarizing data. In the "Create" tab, users can select the "Query" button to open a simple user interface called "Design View" where the *tables* and *fields* of interest are selected. Selecting the "Run" button displays the output of the query. In `database GYRF Growth` a *query* called `Growth` is used to display the data in the correct format for statistical analysis in Chapter 6 (note that underlying data in the *tables* are not changed). Double clicking `query Growth` will display the output of the query, whereas

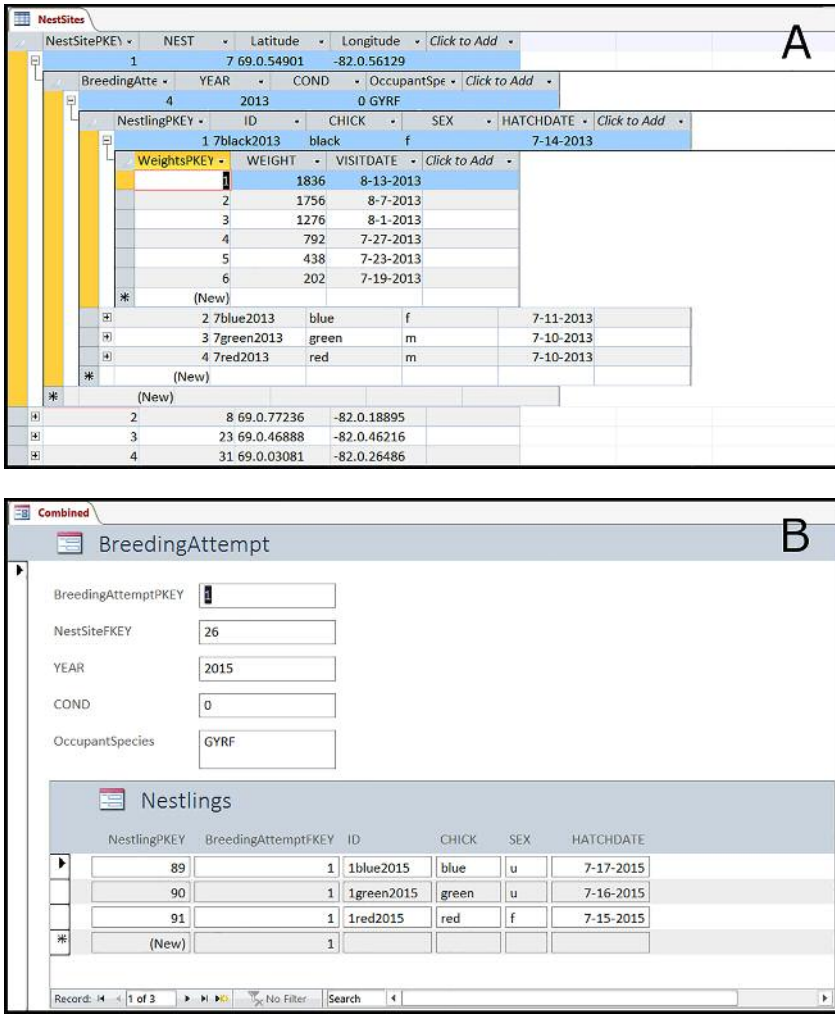

Figure 4.2 Subdatasheets in Microsoft Access displaying multiple related tables (A), and a multiple items form (Nestlings) nested within a simple form (BreedingAttempts). Both subdatasheets and forms facilitate the entry of data into multiple tables.

clicking "Design View" will show the selected *fields, tables,* and, the *relationships* among *tables.* In addition to selecting *fields* from *tables,* a new *field* equal to the difference between `field HatchDate` and `field DateWeighed` is designated as: `AGE.` In this way operations can be performed across *tables.* This *query* also specifies a *criteria* component stating that only records with `field OccupantSpecies` equal to "GYRF" should be returned in the output.

Microsoft Access provides a Graphical User Interface (GUI) in "Design View" that allows users to generate *queries* easily without being fluent in coding language. However, it should be noted that all *query* operations are based on a Structured Query Language (SQL) code that can be viewed in the "SQL View". In plain language the SQL statement "Selects" the desired *fields,* "From" the *tables* that are *related* to one another as specified, "Where" certain criteria are met. These "Select," "From," and "Where" components can be seen below in the SQL Code underlying `query Growth.` Like *forms, queries* are a complex topic and detailed instructions for their use is beyond the scope of this chapter.

```
SELECT Nestlings.ID, NestSites.NEST, Nestlings.CHICK,
Nestlings.SEX, BreedingAttempt.Year, BreedingAttempt.COND,
[Weights]![VISITDATE]-[Nestlings]![HATCHDATE] AS AGE,
Weights.WEIGHT

FROM NestSites INNER JOIN ((BreedingAttempt INNER JOIN
Nestlings ON BreedingAttempt.BreedingAttemptPKEY =
Nestlings.BreedingAttemptFKEY) INNER JOIN Weights ON
Nestlings.NestlingPKEY = Weights.NestlingsFKEY) ON Nest-
Sites.NestSitePKEY = BreedingAttempt.NestSiteFKEY;

WHERE (((BreedingAttempt.OccupantSpecies)="GYRF"));
```

4.4 Additional data-basing resources

4.4.1 Server data-basing and alternative DBMS

Our description of data-basing in this chapter has been limited to local applications. This is to say, *tables* are stored, edited, and queried locally on a single computer. However, one powerful application of data-basing is the ability to centralize *table* storage using online servers. For researchers, this can allow multiple individuals using different computers to collect, enter, and view data in tables. Further, online data-basing offers additional security against hard drive failure or erroneous data loss. Microsoft Access is primarily a local data application, but if used in conjunction with online

data servers (such as Microsoft SharePoint www.microsoft.office.com/sharepoint, or Microsoft SQL Server www.microsoft.com/sql-server) it can manipulate and interact simultaneously with online and local *tables*. Server data-basing may become more useful for researchers as mobile phone technology grows, and mobile devices are better able to interact with online data servers (see SQLTool; www.sqltoolpro.com).

Although we used Microsoft Access as the illustrative DBMS in this chapter, researchers should be aware of alternatives. Many businesses use Oracle (www.oracle.com) or IBM DB2 (www.ibm.com/db2) because these platforms provide improved security, processing speed, and support for a greater number of users and larger data sizes. Despite advantages, the cost of these first-class DBMS may be prohibitive for many field research programs, and Microsoft Access (or other entry level software; e.g., FileMaker; www.filemaker.com) will meet most research needs. Regardless of the DBMS selected to manage a database, the underlying principles of relational data-basing outlined in this chapter will not change.

4.4.2 The Polar Raptor Databank (PRDB)

The Peregrine Fund (www.peregrinefund.org), Habitat Info (www.habitatinfo.com), and the Mohamed bin Zayed Species Conservation Fund (www.speciesconservation.org) have partnered to develop the Polar Raptor Databank (PRDB, https://gis.habitatinfo.com/tpf/). The current version of the PRDB allows users to upload, store, delete, *query*, and share project-specific data via password protected state-of-the-art server technologies. The user interface is a web-based, query driven geographical information system powered by Esri (www.esri.com); it includes a map window, four command dialogues (Query, Tools, Layers, Basemaps) and an attribute window that displays results from *queries*, and allows data export. Access to data is tightly controlled by server security so that sensitive features such as nest sites are only revealed to the owners of the data. Once users have registered projects and nesting sites, data identifying site-specific occupancy status, species, and reproductive status can be uploaded using a pro forma spreadsheet. On-the-fly summaries of activity at each nesting territory are reported on the web interface by clicking on a nesting territory. As updates become available, the developers expect to implement functions that allow data *queries* and downloads consistent with the data chapters that appear in this book.

Literature cited

Applegate, R. D. 2015. The importance of data management in wildlife conservation. Wildlife Society Bulletin 39:449–450.

Briney, K. 2015. Data Management for Researchers: Organize, maintain and share your data for research success. Pelagic Publishing. Exmouth, UK.

Chamanara, J., and B. König-Ries. 2014. A conceptual model for data management in the field of ecology. Ecological Informatics 24:261–272.

Hunt, V. M., S. K. Jacobi, M. G. Knutson, E. V. Lonsdorf, S. Papon, and J. Zorn. 2015. A data management system for long-term natural resource monitoring and management projects with multiple cooperators. Wildlife Society Bulletin 39:464–471.

Léonard, M. 1992. Database Design Theory. MacMillan Publishing Company. Bassingstoke, UK.

R Core Team. 2016. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/.

Reynolds, J. H., M. G. Knutson, K. B. Newman, E. D. Silverman, and W. L. Thompson. 2016. A road map for designing and implementing a biological monitoring program. Environmental Monitoring and Assessment 188:399.

Ritchie, C. 2002. Relational database principles. Cengage Learning EMEA. London, UK.

Rüegg, J., C. Gries, B. Bond-Lamberty, G. J. Bowen, B. S. Felzer, N. E. McIntyre, P. A. Soranno, K. L. Vanderbilt, and K. C. Weathers. 2014. Completing the data life cycle: using information management in macrosystems ecology research. Frontiers in Ecology and the Environment 12:24–30.

Sólymos, P., S. F. Morrison, J. Kariyeva, J. Schieck, D. L. Haughland, E. T. Azeria, T. Cobb, R. Hinchliffe, J. Kittson, A. C. S. McIntosh, T. Narwani, P. Pierossi, M.-C. Roy, T. Sandybayev, S. Boutin, and E. Bayne. 2015. Data and information management for the monitoring of biodiversity in Alberta. Wildlife Society Bulletin 39:472–479.

Sutter, R. D., S. B. Wainscott, J. R. Boetsch, C. J. Palmer, and D. J. Rugg. 2015. Practical guidance for integrating data management into long-term ecological monitoring projects. Wildlife Society Bulletin 39:451–463.