FACTFILE:
GCSE
DIGITAL TECHNOLOGY
Unit 1 – DIGITAL DATA

Rewarding Learning

Fact File 7: Database Applications (1)

Learning Outcomes

Students should be able to:

- demonstrate understanding of and explain basic database concepts such as table, record, field, key field, query, form, report, macro, relationship and importing data;
- identify and use appropriate data types when creating a database structure;
- demonstrate understanding of the need for data validation; and
- describe the following types of validation checks: presence, length, type, format and range.

Content:

- Basic Database Concepts;
- Data Types;
- Data Validation.

Basic Concepts

Databases are very widely used and many of us will have interacted with one – without necessarily being aware of it. If you have entered your personal details into a social networking site, they have almost certainly been stored in a database; if you have browsed an online retail site, product details will almost certainly have been retrieved from a database.

Database

"[A] *database* is a (large) collection of data items and links between them, structured in a way that allows it to be accessed by a number of different application programs. The term is also used loosely to describe any collection of data."

BCS Glossary Glossary of Computing, 14th edition, p 90.

Here are some other contexts in which a database might be used:

- Your school might have a database that contains data about pupils, attendance, subjects and assessment marks;
- An employer might have a database that contains data about employees, job titles and salaries;
- The organisers of the 2016 Rio Olympic Games might have a database that contains data about athletes, events and medals.

Database Structure

The most common way to structure a database is as a collection of interrelated tables. This kind of database is known as a *relational database*.

Database Table

"[A] *table* is the name for each group of similar data with rows for each instance of an *entity* and columns for each *attribute*."

BCS Glossary Glossary of Computing, 14th edition, p 94.

Consider how the organisers of the 2016 Rio Olympic Games might use a database to store data about athletes and events. You can find relevant factual data here:

- **https://www.rio2016.com/en/medal-count-sports**

A small subset of this data (for some cycling events) has been used to create Tables 1 and 2, in Example 1 below.

Example 1: Rio 2016 Olympic Games Database

Table 1: Athlete

Athlete_ID	Surname	Forename	Team
A001	Van Avermaet	Greg	BEL
A002	Cancellara	Fabian	SUI
A003	Van Der Bruggen	Anna	NED
A004	Armstrong	Kristin	USA
A005	Fugslang	Jacob	DEN
A006	Dumoulin	Tom	NED
A007	Johansson	Emma	SWE
A008	Zabelinskaya	Olga	RUS
A009	Majka	Rafal	POL
A010	Froome	Christopher	GBR
A011	Longo Borghini	Elisa	ITA

Table 2: Event

Event_Name	Gender	Gold_Medalist	Time
Road Race	M	A001	06:10:05
Individual Time Trial	M	A002	01:12:15
Road Race	F	A003	03:51:27
Individual Time Trial	F	A004	00:44:26

Table 1 (Athlete) stores, for example, the fact that Christopher Froome has Athlete_ID *A010* and is a member of team *GBR*.

Table 2 (Event) stores, for example, the fact that the gold medal winner of the Men's Road Race was the athlete with ID *A001*. Athlete *A001*'s name is Greg Van Avermaet, but, for this information, we must cross reference with Table 1.

Records & Fields

Record

"[A] *Record* is the basic unit of data stored in a data file. It is a collection of data items, which may be of different data types, all relating to the individual or object that the record describes and is treated as a unit for processing."

BCS Glossary of Computing, 14th edition, p 328.

In the database context, a *record* is really a *row* of data from a database table. For example, the Athlete table contains eleven records, including the following one.

A010	Froome	Christopher	GBR

Records may sometimes be written down in this fashion:

< A010, Froome, Christopher, GBR >

Each record (or row) in the Athlete table describes an individual athlete, while each record (or row) in the Event table describes an individual event.

Field

"[A] *Field* is part of a record designed to hold a single data item of a specified type."

BCS Glossary of Computing, 14th edition, p 329.

We can talk about *field names* and *field values*. For example, the Christopher Froome record above has a field named Team, and this field has the value *GBR*. All records in a database table have the same number of fields, with the same field names. For example, all records in the Athlete table have field names: Athlete_ID, Surname, Forename and Team. These, of course, correspond to the column headings in the table.

Keys

Some fields perform important functions such as identifying a particular record or establishing a relationship between two tables. These are called *Keys* or *Key Fields*.

> **Key Field**
>
> "[A] *Key* is the field within a record used to identify the record."
>
> BCS Glossary of Computing, 14th edition, p 329.

Primary Key: A field that uniquely identifies an individual record in a table can be chosen as the *Primary Key* of the table.

For example, since Athlete_ID uniquely determines an individual record in the Athlete table, it may be chosen as the primary key for that table.

These are all different ways of saying the same thing:

- Athlete_ID uniquely identifies a single record in the Athlete table;
- Each Athlete_ID value is unique;
- No two records in the Athlete table have the same value in Athlete_ID field.

We might also, equivalently, say that: Athlete_ID uniquely identifies an athlete; Athlete_IDs are unique, or no two athletes have the same Athlete_ID.

While *Surname* also uniquely identifies an athlete it would be a poor choice for primary key. This is because a database should be designed to cater for all the possible data that it might reasonably be used to store. While there are no name clashes in the current data, there might well be clashes in data for another event, or for another year's games. For this reason Surname is not a suitable choice for primary key.

In choosing a primary key for the Event table there is only one field that is distinct for each record – the Gold_Medalist field. This is not, however, a suitable choice because it is likely that, when more data is added, there will be some athletes who win more than one gold medal. In order to choose a primary key for the Event table, we must consider what are known as *composite keys*.

Composite Key: A composite key is one that consists of two or more fields.

In the Event table, if we know *both* the name of an event and the gender of the competitors we can uniquely identify an event. This combination of fields is therefore a suitable choice for primary key of the table. As the key consists of two fields, we say that it is a *composite key*.

Sometimes the primary key of one table appears as a field in another table. This happens in the *Event* table, where Athlete_ID values appear in the *Gold Medalist* column.

Foreign Key: When the primary key from one table appears as a field in a second table, it is known as a *Foreign Key* of the second table.

For example, Gold_Medalist is a foreign key of the Event table.

Note that the key does not have to be named the same in both tables to be considered as a foreign key – it just needs to describe the same information. In our example, the primary key of the Athlete table is named Athlete_ID. When it appears in the Event table, it is named, Gold_Medalist. Despite the different names, it is the same data – the Gold_Medalist field contains the Athlete_ID of the athlete who won the gold medal.

Relationships

In the Olympic Games example, Kirstin Armstrong won the gold medal in the Women's Individual Time Trial event. This is an example of a *relationship* – which we might call *event winner* – between a particular athlete and a particular event. Of course other athletes are also event winners – for example, Greg van Avermaet won the gold medal in the Men's Road Race. So we say that a relationship – called event winner – exists between athletes and events.

The way relationships are represented in a relational database involves the use of primary and foreign keys. If the primary key of one table appears as a foreign key in a second table, then we say that there is a relationship between the two tables. For example, the primary key of the Athlete table (Athlete_ID) appears as a foreign key in the Event Table (Gold_Medalist): this indicates a relationship between the tables.

We can see which individual athletes are related to which individual events by cross referencing the values in the respective foreign and primary key fields.

Database Design Language

In designing a relational database, it is useful to be able to describe the organisation (tables, fields, keys and relationships) independently of any actual data that might be stored. The following example illustrates a simple notation that can be used for this purpose.

Athlete(Athlete_ID, Surname, Forename, Team) Event(Event_Name, Gender, Gold_Medalist*, Time) Gold_Medalist -> Athlete

The representation of table names and field names here is fairly obvious. Primary keys are underlined. Any foreign keys are marked by an asterisk, with an additional line (Gold_Medalist -> Athlete) to indicate the table to which the foreign key links.

Entity-Relationship Diagrams (ER Diagrams)

The example above represents database structure using a text based notation. An ER diagram can be used to represent similar information, but in a graphical form. For example:

The representation of table names and field names in this diagram is fairly obvious. The line linking the boxes indicates a relationship between them, with the name of the relationship in the diamond shaped box. The little crow's foot where the relationship line meets the Event box has a particular meaning.

It indicates that any one athlete may be related to (i.e. the winner of) many events. The lack of any similar crow's foot at the Athlete end of the line indicates that any one event may be related to (i.e. won by) at most one athlete. We call this a one-to-many relationship.

Database Query

Since databases are usually large collections of data, it is necessary for a user who wants to retrieve data to be able to specify the subset that she is interested in. This is accomplished using a *database query*.

> **Database Query**
>
> "[A] query is a question used to retrieve selected information from a database."
>
> BCS Glossary of Computing, 14th edition, p 102.

For example we might want to ask the following questions in relation to our Olympic Games database.

- In which event(s) did athlete A003 win gold?
- Who won gold in the Men's Road Race?

Special computer languages, called *query languages*, are used to specify queries in a form that can be processed by a computer. A special piece of software called a *query processor* analyses the query and computes the answer.

Structured Query Language (SQL)

SQL will not be examined in Unit 1 but will be in Unit 2, it is listed here to introduce a fundamental understanding of database query concept.

One of the most commonly used query languages is called *Structured Query Language (SQL)*. SQL is an industry standard for relational database processing.

The basic form of an SQL query has three parts: *select*, *from* and *where*.

- In the select clause, we specify the data that is to be returned.
- In the from clause, we specify where the data to be returned can be found.
- In the where clause, we specify any relevant constraints on the data that is to be returned.

The following examples illustrate some SQL queries.

Example 2: *In which event(s) did athlete A003 win gold?*

```
SELECT Event_Name, Gender, Gold_Medalist
FROM Event
WHERE Gold_Medalist = 'A003'
```

The query processor's response to query this is:

Event_Name	Gender	Gold_Medalist
Road Race	F	A003

Example 3: *Who won gold in the Men's Road Race?*

```
SELECT Gold_Medalist
FROM Event
WHERE Event_Name = 'Road Race' AND Gender = 'M'
```

The query processor's response to this query is:

Gold_Medalist
A001

Note that the response to an SQL query is always a table, albeit sometimes a table with only a single cell.

Example 4: *Who won gold in the Men's Road Race and what was their time?*

```
SELECT Gold_Medalist, Time
FROM Event
WHERE Event_Name = 'Road Race' AND Gender = 'M'
```

The query processor's response to this query is:

Gold_Medalist	Time
A001	6:10:05

Example 5: *Which athletes are members of team NED? Give their full records.*

```
SELECT *
FROM Athlete
WHERE Team = 'NED'
```

The query processor's response to this query is:

Athlete_ID	Surname	Forename	Team
A003	Van Der Bruggen	Anna	NED
A006	Dumoulin	Tom	NED

Example 5 shows how the wildcard character, "*", is used to match all columns in the specified table.

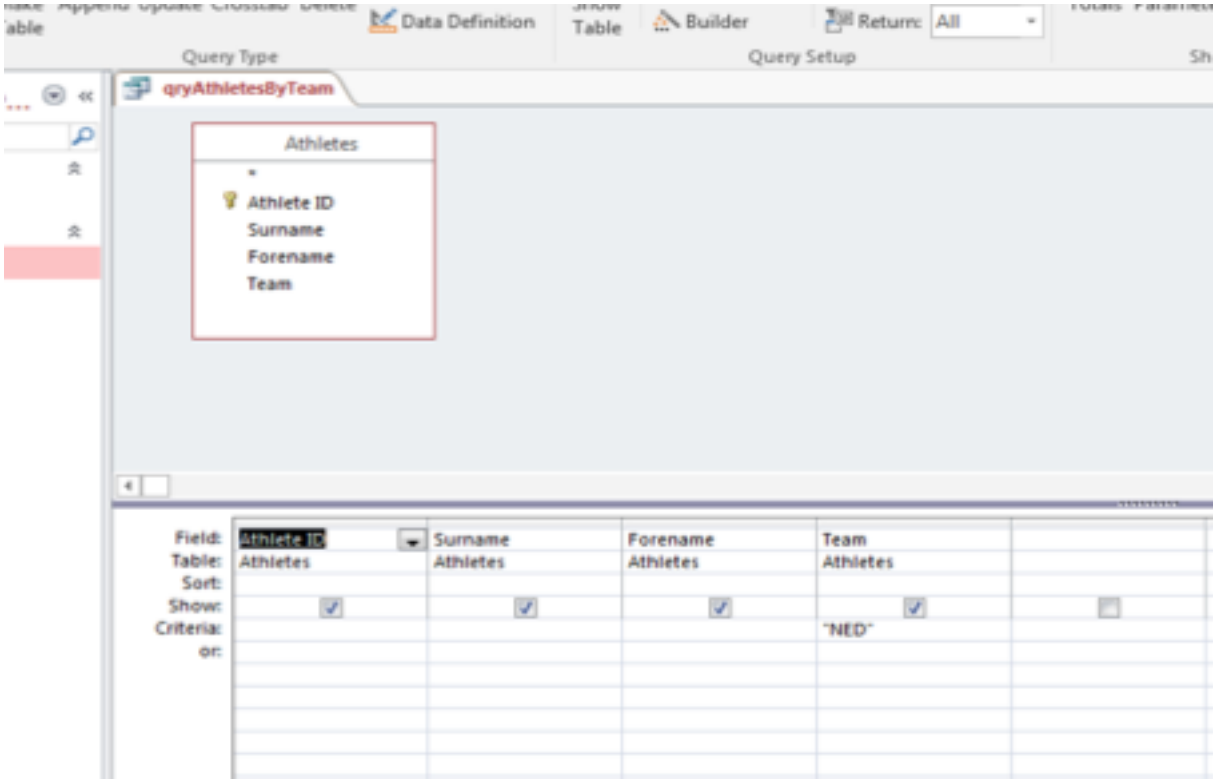
Query by Example (QBE)

SQL is a query language that is oriented towards software developers: it has a precise syntax and is often embedded within program code. QBE, on the other hand, is a simple to use query language that is oriented towards the end user.

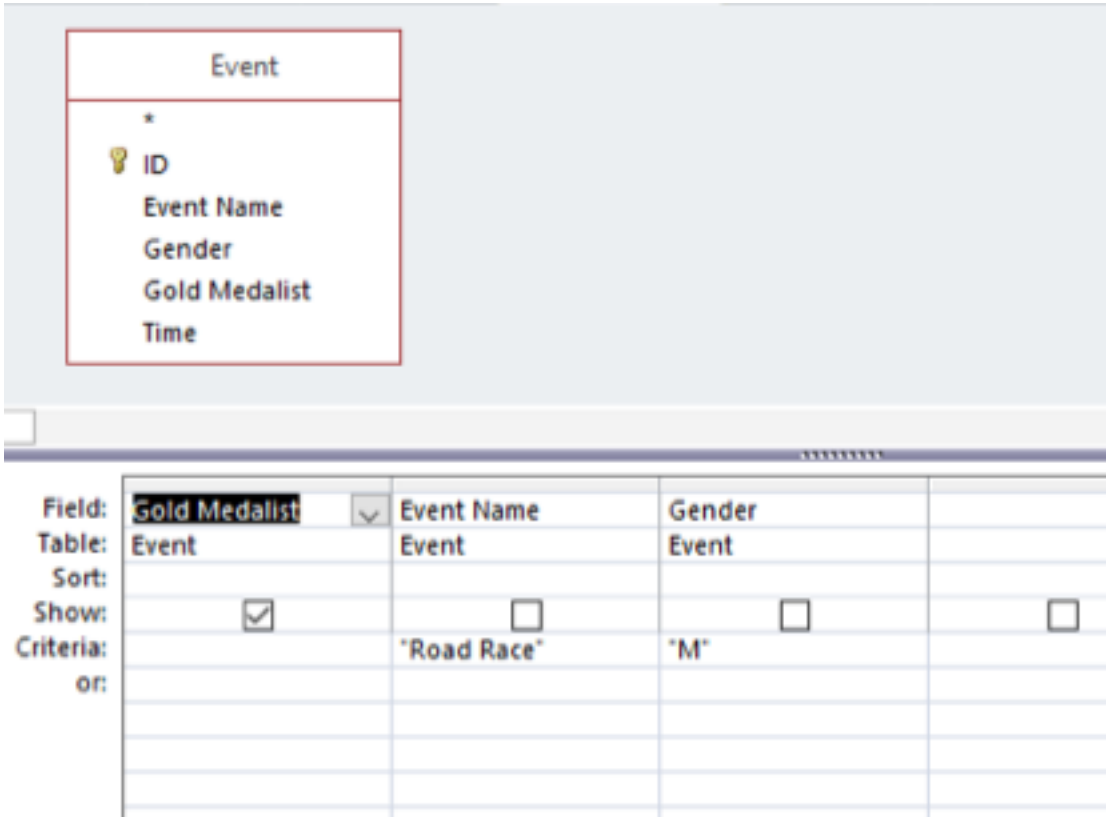
In QBE the user uses a *form based query tool* that contain controls such as text boxes, drop down lists, radio buttons and checkboxes. The user fills in the form to describe the data that she wants to retrieve, and submits this to the query processor.

There is no standard QBE: each implementation does things a little differently. The queries in examples 6-8, below, were created using the QBE features of Microsoft Access 2016. They are QBE equivalents to some of the SQL queries listed above.

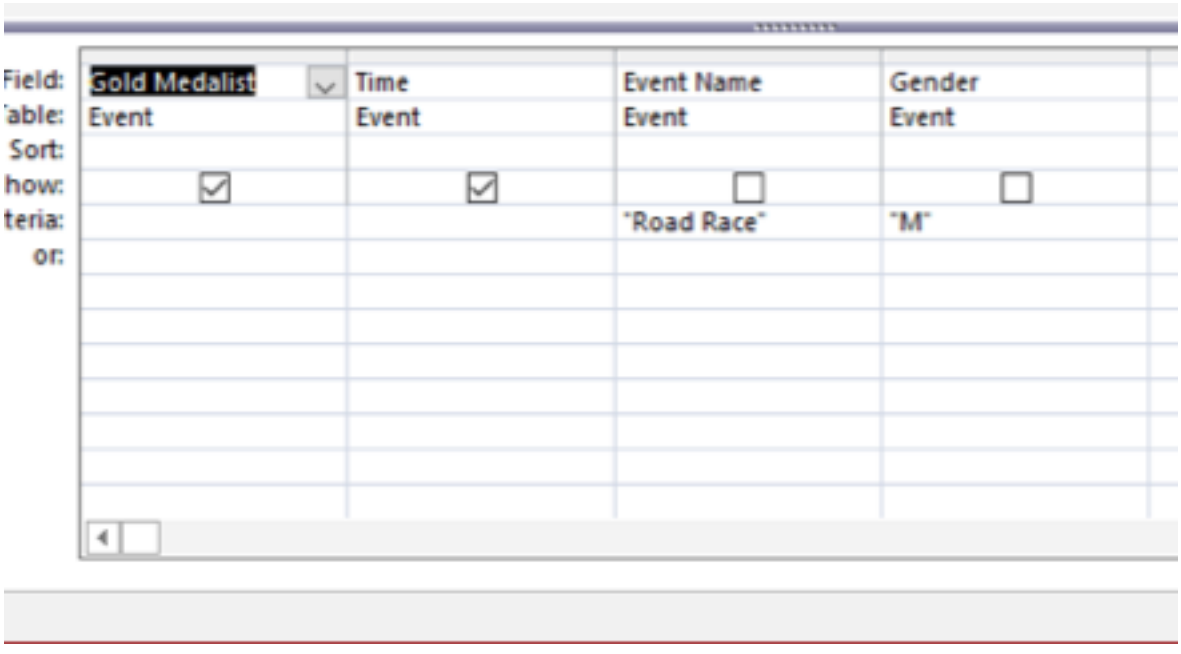
Example 6: *Which athletes are members of team NED? Give their full records.*
Equivalent to example 5.



Example 7: *Who won gold in the Men's Road Race?*
Equivalent to example 3.



Example 8: *Who won gold in the Men's Road Race and what was their time?*
Equivalent to example 4.



Reports

If complex data is extracted from a database, it needs to be carefully presented so that it is meaningful to the user. This presentation is known as a *report*.

Database Report

"[A] report is the presentation of selected data from a database."

BCS Glossary of Computing, 14th edition, p 101.

Reports are often intended to be printed, and require a detailed definition so that they may be generated automatically. A report definition will normally include:

- specification of any data that is to be included (using e.g. SQL);
- formatting and page layout instructions;
- grouping and/or sorting instructions for any tabular data.

Reports are typically defined in advance and may be generated at regular intervals. For example, a retailer may generate monthly sales reports to help senior management monitor performance. In this case the report definition would be created once only, and would not be expected to change from month to month. Each month's report would have the same structure, formatting and layout. Any data contained in the report would, of course, be different for each month.

Here is an example of a report generated to show how many hours staff work for a landscape company on a particular service and how much money was made on each type of service.

Summary of Hours and Service Charge by Type of Service

Type of Service	No of Job Hours	Service Charge
Decking	48	£858.00
Grass Cutting	266	£5,089.00
Hedge Trimming	6	£135.00
Landscaping	108	£9,550.00
Patio Cleaning	40	£755.00
Planting	25	£425.00
Weeding	25	£484.00
GRAND TOTAL	**518**	**£17,296.00**

A report definition may be stored as a *macro*, which can then be run using a single instruction.

Macro

"[A] *macro* is a small program to perform a repetitive task and which can be created and stored for later use by a user."

BCS Glossary of Computing, 14th edition, p 13.

Macros allow functionality to be added to forms, e.g. for navigation or menus or add a new record button.

Importing and Exporting Data

It is sometimes necessary to move data from one database system to another. Since data may be structured and stored differently in different systems this task requires special tools.

Imagine that *format 1* is the internal data storage format for *database 1*. Similarly *format 2* is used by *database 2*. *Format 3* is a third format that both systems can process.

The import export process is as follows:

1. Database 1 retrieves data to be exported, from storage;
2. Database 1 converts data from format 1 to format 3;
3. Database 1 *exports* data in format 3;
4. Database 2 *imports* data in format 3;
5. Database 2 converts data from format 3 to format 2;
6. Database 2 stores data.

Export / Import

"[To] *export* is to create a data file using one piece of software so that it can be read by a different piece of software. *Import* is the corresponding read process to accept a file produced by some other software."

BCS Glossary of Computing, 14th edition, p 18.

Data import and export does not have to be between two database systems: it is also common to be able to import and export between database systems and spreadsheet programs.

Files with a .csv format can be used to transfer data and then import into a database.

Data Types

In designing a database it is important to choose appropriate data types for each field in each table. Making good choices at this stage may make it easier to carry out any processing that is later required, and may enable more efficient use of storage space.

Data Types

"[A] *data type* is a formal description of the kind of data being stored or manipulated within a program or system, for example alphanumeric data, numerical data or logical data."

BCS Glossary of Computing, 14th edition, p 330.

The range of data types available varies from one database system to another. The table below illustrates some common data types.

Data Type	Explanation
CHAR(N)	CHAR(N) is string data type, whose values are constrained to have exactly N characters. For example the following are all CHAR(4): "Fred", "Fre ", "Fr d", "1234".
TEXT	TEXT is a string data type, whose values may have a variable numer of characters. For example: "Fred", Fre", "Fr", "F", "Fred12345".
ENUM	ENUM is a string data type, whose values are constrained to be one of a list of permitted values.
INT	INT is a numeric data type that can store whole numbers – which may be either positive or negative. For example: 0, 1, 2, -99.
REAL	REAL is a numeric data type that can store numbers with decimal places. For example: 1, 2.718, 3.141, -2.718, -3.141.
DATE	DATE is a data type that can store dates. The display format can be specified – e.g. 1/1/2017, 1 Jan 2017.
TIME	Time is a data type that can store time values (hours, minutes & seconds). For example, 03:15:00 (quarter past 3 am).
BOOLE	BOOLE is a data type that can store the values TRUE and FALSE.

These data types would be suitable choices for the fields in the Athlete table:

- Athlete_ID: CHAR(4)
- Surname: TEXT
- Forename: TEXT
- Team: CHAR(3)

Software applications such as database or spreadsheet programs customise their data types to suit, e.g. currency or number instead of int.

Data Validation

If incorrect data is entered into a database, the system will be less useful: in some instances it may even be dangerous. It is not always possible to prevent incorrect data being entered, but in certain circumstances it may be. For example, a well designed system ought to be able to prevent a user entering text data into a field that has been designed to store numeric data.

Validation

"*Validation* is the automatic checking of data entered into a computer system."

BCS Glossary of Computing, 14th edition, p 75.

Validation uses the properties of the data to identify any inputs that are obviously wrong. Validation checks should be built into all data entry forms and processes.

Some common kinds of validation checks are listed below, with illustrative examples.

Presence Checks

It would be meaningless, for example, to attempt a bank withdrawal without specifying an account number. Consequently any form that enables the user to make such a withdrawal must ensure that the account number field is not left blank. This is an example of a *presence check*.

Length Checks

The number of characters in a name can vary, but it is difficult to imagine one that is less than three characters long – or one that is more than twenty characters long. It would be reasonable to reject any attempted data entry that contains a name that is outside these limits. This is an example of a *length check*.

Type Checks

In a database that records the number of books in a bookstore, it would be strange to record a fractional number such as 36.4. Consequently any relevant data entry form should ensure that no fractional values are accepted. This is an example of a *type check*.

Format Checks

Some data values are required to conform to given patterns. For example, UK post codes consist of two letters followed by three digits, followed by two letters. Consequently any relevant data entry form should check for this, and reject ill-formed postcodes. This is an example of a *format check*.

Range Checks

Human adults come in a range of heights, but it is difficult to imagine anyone shorter than (say) two feet or taller than (say) ten feet. It would be reasonable to reject any attempted data entry contains a height that falls outside of this range. This is an example of a *range check*.

Exercises

Changes are to be made to the Olympic Games database to enable it to hold silver and bronze medalists as well as gold.

1. Update the design to achieve this – i.e. What, if any, new tables, fields and relationships are needed.

2. Enter some data into your new database structure (this is called *populating* the database). Include sufficient data to provide answers to the questions 3(a&b) below. This link will help you find the data that you need.

   a. https://www.rio2016.com/en/medal-count-sports

3. For the new database design, write queries to answer these questions:

   a. *Who won silver in the Men's Road Race?*

   b. *Who won bronze in the Women's Road Race?*

4. Suggest suitable data types for *all* fields in your new database structure.