COMP0035 Tutorial 8: Database design

Please refer to the database design guide on Moodle or online.

In this tutorial you are going to design a database that will include:

- new data implied in the user stories and application design
- data from the data set which includes two .csv files, one containing details of paralympic events and one containing details of National Olympic Committee country code.

Data set fields

Event

Each row represents the details for a Paralympics event.

Attribute	***Example***
Host city	Tokyo
Year	1964
Country	Japan
Region	JPN (3-letter code representing the region)
Type	Summer
Start	1964-11-08
End	1964-11-12
Participants (M)	195.0 (total male participants)
Participants (F)	71.0 (total male participants)
Participants	266 (total participants)
Duration	4 (number of days the event lasted)

Region

Each row represents one region as defined by the National Olympic Committee.

Attribute	*Example*
NOC	JPN (3-letter code representing the region)
region	Japan, the region’s name
notes	any notes relating to the region, typically blank

Application design

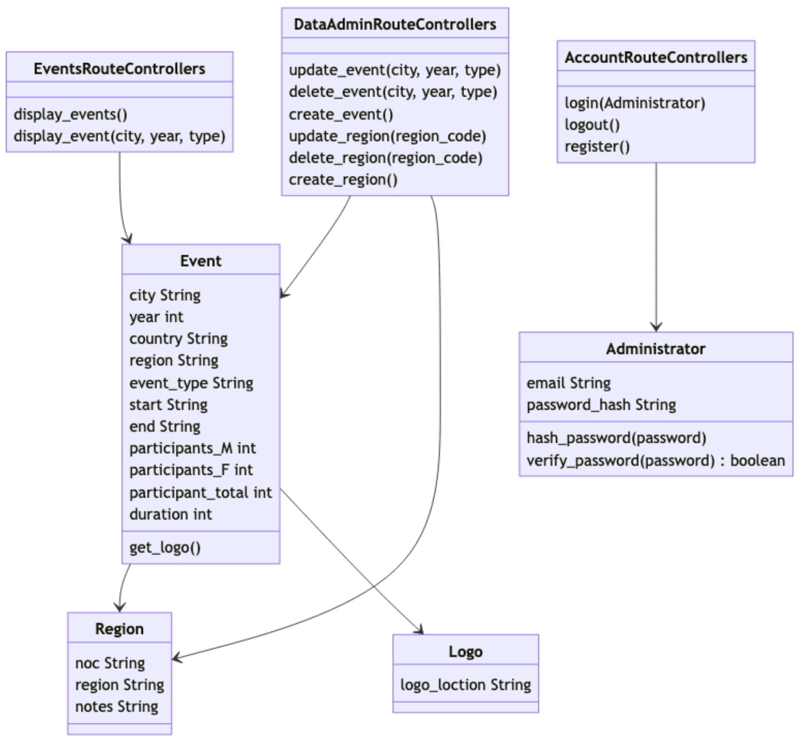
The following is one possible design for the Paralympics web app that provides information on different events and allows administrators to manage the underlying data. You may not agree with the design, for example:

- should the functions for the routes for Event and DataAdmin be split into separate modules? Is cohesion greater if the event related functions are grouped together?
- should the functions for login, logout etc be part of the Administrator class?

To uniquely identify an event you currently need to know city, year and type.

The class shape has been modified to represent a Python module for the ‘Controllers’.

Optional parameters for the URL are indicated using ‘ though could also have been shown as ‘?paramname=’. Again, this is not textbook use of the class diagram shape but is adapted to allow us to display useful information.



List of routes:

Route	***HTTP***	***Description***
/login	GET, POST	GET returns the login form POST Takes the login form values and validates against administrator records in the database and returns the data management page.
/logout	GET	Sets login status to False and returns to the index page
/register	GET, POST	GET returns the form to register a new accont POST Takes the login form values and if valid, creates a new administrator records in the database.
/admin/event	GET	Returns a page of all events with the option to update, delete or add new event
/admin/region	GET	Returns a page of all regions with the option to update, delete or add new region
/event/	GET	Displays a page with the details for one event
/event/	GET	Displays a page with the summary for all events
/event/update	GET	GET Displays a form of fields for an event that can be edited POST Takes the changed parameter values from the form and updates the database
/event/delete/	DELETE	Deletes the identified entry from the database
/event/add	GET, POST	GET Displays a form of fields for an event that can be edited POST Takes the changed parameter values from the form and updates the database
/region/update	GET	GET Displays a form of fields for a region that can be edited POST Takes the changed parameter values from the form and updates the database
/region/delete/	DELETE	Deletes the identified entry from the database
/region/add	GET, POST	GET Displays a form of fields for a region that can be edited POST Takes the parameter values from the form and adds a new region the database

Activity 1: Conceptual design

Present the results of the stage as an ERD. The ERD format is explained in the database design guide.

1. Identify the entities and attributes. You the data set detail and the architecture diagram. If you did not have an architecture diagram then you could use the data driven design approach and identify nouns and adjectives from the requirements to identify potential entities and their attributes.
2. Draw the entities and their attributes as rectangles.
3. Identify the relationships between identities and draw lines between the entities to represent these.
4. Identify the multiplicity (one-to-many, many-to-many) and draw these on the relationship lines.

You can do this on paper, on PowerPoint/Word, or there are links to a few free online tools in the guide. The diagrams in this tutorial document were created using a Mermaid plugin for PyCharm. Mermaid creates diagrams in markdown.

Activity 2: Logical design

Document the results of this stage, the 'schema', as an ERD; though you may need to add additional details that the ERD doesn't allow.

Read the database design guide section on normalisation and logical design.

1. Are there multiple values in any of the fields/attributes?

In this design, none of the fields hold multiple values so there are no changes to be made.

If there were, you would need to add a new table.

2. Identify the primary key for each table.

Primary keys can be composite (ie more than one attribute or column taken together); however this is less simple to implement in SQLite and will be easier in term 2 if you introduce a single unique identifier in your design.

Points to consider:

- The unique identifier for an event is currently a combination of city, year and type.
- Is email address unique? It could be a unique identifier for the Administrator, however long character strings or variable length are generally considered a computationally inefficient key field. An integer that can be autoincrement is typically used.
- The region field has a 3 character unique field, the NOC code. This is short and uniform and is likely reasonable to use as the primary key.

Add 'PK' next to the primary key attribute(s) for each table to show which is/are the primary key.

2. Where there is a related entity (table) add the foreign key

The foreign key for the 'many' side of the relationship will be the primary key field from the 'one' side.

Add the field to the table and add 'FK' next to it.

3. Are there attributes that depend on more than just the primary key?

'Country' is in both tables. In the Event table this also determined by 'region' and 'event_id'. It may be better to remove this field from Event and maintain it in Region only (in event the data would be repeated in multiple places, in Region it is held only once).

Removing 'Country' improves the database design but may complicate your application design as you will need to query both Event and Region to get the data for each event.

There is no 'right' answer to this. In the paralympics data it may be better to leave the country in as the value for country varied from the official NOC region name (e.g. UK, United Kingdom, Great Britain).

4. Indentify the data types

SQLite data type definitions are flexible, that is SQLite will attempt to match the provided data type in the data to the column definition.

SQLite will support most of the variants of SQL data types but will store the data type as one of the following:

- NULL. The value is a NULL value.
- INTEGER. The value is a signed integer, stored in 0, 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
- REAL. The value is a floating point value, stored as an 8-byte IEEE floating point number.
- TEXT. The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE).
- BLOB. The value is a blob of data, stored exactly as it was input.

Consider which to use for each of the fields. Add the data type to each attribute in the ERD.

5. Are there any constraints?

Following are commonly used constraints available in SQLite:

- NOT NULL – Ensures that a column cannot have NULL value.
- DEFAULT – Provides a default value for a column when none is specified.
- UNIQUE – Ensures that all values in a column are different.
- PRIMARY KEY – A combination of a NOT NULL and UNIQUE. Uniquely identifies

each row in a table.
- CHECK (expression) – Ensures that all values in a column satisfies certain conditions.

Review each field and decide which constraints need to be added. Add any constrains to your ERD.

Review and iterate

Review the design and consider if any further changes are required.

In the example conceptual diagram there is a separate 'Logo' entity that will be used for the event. Paralympic logos have an image file and the logo may also have a name. These are usually specific to each event. This could be added as a new attribute to the event table rather than a new table.

The application design has assumed the logo will be stored in a file system and the location of the file will be passed. It is possible to store images in a database as a BLOB. Search and you will find many articles on the merits of storing images in a database or a file system. Read a few articles, understand the tradeoffs and make a decision!

Activity 3: Review the application design

Now that you have designed the database, and your app will use the data from the database rather than the .csv file, then the introduction of the 'event_id' field should simplify some of your functions as you no longer need 3 fields to be able to identify an event.

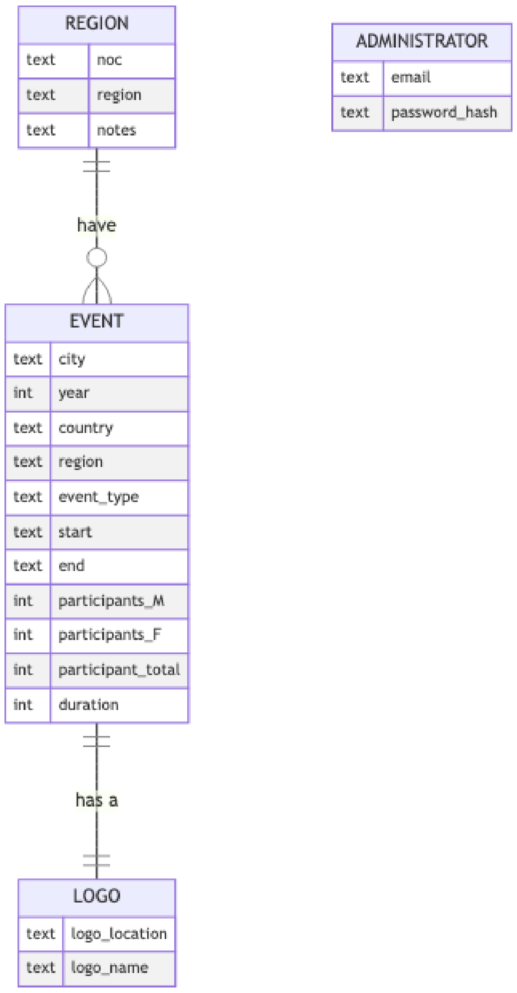
Depending on your choice of how to handle the logo files, you may also change the logo class.

Potential solutions

Conceptual design

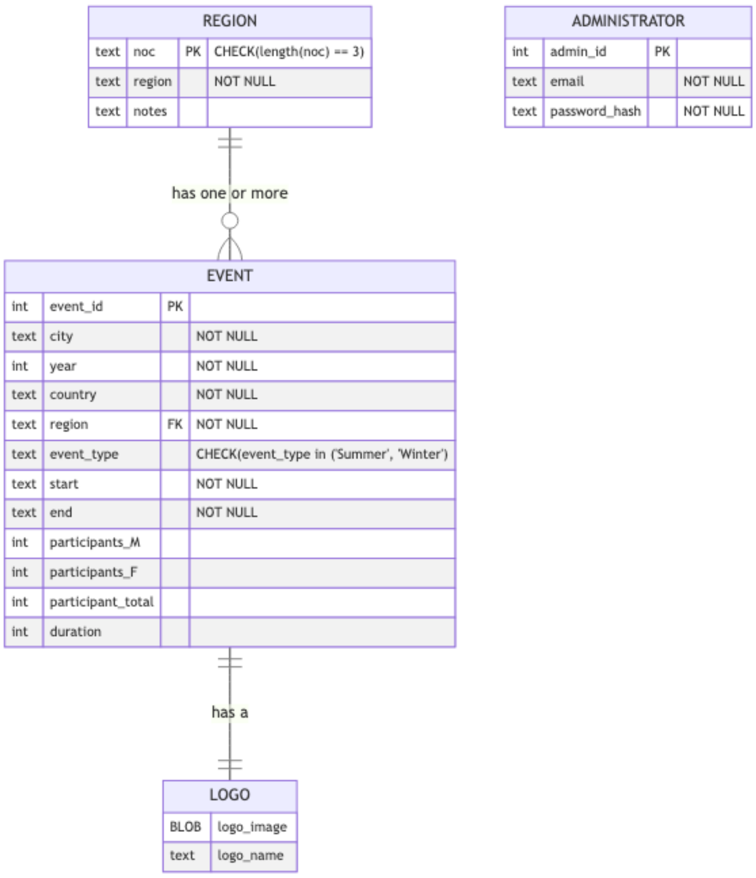
You may have only noted the attributes and not the data types. The limitations of the tool used to create the diagram mean that the data type had to be added. These datatypes refer to the SQLite data types, not the Python data types shown in the application design diagram.

Most diagrams also show the attribute name before the data type.



Logical design

Depending on your choices, yours may look different to this.



REGION			
text	noc	PK	CHECK(length(noc) == 3)
text	region		NOT NULL
text	notes		

ADMINISTRATOR			
int	admin_id	PK	
text	email		NOT NULL
text	password_hash		NOT NULL

EVENT			
int	event_id	PK	
text	city		NOT NULL
int	year		NOT NULL
text	country		NOT NULL
text	region	FK	NOT NULL
text	event_type		CHECK(event_type in ('Summer', 'Winter')
text	start		NOT NULL
text	end		NOT NULL
int	participants_M		
int	participants_F		
int	participant_total		
int	duration		

LOGO	
BLOB	logo_image
text	logo_name