Assignment 1

Stage 2 (ER to Relational Mapping)

Introduction

This document contains the standard ER design for Stage 2 of Assignment 1. You must convert this design into a PostgreSQL relational schema (a collection of `create table` statements) and submit it via the Assignments link on the course web site. In performing the conversion from the ER design to a relational schema, you should follow the approach given in the lecture notes on "ER to Relational Mapping".

Submission

Submission : Through WebCMS3 submission link. You will submit a single SQL file called ass1.sql.

Deadline : Thursday 4th July 9pm.

Late Penalty: Per course outline.

PostgreSQL version 9.4 online documentation

The main index page to PostgreSQL 9.4.x Documentation

Requirements on your Submission

The schema you submit will be marked by a program (auto-marked). In order for the program to recognise what you've done as being correct, your SQL **must adhere to the following requirements**:

- all tables must have an appropriate primary key defined; all foreign keys must be identified
- use appropriate domains for each attribute (e.g. a birthdate would be done as an SQL `date`, a counter would be done as an SQL `integer` constrained to be ≥ 0)
- if an attribute is a string, and no maximum length is specified, use PostgreSQL's (non-standard) `text` type; otherwise, use an appropriate `varchar(N)` type or one of the supplied domain types
- if an attribute is a boolean value, use the SQL `boolean` type
- wherever possible, not-null, unique and domain constraints must be used to enforce constraints implied by the ER design
- derived (computed) attributes should not be included in the SQL schema
- wherever possible, participation constraints should be implemented using the appropriate SQL constructs
- map all of the entity class hierarchies in the ER design using the **ER-style** mapping (i.e. one table for each entity class).
- all relationships should be mapped using the approaches described in the lecture notes; in particular, you should avoid "over-generalising" your SQL schema relative to the ER design (e.g. a 1:n relationship should *not* be mapped in such a way that it can actually be used to form an n:m relationship)

Since the assignment is going to be auto-marked, it is very helpful if you use the names that the auto-marker expects. Please follow as much as possible the following naming conventions:

- each table that represents an entity should be given a name which is the "pluralised" version of the entity name (e.g. entity `Person` is mapped to a table called `People` and entity `Event` is mapped to a table called `Events`)
- each table that represents a relationship can be given the same name as the relationship in the ER diagram
- each data attribute in the SQL schema should be given the same name as the corresponding attributes in the ER
- if an attribute in the SQL schema is derived from a relationship in the ER diagram, name it after the relationship (suitably modified to make sense, e.g. if the relationship is `owns` and the attributes is in the table for the entity that is being owned, then you would changed the name to `ownedBy`)

- when mapping multi-valued attributes, name the new table by concatenating the entity and attribute names
- when mapping composite attributes, use the names of the "leaf" attributes
- if names in the ER diagram contain multiple words, concatenate them into a single word in `camelCase` in the SQL schema

Note: if the name you want to use clashes with a PostgreSQL keyword (e.g. `user`), you will need to write the name in double-quotes (i.e. `"user"`) and in all lower-case

Place the schema in a file called **`ass1.sql`** and submit it via WebCMS (see above) before the deadline. To give you a head-start, a template for the schema is available, which has (parts of) some of the required tables already defined. Note that you will need to add more tables, as well as filling out the attributes in the supplied tables. Your submission must follow this format, so save a copy of this and edit it to produce your submittable `ass1.sql` file.

The reason for insisting on strict conformance to the above is that your submission will be auto-marked as follows:

- we will create an initially empty database (no tables, etc.)
- we will load your schema into this database
- we will usa a script to compare your schema with the expected schema

The comparison will make use of the meta-data which has been added to the database by loading your schema. Needless to say, if you schema has load-time errors, then it's not going to be possible to compare it against the correct version. Therefore it is essential that you check that your schema can load into an *initially empty* database before you submit it.

Following the instructions above is considered to be a requirement of this assignment. If you stray from the expected schema, your submission will be marked as incorrect. Our auto-checking scripts have a little flexibility, but not much, so don't rely on it.

Please don't try to second-guess or "improve" the standard design below. Even if you think it's complete rubbish, just translate it as given. If you think that it's incorrect or that the information supplied isn't enough to do the mapping unambiguously, post a message on the course MessageBoard for clarification Also, if you want to give opinions on the standard schema use the "Assignment 1" topic on the course MessageBoard. But we won't update the standard ER given as the spec for Stage 2.

Standard ER Design

This ER design gives one possible data model for the `et.org` application introduced in the first stage of this assignment. The design here is based on the discussions on the MessageBoards, on my experience with EventBrite, and on my interpretation of the more ambiguous aspects of the requirements. This isn't necessarily the design that would be used in practice and may not even follow all of the requirements from Stage 1 precisely. It has been designed to make Stage 2 of the assignment more interesting (i.e. to give you experience with a range of modelling constructs and translation mechanisms).

To make the presentation clearer, the design is broken into a number of sections. Note that an entity will have its attributes and class hierarchy defined exactly once. If an entity is used in a later section of the design (e.g. to show relationships), it will simply be shown as an unadorned entity box (and you should assume all of the attributes and sub/super-classes from its original definition).

The development of any significant design requires assumptions. Assumptions specific to particular entities and relationships are presented below each diagram.

A general strategy used in the design is to introduce a numeric primary key called `id` into all major entities. This is despite the fact that we could have made a primary key from existing attributes in many cases (e.g. `email`). The reason for doing this is that primary keys typically end up as foreign keys in other tables, and thus their values need to be copied to many places in the database. "Natural" keys (such as `email`) are strings (typically 40-60 bytes), whereas numeric keys are 4-byte integers, so there is a clear space saving in maintaining copies of smaller keys. Using numeric keys also makes indexing and various query processing techniques faster. One disadvantage is adding an "extra" attribute into each table.

Other notational conventions in the ER diagrams:

- primary key attributes for entities are underlined

- total participation in a relationship is indicated by a thick line
- an arrow indicates that at most 1 entity is involved in the relationship

Note that the data here is sufficient to allow the `et.org` site to be built. Some notions mentioned in the Stage 1 requirements are related to the working of the application and do not need to be explicitly modelled here. Actions (e.g. adding a person to a contact list) typically do not have a presence in the data model either, although they clearly affect the data in the database.

Data Types

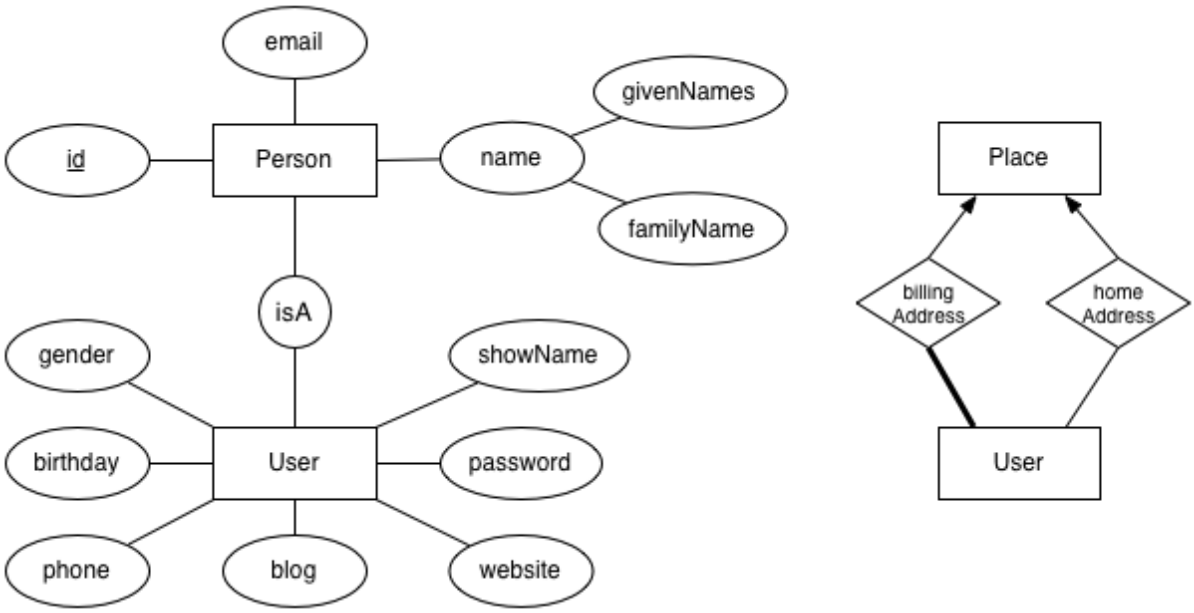
To make your life simpler, I've defined some useful data types using the `create domain` statement. Some of the `create domain` statements use standard SQL patterns for specifying constraints, while others use PostgreSQL-specific regular expressions for this prupose. The domain definitions are given at the top of the template file.

You shouldn't need to use many `varchar(N)` types in this assignment. The above types ought to be sufficient for most of the fields in the database. Use them whereever you think it's appropriate.

You can ignore the `PrivacyValue` type; it is not relevant for this assignment.

Users and People

The following diagram shows the entities, attributes and relationships that provide the information about people on the **et.org** site.

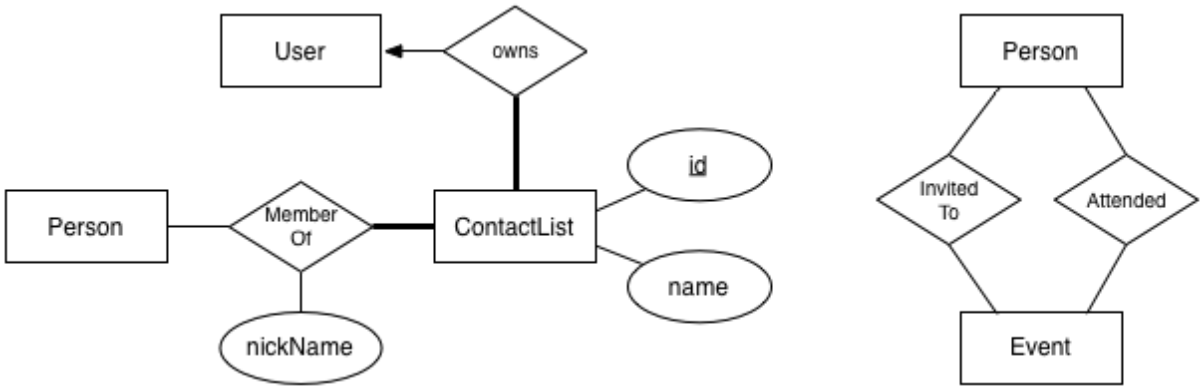


Comments:

- we use a numeric ID as a primary key, since People and Users will be extensively referenced in the database
- for every person in the database, we need to know their email and their given-names
- information which every user is required to provide (as well as their person data): password and billing address
- users can provide a name for the system to display them as; if none is supplied, `et.org` will form a name from the family- and given-names
- note the use of `isA` in a circle to indicate that the `Person` entity has only a single sub-class (`User`); remember that you must implement this (very small) class hierarchy via the ER-style mapping
- names are typically no longer than 50 chars (for the given- and family-name components) and less than 100 chars for full names
- a user's blog and website are both given as URLs

Contact Lists and other People relationships

The following diagram shows entitites, attributes and relationships that deal with various groups of people on the **et.org** site.

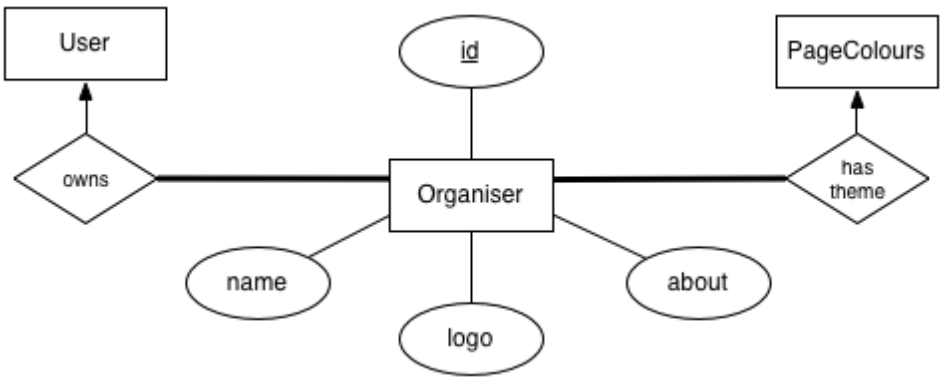


Comments:

- users can build contact lists for easy reference to groups of people
- every contact list must have a name; contact lists cannot be empty
- all members of a contact list must be entered into the database as people
- the owner of a contact list might want to refer to a list member by a different name to the one in their `People` entry, and so can define a nickname for them in that list
- contact lists can be used to generate invitations to events (doing this will produce a collection of invitations, one for each person on the list)
- we can also record attendees at events, but only for people who are already known to the system.

Organisers

The following diagram shows entities, attributes and relationships relevant to event organisers on the **et.org** site.

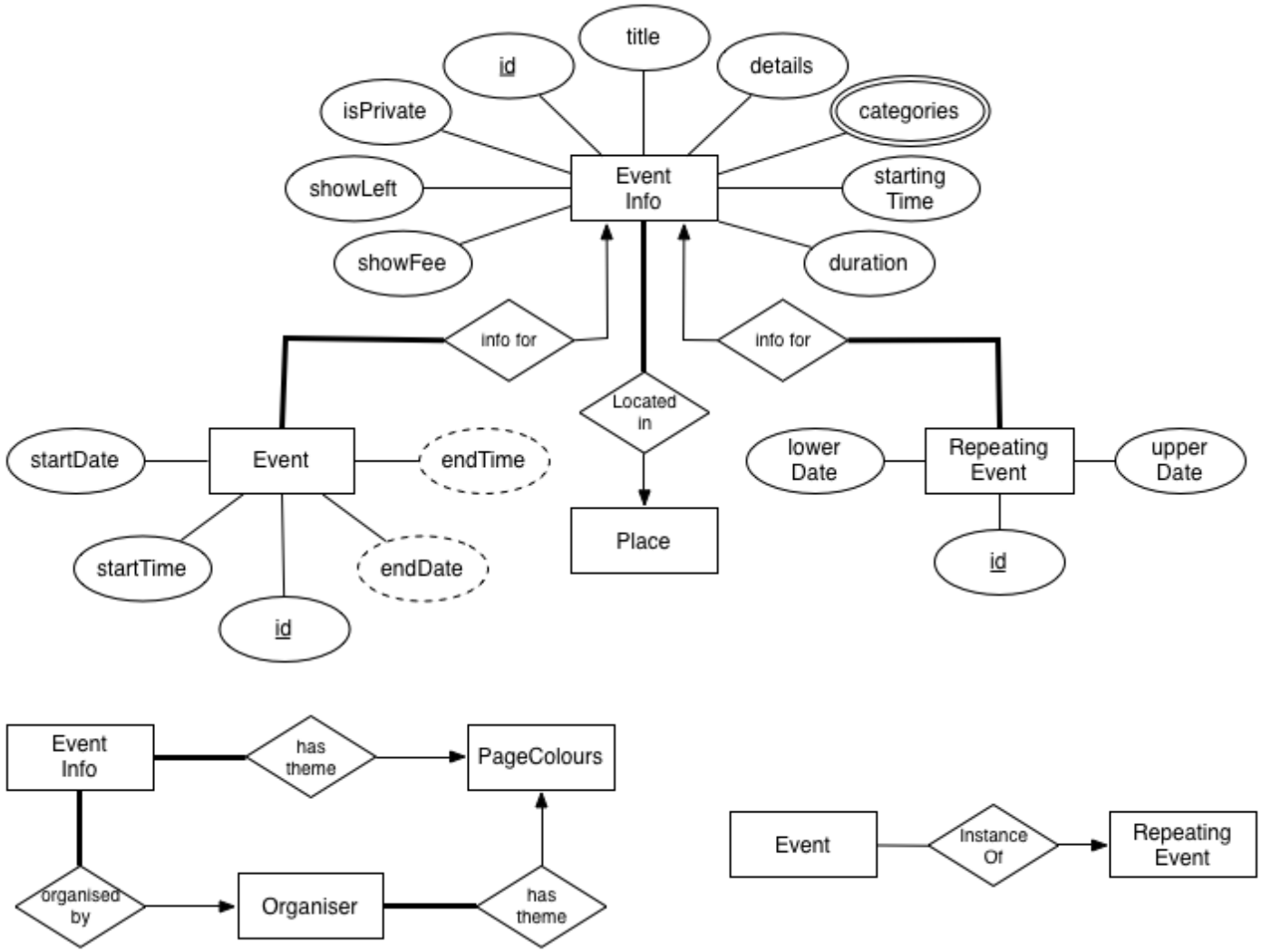


Comments:

- organisers are created by users for the purpose of being the "front facing" contact for events
- every organiser must have a name; each organiser may also have a logo and some descriptive material
- organiser names may be moderately long (up to 100 characters)
- logos must be JPEG images and are stored in the database as `bytea` values (see the PostgreSQL documentation for details on `bytea`)
- the descriptive material on an organiser can be arbitrary text (most likely HTML)
- the system generates a page for each organiser
- every organiser is required to specify a theme which gives the colour scheme for their page

Events

The following diagram shows entities, attributes and relationships for the events that are managed in the **et.org** site.



The design for Events is complicated by the fact that we have both one-off and repeating events. Repeating events effectively define a series of one-off events recurring at regular intervals. Each instance of a repeating event shares most of the same information (e.g. title, details, privacy) as defined for the repeating event. To avoid duplication, the common event information is moved to an EventInfo entity and both the repeating event and event instances refer to this. An Event (either a one-off or an instance of a repeating event) thus has some of its information in the Event entity and some of its information in the EventInfo entity. Note that this is not a class hierarchy, because EventInfo entities, Event entities and RepeatingEvent entities have their own identities.

Comments on EventInfo:

- `EventInfo` entities define the core information for both one-off events and repeating event series
- this means that all events in the series will have the same common data (e.g. title, details, location); the only thing that may vary is the actual time that the events occur
- `EventInfo` entities are required to have a title (up to 100 chars), some details (arbitrary text), a location and a starting time
- a duration may also be specified to indicate how long the event runs (implement this as a PostgreSQL `interval` value; see the PostgreSQL documentation for details);
- events have a set of boolean flags to indicate how/whether they should be displayed: `isPrivate`, `showLeft`, `showFee`
- `showLeft` indicates whether the event page will show the number of tickets remaining to be sold
- `showFee` indicate whether the service fee charged to the organiser will be included in the ticket price shown to purchasers
- each flag is required to be set; by default, events are private, do not show remaining tickets and do not show the service fee in the ticket price
- each event may be tagged with some categories to identify what kind of event it is; categories are simply short text strings like "music", "food/wine", "theatre", "festival", etc.
- ticket types will also be linked to the `EventInfo` entity, meaning that each instance of a repeating event series has the same ticketing structure

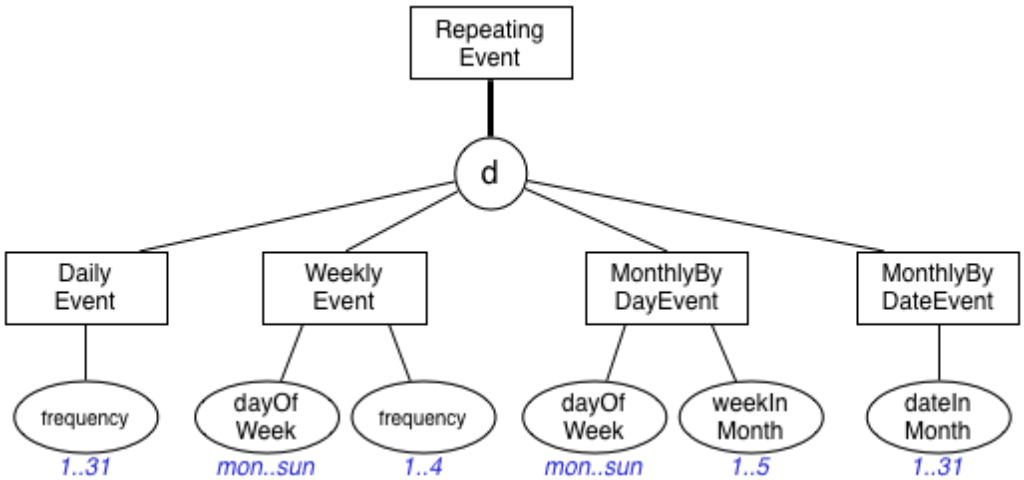
Comments on Events:

- the `Event` entity corresponds to a specific happening at a given place and given time
- the core information for an event is contained in its associated `EventInfo` entity
- an `Event` which is an instance of a repeating event will be associated to a `RepeatingEvent` entity; a one-off event will not refer to any repeating event
- the `Event` entity serves primarily to hold timing information for the event
- events must also have a starting date and time specified
- for display purposes only, we might compute an end-date and end-time based on the start-date/time and the event duration; if no duration is given, we simply display the start-date/time details

The date/time information in an `Event` is determined as follows:

- for an event which is a genuine one-off event ...
  - the startDate will be provided by the organiser
  - the startTime will be set using the starting time in the associated `EventInfo`
  - the endTime and endDate will be computed using the startDate and startTime and the duration specified in the `EventInfo`
  - if no duration is given, the endDate and endTime will be null
- for an event which is an instance of a repeating event series ...
  - the startDate will be calculated according to the repetition information
  - the startTime will be determined from the `EventInfo` starting time
  - the endDate and endTime will be computed from the `EventInfo` duration, if supplied

Note that the organiser can override the startTime for a repeating event instance if one event happens to start at a different time. This is the reason we have startTime in the `Event` entity, rather than simply using the `EventInfo` starting time



Comments on Repeating Events subclasses:

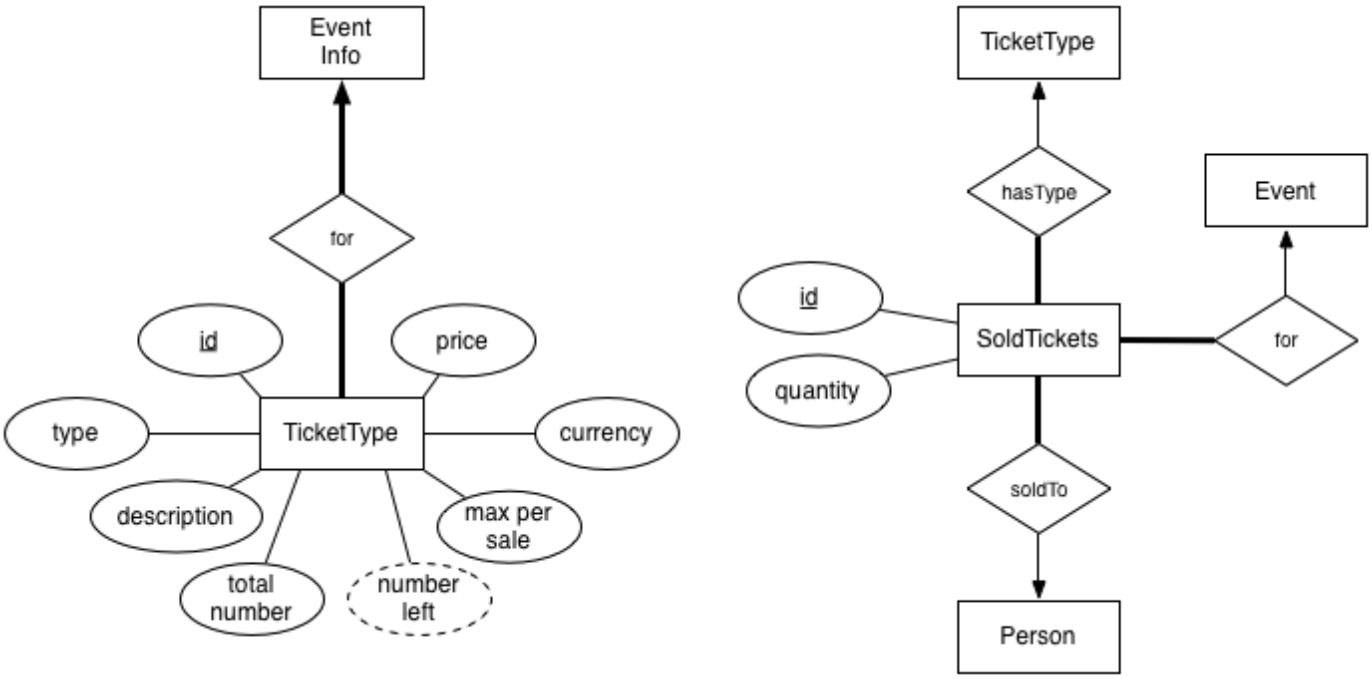
- a series of similar events is specified via a `RepeatingEvent` entity
- various common styles of repetition are supported via the subclasses of `RepeatingEvent`
- both the lowerDate and upperDate must be supplied, and these give lower and upper bounds on the days when instances of the repeating event can occur
- the first event in the series will occur on or after the lowerDate and the last event in the series will occur on or before the upperDate
- for a daily event, we specify whether it repeats every day, or every two days, etc.; if an event repeats every 32 days or more, we may as well make it a monthly event
- for a weekly event, we specify which day of the week the event occurs on and how many weeks between recurrences (e.g. "every 2nd Tuesday")
- there are two kinds of monthly repetition: by date or by day+week
- for monthly-by-date (e.g. "on the 12th of each month"), we simply give the date in the month when the event occurs
- for monthly-by-day (e.g. "on the third Tuesday of each month"), we give which day of the week and which week in the month when the even occurs

When a `RepeatingEvent` entity is defined, all of the `Event` instances determined by the repetition are generated. If the `RepeatingEvent` entity is subsequently modified, the `Events` will be regenerated

(possibly resulting in sold tickets for any deleted events needing to be refunded). Note that there are some repetition cases that cannot be represented in this scheme (e.g. "the third Tuesday in every fifth month").

Tickets

The following diagram shows entities, attributes and relationships on tickets in **et.org**.

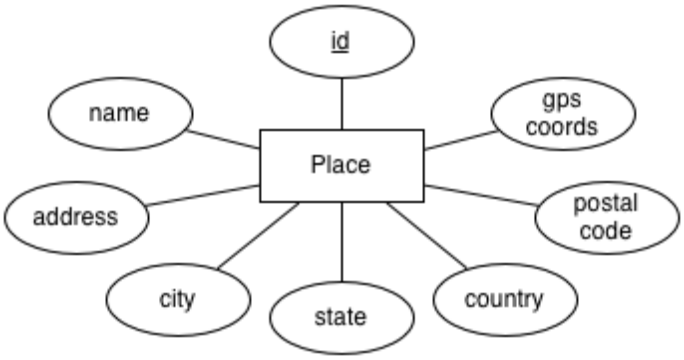


Comments:

- individual tickets are not directly represented in `et.org`, but rather we represent ticket types (ticket classes) and ticket sales are represented as a block of tickets of a given type
- each ticket type is associated to an `EventInfo` entity; several ticket types will typically be associated with each `EventInfo`
- a ticket type is defined by a type name (e.g. "First Class", "Standard", "Mosh Pit", etc.) and may have an accompanying description (up to 100 chars)
- for each ticket type we also need to record the total number available, and the maximum number of tickets that can be purchased by a user in a given puchase (if users are greedy they can simply make multiple purchases)
- each ticket type has a quoted price, which is specified in a particular currency (although since sales happen outside **et.org**, customers can preumably purchase using a different currency)
- currencies should be given using the standard ISO4217 3-letter currency codes
- a ticket sale indicates that some tickets of a certain type have been sold to a given person for a given event
- a ticket sale is represented by a record which has a unique ID, is linked to the specific ticket type and has a quantity of tickets attached to it
- ticket sales are linked to a person; the purchaser would be given the sales ID as part of the URL that they can use to examine their purchase, even if they're not a user who can log in to the system
- if the system wants to provide printing, it could use the sales ID plus sequencing numbers to generate a set of unique ticket numbers
- for display purposes, we may also wish to compute the number of tickets remaining in each ticket type; this could be derived from the total available tickets and the quantity in the ticket sales

Places

The following diagram shows entities, attributes and some relationships related to location information within the **et.org** site.

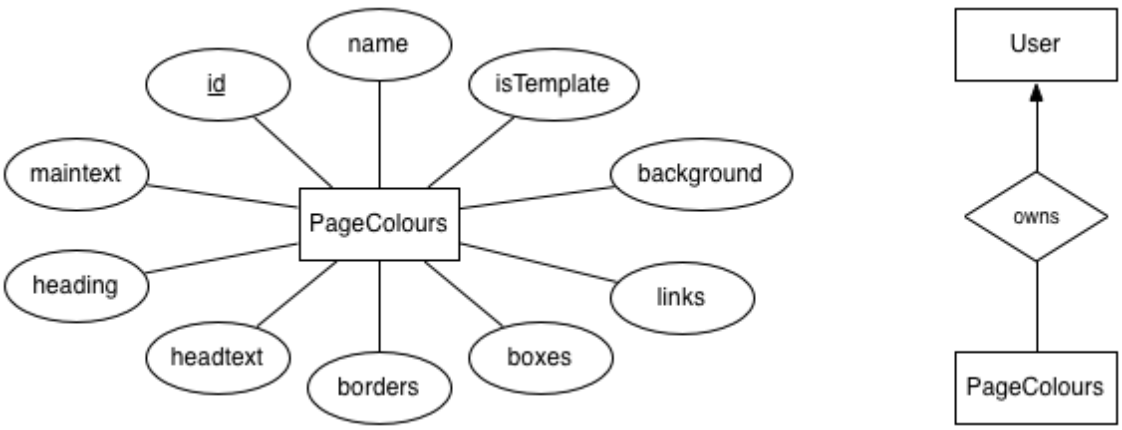


Comments:

- `Places` are used for a range of location/address scenarios
- each place record must have an ID and a name (text string up to 100 chars) and a country name (text string up to 50 chars)
- apart from these required fields it may have any other combination of the attributes specified or not specified
- this allows us to represent complete postal addresses (with street address, city, state, postcode and country) if needed
- alternatively, an organiser might just specify a name (e.g. "Acer Arena") and a city (e.g. "Sydney") (and, of course, a country) to identify where the event is held
- it is also possible to optionally provide GPS coordinates in the form used by Google maps; if these are provided, then a map can be displayed with the event location pinpointed
- every event has to be associated with a location, but the description of the location can be as vague or detailed as the organiser wishes
- becuase of their flexibility, `Place` records are also used to handle addresses associated with users

Page Configuration

The following diagram shows entities and attributes for page settings that are used to control the appearance of web pages on the **et.org** site.



Comments:

- a table of page configurations is stored, where each configuration setting specifies the colour setting for some component of the page
- some of these page colour records can be made publically available as templates and users can link to them when setting up themes for organisers and events
- users can also create their own entries in this table to develop custom settings
- page configurations are not treated as templates by default
- as well as having values for all colour settings, page configurations must have a name specified (even if only used by one user)
- page configurations are owned by the user who created them, except for system templates which are owned by nobody
- users can specify their own themes as templates, in which case other users can make use of them

Hint: if you keep the table declaration order given in the template, there will be a forward reference to the `Users` table from within the `PageColours` table. You can deal with this either by re-ordering the table declarations or by using an `alter table` statement to add the foreign key constraint to the `PageColours` table after you have created the `Users` table.

What To Do Now

Make sure you read the above description thoroughly, and review the notes and exercises on ER-to-relational mapping. Grab a copy of the ass1.sql template and see what's provided there. If any aspect of this design requires further clarification, ask for it under topic "Assignment 1 (Stage 2)" on the course MessageBoard, or email me personally if it might give away some of the solution.

Reminder: before you submit, ensure that your schema will load without error if used as follows:

```
% dropdb ass1
% createdb ass1
% psql ass1 -f ass1.sql
... will produce notices, but should have no errors ...
% psql ass1
... can start using the complete database ...
```

If I have to fix errors in your schema before it will load, you will incur a 1 mark "administrative penalty".

Have fun, *Helen*