Brisbane Olympics

Scenario

Brisbane is hosting the world's simplest Olympics!

In the spirit of Simplification + Time Saving, Brisbane are using a minimal database and python backend to schedule an Olympics. The database consists of types of Person (Athlete, Official), types of Location (Residential Village, Sporting Venue) and their various relationships.

A python query file adds extra validation to sql queries to smbolise "business logic" that normnally lives outside the database and in the application layer.

Design Choices

Uniqueness, deletion, immediate and deferred constraints:

Uniqueness constraints are included for two types of reasons. Attributes that are obviously unique, like a person's email, or for limiting the role a single person or location can perform. The foreign key referring to person in the athlete and official tables is unique. We assume officials only perform one role (for multiple events) during the entire Olympics, as both judging and refereeing are highly skilled roles. It is not realistic for one person to be a world class gymnastics judge and a world class handball referee. Similarly, the same location cannot be a venue twice or village twice.

On deletion of key records like a person, athlete, official, or event, there is no need to keep track of any of their acitivies (eg. booking trips). Meanwhile supplementary records like vehicles and venues should not be possible to delete while they are in use, as many people depend on them. Only foreign keys refering to key tables are set to ON DELETE CASCADE.

Logical constraints (eg. trip duration > 0) are immediate, while referential constraints (eg. foreign keys) are deferred to accommodate bulk transactions.

An additional attribute has been added to distinguish between sporting venues and accomodation villages. Accomodation villages have a capacity of residents, and venues have a type (eg. stadium). Since postgres doesn't allow subqueries within CHECK expressions, it isn't possible to simply add a type attribute in location (village/venue) and check that only appropriate locations host events or accomodate people. Adding these additional attributes allows us to create separate "Venue" and "Village" tables which participate in appropriate relationships.

Business logic constraints:

Some constraints are so nuanced and variable that they are better handled by the application or end user. These are listed below.

Country codes are stored as 2-character codes. Validation of country codes is omitted for brevity. It is likely to come from an application, as not all countries participate in each Olympics. There are also many controversial areas on many continents, so the list of 'valid' country codes is best left to the application and the organisers of the next Olympics.

Although each contingent will be allocated one village for the entire Olympics, we chose to not enforce this constraint. This is better handled by the application or end user. We considered adding an attribute in the Village table to keep track of which continent lives there, but finally decided it is not the role of the database to say which particular contingent lives in in which particular village. There are typically 200 countries participating, and likely a few contingents will share each village, or a few villages. This is more realistic than demanding 200 new villages be built in Brisbane. Individual people may also need to swap villages in exceptional cases, eg. emergency maintenance. So while it is known that a contingent usually stays together in one village, this is not enforced in the database level for end-user convenience.

Similarly, allowing the application or end user to control event scheduling allows for much more flexibility in event scheduling than enforcing any constraints at the database level. Event duration, interval required between events, and appropriate time of day depends on many factors and are best left to the application layer. Eg. a dressage stage would require a lot of work to prepare between events, while a track doesn't. Marathons and sprints require different running times. No event runs at 3am, but there are some late-night and early morning events, and some sports may run into overtime, while other sports do not, so the exact cut-off times are variable. Allowing the application or end user to control this allows the organisers to customise constraints with greater versatility.

Athletes and officials booking themselves on to trips is modelled as an aggregation of two relationships. Each vehicle has a capacity N, so each trip with that vehicle can only be booked by N people. To enforce this we could add a "number of current bookings" attirbute and a TRIGGER in the Trip table which updates the current number of bookings and ensures it doesn't exceed the vehicle capacity. We preferred to leave this up to the application or end user, as raising exceptions could cause the end user inconvenience, and triggers were not explicitly listed in the assignment document.

A sanity check that the duration of a trip is positive is enforced, but trip length is not constrained further. While any trip between the same two locations should have a similar duration, this is still variable due to weather, traffic, etc. The origin and destination could conceivably be the same in case of scenic tours for athletes who have completed all their events.

Derived attributes:

Derived attributes are included in the ERD. A person's age is trivially derived from DoB, so there is no need to include this in the RM diagram or SQL, as any query can calculate this from DoB, however it is still shown in the ERD. A location's postcode is derived from its suburb (and/or region), but we include it in the RM diagram and SQL table since this is not a trivial calculation that a query can perform.