UNIVERSITE LIBRE DE BRUXELLES January 22, 2013
Faculté des Sciences Appliquées

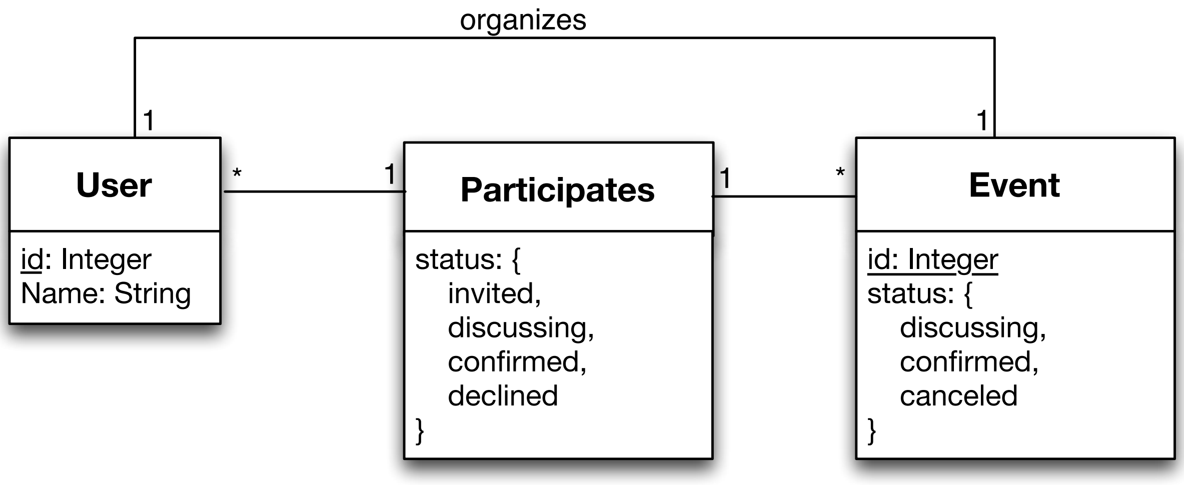
INFO-H-415 Advanced Database Management Systems
January Exam

The exam is divided in four sections. All sub-questions are worth approximately the same amount of points. However, some of these will only take you a minute, some require a bit more thinking, and a couple would require weeks to answer perfectly. Make the best use of your time.

1 Active Databases (3 points)

You're designing a database for a personal event organizing tool. It allows users to organize events ("Indian dinner", "Weekend in Paris", "Rihanna Concert", etc.) by inviting their friends and discuss the organization. (e.g. choosing a time and place for the event) The discussion involves participants proposing alternatives (e.g. "my place", "last week-end of february", "this tuesday"...) and voting on them. After some discussion the organizer makes the final choice, confirms the event and asks all participants to confirm their attendance. Now that the parameters of the event are well-known, each participant has the opportunity to confirm or decline his participation.

Here is a partial diagram of part of the data model for the application:

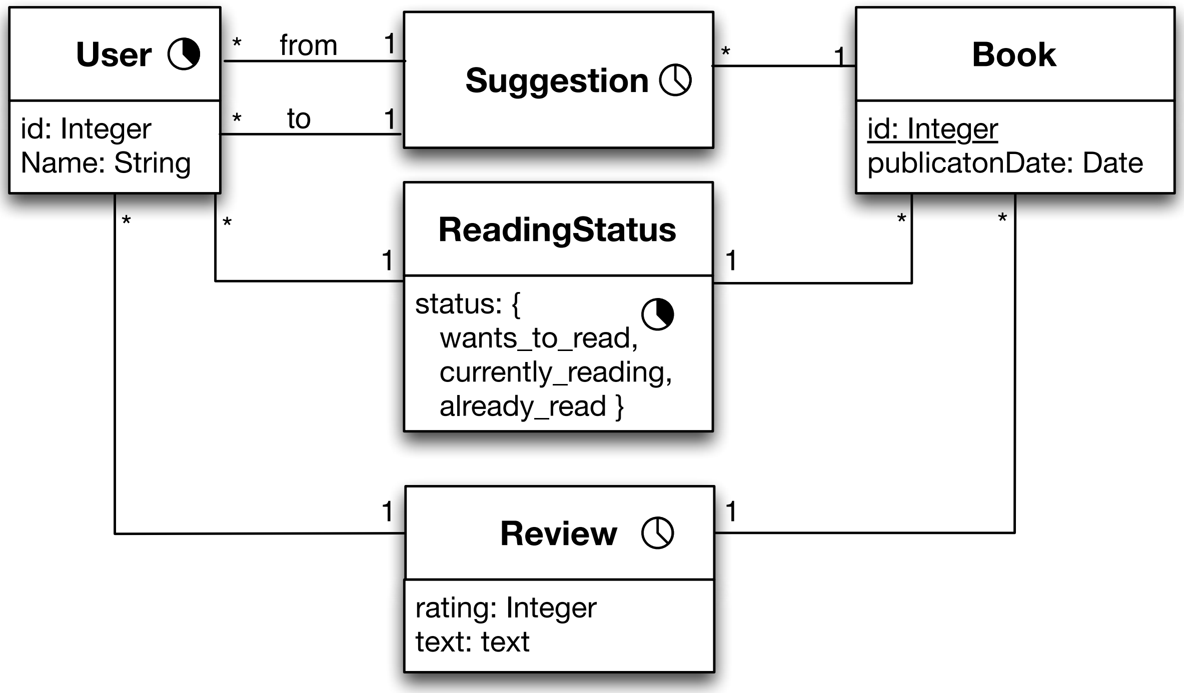


Describe the triggers needed to enforce the following integrity constraints. Whenever multiple triggers are needed to enforce a single integrity constraint, list all of them, but write the code in full for only one of them. Throughout the entire question you should provide at least one example for each of ON INSERT, ON UPDATE, and ON DELETE triggers :

1. Participants are always created "invited".
2. Participants can't come back to being "invited" from another status.
3. Participants can't confirm their participation to an event that is not itself confirmed.
4. Participants can't change their status for a confirmed event.
5. The organizer can't revert an event to "discussing" if some users have already confirmed.

2 Temporal Databases (7 points)

You've just joined wellread.com, a web startup that helps users keep track of the books they read and want to read, suggest books to each others, rate and review books they've read. Here is a conceptual temporal schema for this website:



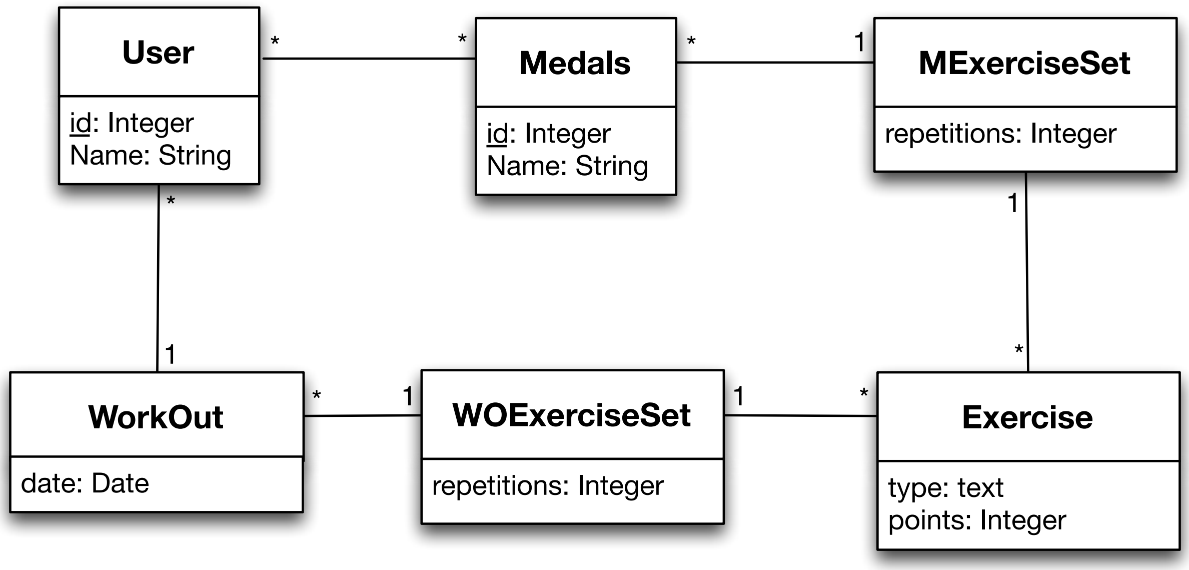
Reminder: The "empty clocks" denotes things that occur at one point in time, while the "full clocks" denotes things that occur over an interval of time.

1. Define a relational model corresponding to this conceptual model. (Note: many possibilities exist. Write down all assumptions you need to make.)
2. Write the code ensuring the following integrity constraints. Whenever multiple triggers are needed to enforce a single integrity constraint, list all of them, but write the code in full for only one of them. Throughout the entire question you should provide at least one example for each of ON INSERT, ON UPDATE, and ON DELETE triggers :
   (a) A User can only suggest a book during his lifecycle
   (b) A User can't write a review unless he's already read the book
   (c) After a User has read something, he can't want-to-read it anymore
3. Write the following SQL queries :
   (a) List all books that User "Jean Valjean" wants to read
   (b) List all books nobody has finished reading yet
   (c) List the currently unpublished book with the most suggestions
   (d) List all the people who are currently reading books that User "Boris Verhaegen" previously suggested to them

3 Object Databases (5 points)

You're designing a database for an exercise tracking application, which allows users to log their workouts. For example, user *Jean worked out on january 25, doing 50 push-ups, 200 crunches and running three kilometers.* Each repetition of an exercise earns the user some points, for example: 1 point per crunch, 2 points per push-up, 10 points for running a kilometer. Users earn medals when they cross certain thresholds, for example having logged 1000 push-ups, or reaching 5000 points.

Here's a UML representation of the schema :



And here is part of the associated types in Oracle:

```
CREATE TYPE TUser;
CREATE TYPE TWorkOut;
CREATE TYPE TWOExerciseSet;
CREATE TYPE TMExerciseSet;
CREATE TYPE TExercise;
CREATE TYPE TSetRefUsers AS TABLE OF REF TUser;
CREATE TYPE TSetRefWorkOut AS TABLE OF REF TWorkOut;
CREATE TYPE TSetRefWOExerciseSet AS TABLE OF REF TWOExerciseSet;
CREATE TYPE TSetRefMExerciseSet AS TABLE OF REF TMExerciseSet;

CREATE OR REPLACE TExercise AS OBJECT(
        type VARCHAR2(40),
        points INTEGER,
        WOExerciseSetRefs TWOSetRefExerciseSet,
        MExerciseSetRefs TMSetRefExerciseSet
);
CREATE OR REPLACE TYPE TWOExerciseSet AS OBJECT(
        repetitions INTEGER,
        exerciseRef REF TExercise,
        workOutRef REF TWorkOut,
);
CREATE OR REPLACE TYPE TWorkOut(
        date DATE,
        userRef REF TUser,
        WOExerciseSetRefs TSetRefExerciseSet
);
```

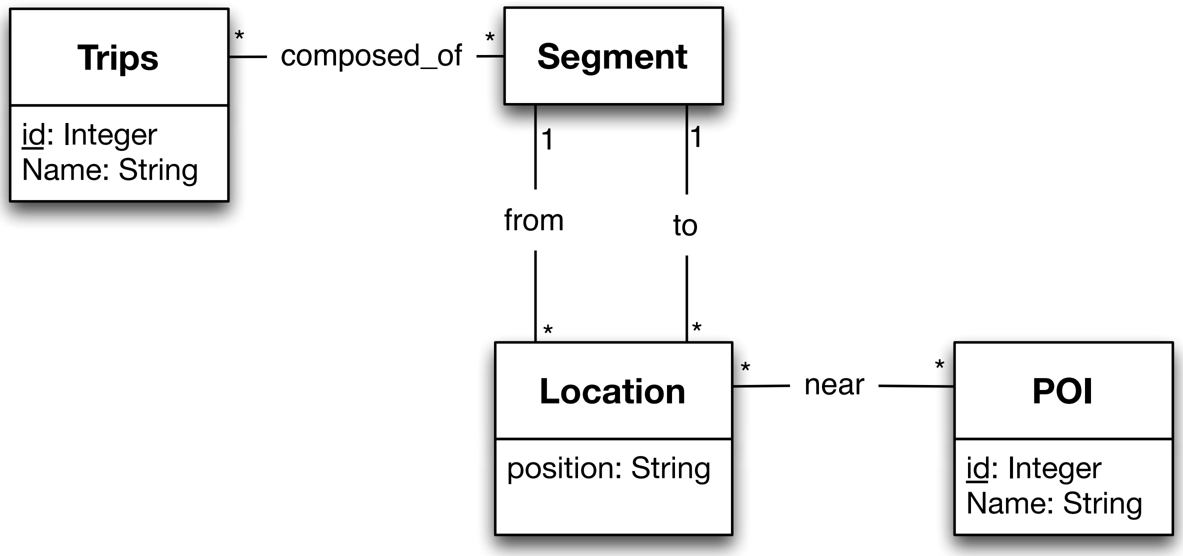
1. Write the type definitions for the User, Medals, and MExerciseSet objects, then write the following Oracle queries:

2. List all of User "Chuck Norris"'s workouts, listing the exercises he's done.

3. List all the users who have the medal ”Fab’ulous”
4. List all the users who have earned the medal ”Fab’ulous” and how many of the exercise ”crunch” they’ve done.
5. Find all the users who are stored as having earned the medal ”Fab’ulous” but haven’t actually logged all the exercises needed to earn it.

4 Spatial Databases (5 points)

You’ve just taken over maintaining a database for a travel agency that offers all-inclusive bus trips. Each trip is composed of a succession of segments : tourists are taken by bus from their starting location to the first stop, where they are left for a while to explore nearby points of interests (museums, cafes, sights, etc.) They then board the bus and travel on the next segment, eventually reaching the second stop. The process repeats until the trip loops back to the original location.



When you first look at the system however, (schema above) you’re shocked to discover that it doesn’t use spatial databases at all: it’s a standard SQL database where locations are stored as simple strings containing latitude and longitude coordinates, segments are simple pairs of from/to locations with no information about the actual path followed by busses along a segment, and the precise location of points of interest is never even recorded (only the fact that they are ”near” the bus stops from where they should be visited). For all these reasons, the system is no help at all for designing new trips or reorganizing segments.

You decide to refactor the database and turn it into a real PostGIS geographical database, where all Points of Interests have their location stored and the actual path followed by busses along segments are stored precisely.

1. In plain english, how would you modify the database (what would you add and/or remove) to make it a postGIS geographical database that stores the information described above ?
2. Write down the PostGIS commands needed to add the needed geographical columns to the existing schema

3. Select the name and length of the longest trip

4. Assuming you have an altitude table that contains a raster containing altitude data for the entire region, select the highest POI in the dataset.

5. For each segment, list POIs that the bus passes by without stopping, i.e. POIs that are less than 1km from the path of the bus but more than 5km from any of its stops.

You might need some of the following PostGIS functions:

1. AddGeometryColumn(table_name, column_name, 4326, varchar type, integer dimension) Where *type* can be any of POINT, LINE, MULTILINE, MULTIPOLYGON, etc.

2. *ST_Length2D_Spheroid(geom, SPHEROID["GRS_1980",6378137,298.257222101])* - compute the length of the (linear) feature geom in meters.

3. *Geography(geom)* - Converts a geometry to geography

4. *ST_DWithin(geography_1, geography_2, distance_meters)* - returns true if both geographies are within *distance_meters* from each other

5. *ST_SummaryStats(rast, 1)* - Returns min, max and average values for band 1 of raster rast.

6. *ST_Value(rast, geom)* - Returns the value of raster rast at the (point) geometry geom.

7. *ST_Intersects(geom, rast)* - Returns true if the geometry and raster intersects.