

DIT032 – DATA MANAGEMENT

ASSIGNMENT 2 – SOLUTIONS

University of Gothenburg

Deadline: **Tue 20.02.2018 23:55**

General Remarks

- There are in total 100 points to be reached for this assignment, and you need at least 70 points to pass the assignment. **Note that you need to pass all assignments individually.** It is not possible to "carry over" some points from this assignment to the next ones.
- Assignment solutions are to be produced in teams of two students. Discussions with other colleagues (e.g., through the forum in GUL) are encouraged, but don't share your assignment solutions, or significant parts, with your colleagues (neither through the forum nor through other means). If we find that multiple groups submitted the same assignments, we reserve the right to fail **all** involved groups.
- Remember to also hand in self- and peer assessment forms. You can use the forms available separately on GUL. Please contact us if you have any questions regarding this part. Assessment forms can either be scanned and attached to the submission (sign first!), handed in during the lecture or supervision, or brought in person to the office hour.
- We offer two tutor supervision hours (Wednesday and Friday, 13:15 – 15:00), where you can talk to the tutors, get feedback on your assignment work, and ask questions.
- **No** deadline extensions are given. Start early and, after finishing your assignment, upload your submission as a **single ZIP file** in GUL. The submission system closes automatically, and we do not accept late submissions. If you can't make it in time, you will need to submit during the first re-submission date in May.
- Shortly after the deadline, we grade the assignment and send you feedback. If you fail the assignment, you have the possibility to submit an improved version of the assignment for re-grading in May or October (see the website for details). We only re-grade failed submissions. If you have detailed questions it is better to ask them in the forum or during the supervision.

Assignment 2 Title Page

(use this page unfilled as the title page for your submission document)

Student	
----------------	--

Task		Pts.	Comments
ER Mapping to Relational Schema	Mapping ER Model to DDL (max 22 pts)		
Structured Query Language (SQL)	Formulating SQL Queries (max. 45 pts)		
Java Database Connectivity (JDBC)	Simple JDBC Query (max. 5 pts)		
	Seed the Database with Fake Data (max. 20 pts)		
	Query the Database (max. 8 pts)		

Total	
--------------	--

1 ER Mapping to Relational Schema

1.1 Mapping ER Model to DDL (22 points)

The goal of this task is to map the travel planner ER model from the previous assignment to a relational database schema.

Create a new database called `travel_planner` for this task. Save all your SQL code to a single file called `travel_planner.sql`, which you submit as part of your solution. Use SQL comments to clearly label where DDL3 and DDL4 are implemented. Make sure your solution is executable via:

```
psql -U postgres -d travel_planner -f travel_planner.sql
```

DDL1. Map the simplified EER diagram (provided in the appendix 3.4) from the travel planner task to a relational schema expressed in DDL. Use appropriate mapping strategies and explain your choice. Model all attributes, relationships, primary keys, and foreign keys.

DDL2. Insert at least 3 entries of test data into every table.

DDL3. Specify a constraint to ensure that the Rating of every place is in the range [0,5].

DDL4. Specify a constraint to ensure that every Email_address from a user contains the @ character.

Solution

The DDL code of the mapping and example entries are provided in `solutions/travel_planner.sql`.

2 Structured Query Language (SQL)

The following tasks are based on the *Mondial III* database [1], which you setup in task 3 of assignment 1. Refer to the appendix 3.4 for its relational schema. You may also find the referential dependencies overview in the appendix helpful.

2.1 Formulating SQL Queries (45 points)

Formulate SQL queries that answer the following questions. Ensure that your queries are correct for all Mondial database instances (i.e., do not use any constants other than the ones provided in the descriptions). Save all your queries into a SQL file called `mondial_queries.sql`, which you submit as part of your solution. Label each query with an SQL comment as for example:

```
-- SQL1
SELECT your FROM solution;
```

Solution

The solutions to the SQL queries 1-15 are provided in `solutions/mondial_queries.sql`

SQL1. The name of all islands that are of type coral or atoll. You may use the constants 'coral' and 'atoll'.

RA: $\pi_{name} (\sigma_{type='coral' \vee type='atoll'} (Island))$

Solution

```
SELECT name FROM island WHERE type='coral' OR type='atoll';
```

SQL2. The name, province, country, and population of all cities above the arctic circle. You may use the constant 66.33.

RA: $\pi_{name, province, country, population} (\sigma_{latitude > 66.33} (City))$

Solution

SELECT name, province, country, population FROM city WHERE latitude > 66.33;

SQL3. The country code of all countries belonging to the continent Europe (at least partially) that are not members of the European Union. You may use the constants 'Europe' and 'EU' (i.e., the abbreviation of the European Union).

RA: $\pi_{country} (\sigma_{continent='Europe'} (encompasses))$

—

$\pi_{country} (\sigma_{organization='EU'} (isMember))$

Solution

(SELECT country FROM encompasses WHERE continent='Europe') EXCEPT (SELECT country FROM ismember WHERE organization='EU');

SQL4. The name of all countries where more than half of the population speaks English. You may use the constant 'English' and 50.

RA: $\pi_{Country.name} (Country \bowtie_{code=country} (\sigma_{Language.name='English' \wedge percentage > 50} (Language)))$

Solution

SELECT c.name FROM country c JOIN language l ON c.code=l.country WHERE l.name='English' AND l.percentage>50;

SQL5. The name of all provinces that have waters (i.e., lake or sea) deeper than 1000 meters. You may use the constant 1000.

RA: You can inline *deepLake* and *deepSea* into a single query.

$deepLake \leftarrow \sigma_{depth > 1000} (Lake)$

$deepSea \leftarrow \sigma_{depth > 1000} (Sea)$

$\pi_{province} (deepLake \bowtie_{name=lake} geo_lake)$

∪

$\pi_{province} (deepSea \bowtie_{name=sea} geo_sea)$

Solution

(SELECT province FROM geo_lake JOIN lake ON name=lake WHERE depth>1000) UNION DISTINCT (SELECT province FROM geo_sea JOIN sea ON name=sea WHERE depth>1000);

SQL6. The difference in population between 2001 and 2011 for every country (i.e., the country code). You may use the constants 2001 and 2011.

RA: $year01 \leftarrow \rho_{country, pop01} (\pi_{country, population} (\sigma_{year=2001} (Countrypops)))$

$year11 \leftarrow \rho_{country, pop11} (\pi_{country, population} (\sigma_{year=2011} (Countrypops)))$

$\pi_{country, pop11-pop01} (year01 * year11)$

Solution

SELECT country, pop11 - pop01 AS pop_diff FROM (SELECT country, population AS pop01 FROM countrypops WHERE year=2001) AS c01 NATURAL JOIN (SELECT country, population AS pop11 FROM countrypops WHERE year=2011) AS c11;

SQL7. The abbreviation and average GDP for every organization with at least one member country. RA:

$organization \mathcal{F}_{AVG_gdp} (Economy \bowtie_{Economy.country=code} (isMember \bowtie_{isMember.country=code} Country))$

Solution

SELECT organization, AVG(gdp) FROM economy JOIN country ON economy.country=code JOIN ismember ON ismember.country=code GROUP BY organization;

SQL8. The name of all lakes that are adjacent to at least two countries. Use a single SQL query instead of variables.

RA: $geo_lake1 \leftarrow \rho_{lake1, country1, province1} (geo_lake)$

$geo_lake2 \leftarrow \rho_{lake2, country2, province2} (geo_lake)$

$\pi_{lake} (geo_lake1 \bowtie_{lake1=lake2 \wedge country1 \neq country2} geo_lake2)$

Solution

SELECT DISTINCT g1.lake FROM geo_lake g1 JOIN geo_lake g2 ON g1.lake=g2.lake AND g1.country <> g2.country;

SQL9. The name, the number of lakes, and the number of rivers of all countries. Do also include countries with no lakes or rivers (you may leave them as NULL values). Use a single SQL query without variables to solve this question.

RA: $lakeCount \leftarrow \rho_{country, num_lakes} (country \mathcal{F}_{COUNT_lake} (geo_lake))$

$riverCount \leftarrow \rho_{country, num_rivers} (country \mathcal{F}_{COUNT_river} (geo_river))$

$waterCount \leftarrow lakeCount \bowtie riverCount$

$\pi_{name, num_lakes, num_rivers} (Country \bowtie_{code=country} waterCount)$

Solution

SELECT name, lake_count, river_count
FROM country LEFT JOIN (
(SELECT country, COUNT(DISTINCT lake) AS lake_count
FROM geo_lake GROUP BY country) AS lc NATURAL FULL JOIN
(SELECT country, COUNT(DISTINCT river) AS river_count
FROM geo_river GROUP BY country) AS rc) AS both_counts
ON country.code=both_counts.country;

SQL10. The name and elevation of the highest mountain in Sweden. You may use the constant 'S' (i.e., country code of Sweden). You don't need to handle the special case of multiple highest mountains. Tip: Understanding U1 might be beneficial to solving this task.

RA: Use the SQL *WITH*¹ clause to match the RA expression closely. Provide a second solution without using *WITH* by reformulating the SQL query.

$sweMountain \leftarrow \pi_{name, elevation} (Mountain \bowtie_{name=mountain} (\sigma_{country='S'} (geo_mountain)))$

$sweMountain - sweMountain \bowtie_{elevation < elevation2} \rho_{name2, elevation2} (sweMountain)$

Solution

1) Closely matching with RA:

¹<https://www.postgresql.org/docs/current/static/queries-with.html>

WITH sweMountain AS (SELECT name, elevation FROM mountain JOIN geo_mountain ON name=mountain WHERE country='S')

(SELECT * FROM sweMountain) EXCEPT (SELECT DISTINCT s1.name, s1.elevation FROM sweMountain s1 JOIN sweMountain s2 ON s1.elevation < s2.elevation);

2) Simplified SQL:

SELECT name, elevation FROM mountain JOIN geo_mountain ON name = mountain WHERE country='S' ORDER BY elevation DESC LIMIT 1;

SQL11. The iatacode of the airport closest to the artic circle.

RA: Use the SQL *WITH*¹ clause to match the RA expression closely. Provide a second solution without using *WITH* by reformulating the SQL query.

$MyAirport1 \leftarrow \rho_{iatacode, x1} (\pi_{iatacode, |latitude-66.33|} (Airport))$

$MyAirport2 \leftarrow \rho_{iatacode2, x2} (MyAirport1)$

$\pi_{iatacode} (Airport) - \pi_{iatacode} (MyAirport1 \bowtie_{x1 > x2} MyAirport2)$

Solution

1) Closely matching with RA:

WITH myAirport AS (SELECT iatacode, abs(latitude-66.33) AS circle_dist FROM airport) (SELECT iatacode FROM myAirport) EXCEPT (SELECT a1.iatacode FROM myAirport a1 JOIN myAirport a2 ON a1.circle_dist > a2.circle_dist);

2) Simplified SQL:

SELECT iatacode FROM (SELECT iatacode, abs(latitude-66.33) AS circle_dist FROM airport ORDER BY circle_dist ASC LIMIT 1) AS closest_airport;

SQL12. The country code and the number of borders longer than 100 km. You may use the constant 100.

RA: $code \mathcal{F} COUNTlength (\sigma_{length > 100} (borders \bowtie_{country1=code \vee country2=code} country))$

Solution

SELECT code, count(length) FROM borders JOIN country ON country1=code OR country2=code WHERE length > 100 GROUP BY code;

SQL13. The name of all countries that only have borders longer than 100 km. You may use the constant 100.

Solution

WITH shortBorderCountries AS (
 (SELECT country1 AS country FROM borders WHERE length <= 100)
 UNION DISTINCT
 (SELECT country2 AS country FROM borders WHERE length <= 100)
)
 (SELECT name FROM country) EXCEPT (SELECT name FROM shortBorderCountries JOIN country ON code=country);

SQL14. The names of the richest country in terms of GDP per population (i.e., the GDP per capita) for every continent.

Solution

WITH rel_eco AS (
 SELECT enc.continent, c.name, (eco.gdp / c.population) AS rel_gdp

```

FROM economy eco
JOIN country c ON eco.country = c.code
JOIN encompasses enc ON c.code = enc.country
)
SELECT r1.continent, name
FROM rel_eco r1
INNER JOIN (
SELECT continent, MAX(rel_gdp) AS rel_gdp FROM rel_eco GROUP BY continent
) r2 ON r1.continent = r2.continent AND r1.rel_gdp = r2.rel_gdp;

```

SQL15. The name of all countries in the continent America for which the database records at least one spoken language that is not English and that is also spoken at least 80% in any country within the continent Africa. You may use the constant 'America', 'Africa', 'English', and 80.

Solution

```

SELECT DISTINCT country.name
FROM country
JOIN encompasses ON country.code = encompasses.country
JOIN language lang ON country.code = lang.country
WHERE encompasses.continent = 'America'
AND EXISTS(
SELECT l.name
FROM language l
JOIN encompasses e ON l.country = e.country
WHERE l.name = lang.name
AND e.continent = 'Africa'
AND lang.name <> 'English'
AND l.percentage >= 80);

```

3 Java Database Connectivity (JDBC)

The goal of this task is to compose SQL queries from within Java using JDBC.

3.1 Setup the Java Project Template (0 points)

As prerequisites, you need to have a Java JDK (minimum version 8²) installed and Maven³ (>= 3.5) configured.

1. Download the Java project jdbc from GUL for assignment 2.
2. Compile and resolve dependencies by running `mvn compile`.
3. Create a PostgreSQL database called `mydb`. Ensure that you can connect to *mydb* with the user `postgres` without password.
Feel free to change the credentials locally but make sure your submission works with the user `postgres` and without password!

²<http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

³<https://maven.apache.org/download.cgi>

3.2 Simple JDBC Query (5 points)

JDBC1. Complete the Java class `JDBCTestDriver.java`:

- (a) Establish a DB connection to the PostgreSQL database `mydb`.
- (b) Prepare the SQL statement `SELECT 7 AS test;`
- (c) Execute this SQL query and print the result to the console.
- (d) Close the statement and connection.
- (e) In case of an error, catch the exception, print an error message and exit the program with a failure status.

Solution

```
solutions/jdbc/src/main/java/se/chalmers/dm/JDBCTestDriver.java
```

3.3 Seed the Database with Fake Data (20 points)

Write *all* your SQL code into separate files under `resources/*.sql` and load it using the `sqlQuery(String filename)` helper method from `QueryHelper.java`. Make sure your SQL queries are protected against SQL injection vulnerabilities using the abstractions covered in the lecture.

JDBC2. Write the Java classes `Seeder.java`, `ConnectionHelper.java`, `User.java`, and `WebPage.java` such that the given code in `SeederTestDriver` works as intended:

- (a) `ConnectionHelper.createPostgresConnection()` returns a DB connection to the `mydb` database.
- (b) `seeder.createUserTable();` creates a table called "user" with the fields `Ssn`, `Name`, `Email`, and `IsActive` where the `Ssn` serves as primary key. Use appropriate data types based on the fake data (see next step).
- (c) `seeder.insertFakeUsers(10);` creates 10 fake users and stores them in the database. Use the `java-faker` library⁴ for generating valid SSNs, full names, email addresses, and randomize the active flag.
- (d) `seeder.createWebPageTable();` creates a table called `webpage` with the fields `Url`, `Author` as a foreign key pointing to a user's `Ssn`, `Content`, `Popularity` where the `Url` serves as primary key. Use appropriate data types based on the fake data (see next step).
- (e) `seeder.insertFakeUsersWithWebPage(100);` creates 100 fake users where every user is stored in the database, a single webpage authored by this user is created and also stored in the database. Use the `java-faker` library again to generate URLs, content using the Chuck Norris facts, and randomize the popularity score in the interval `[0, 100]`.

Solution

```
solutions/jdbc/src/*
```

⁴<http://dius.github.io/java-faker/apidocs/index.html>

3.4 Query the Database (8 points)

Write *all* your SQL code into separate files under `resources/*.sql` and load it using the `sqlQuery(String filename)` helper method from `QueryHelper.java`.

JDBC3. Write the Java class `Queries.java` such that the given code in `QueriesTestDriver` works as intended:

- (a) `List<User> activeUsers = queries.findActiveUsers(connection);` returns the list of active users from the database. Apply changes to `User.java` if necessary.
- (b) `List<String> quotes = queries.findSpecialQuotes(connection);` returns a list of strings with the content of all webpages that contain the word `all` using case insensitive search.

Solution


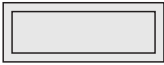
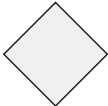
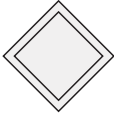

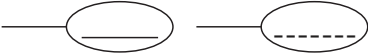

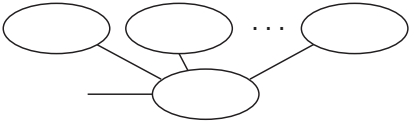

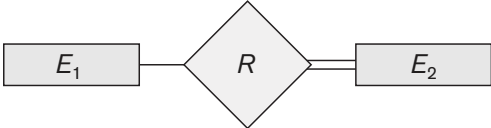
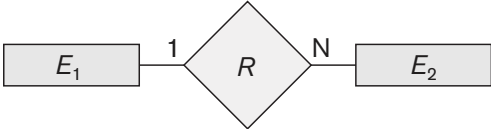
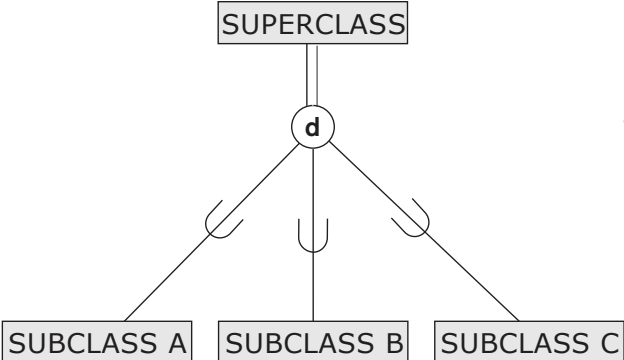
`solutions/jdbc/src/*`

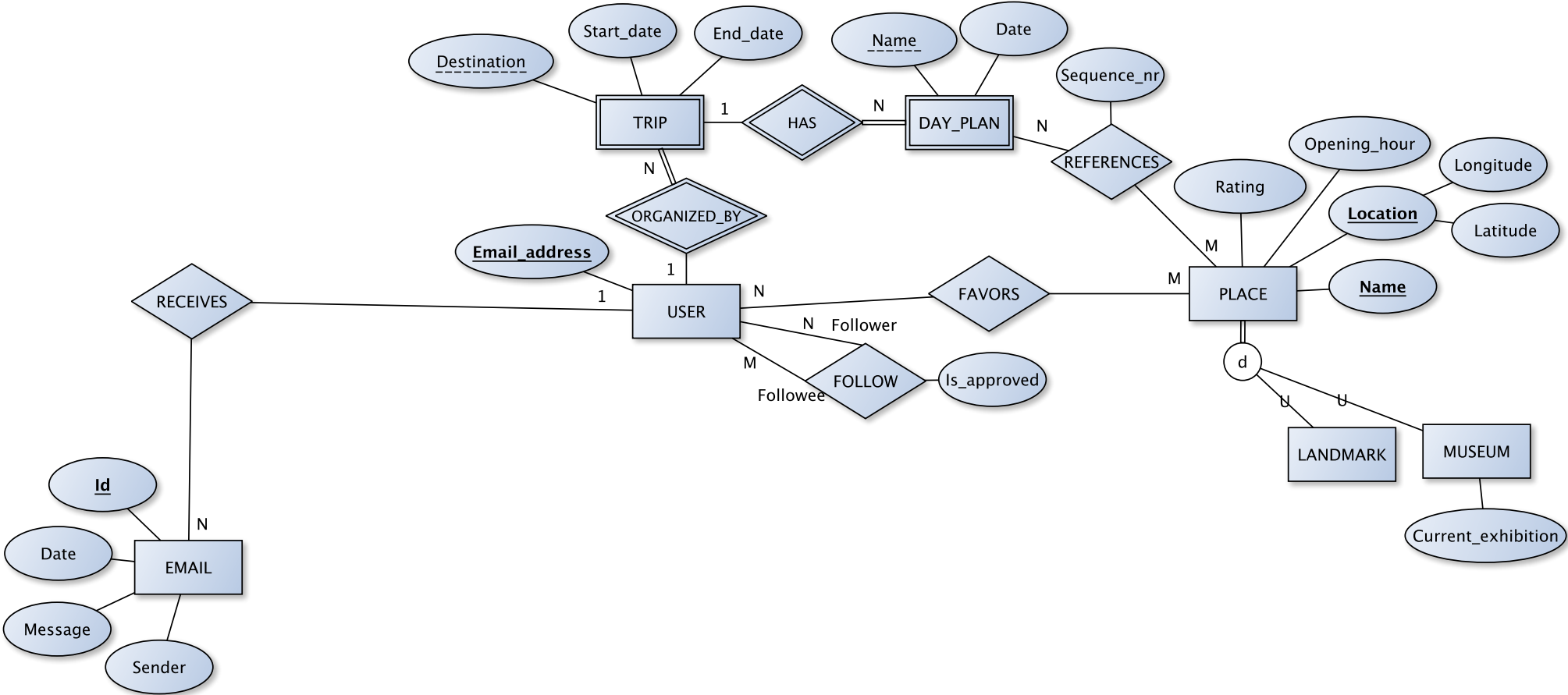
References

- [1] Wolfgang May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999. Available from <http://dbis.informatik.uni-goettingen.de/Mondial>.

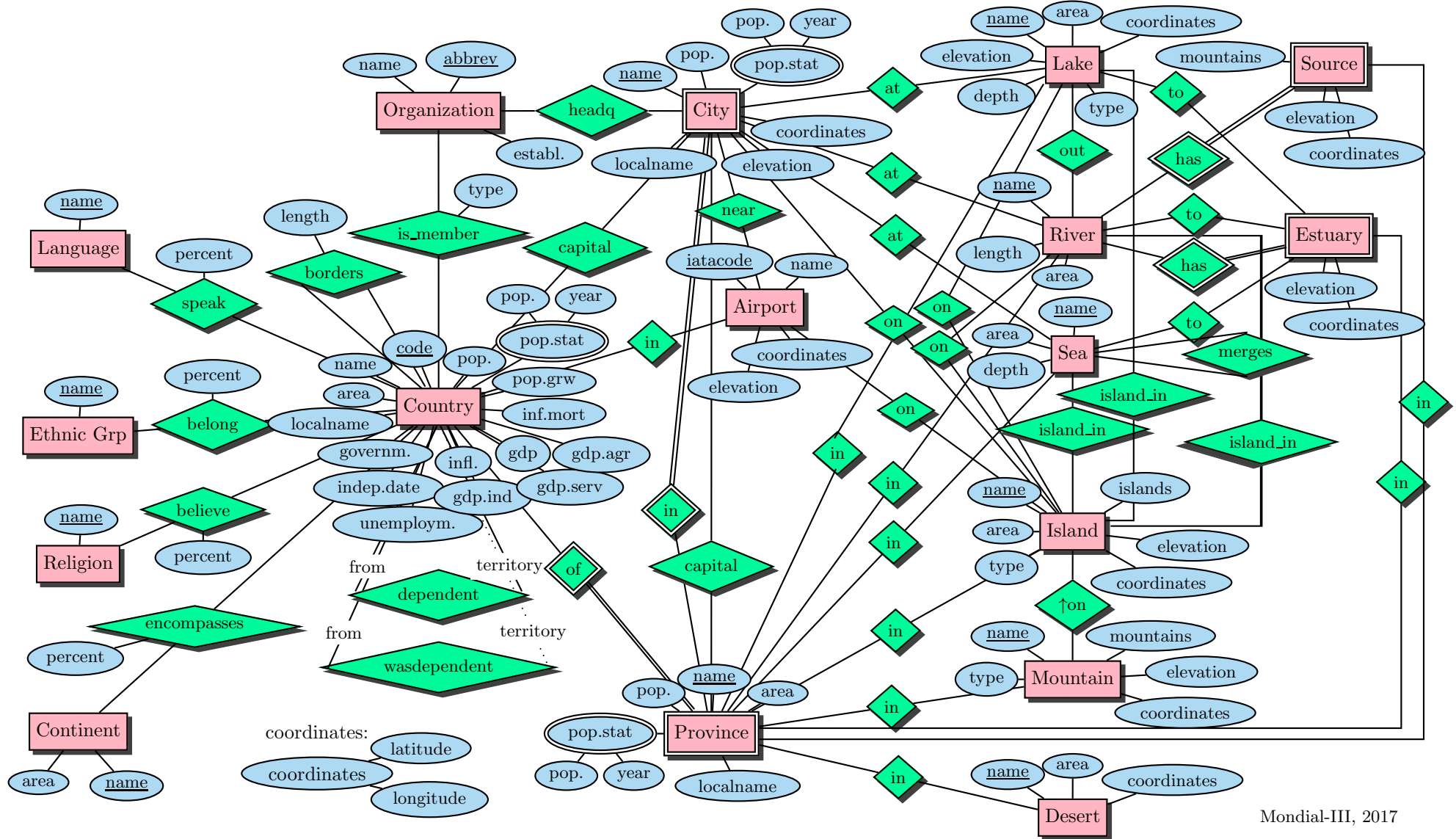
Appendix

- A) EER Notation Cheatsheet
- B) EER Diagram Travel Planner Solution Simplified
- C) ER-Diagram of the Mondial Database
- D) Referential Dependencies of the Mondial Database
- E) The Relational Schema of the Mondial database

Symbol	Meaning
	Entity
	Weak Entity
	Relationship
	Identifying Relationship
	Attribute
	Key Attribute / Dashed Underline for Partial Key
	Multivalued Attribute
	Composite Attribute
	Derived Attribute
	Total Participation of E_2 in R
	Cardinality Ratio 1 : N for $E_1 : E_2$ in R
	Total Disjoint Specialization



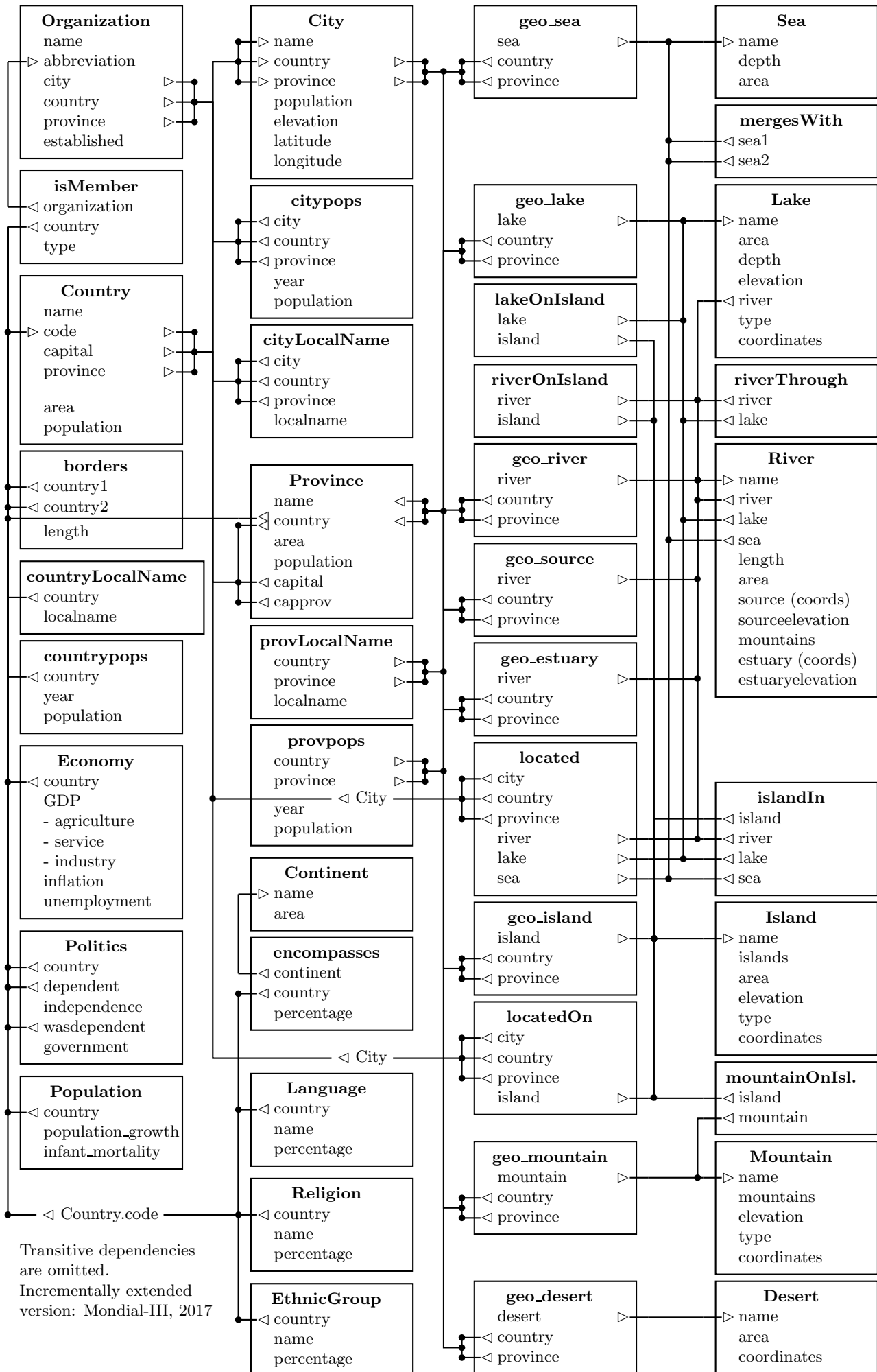
ER-Diagram of the Mondial Database



Mondial-III, 2017

Updates:
2018-01-24 add total participation for weak entity types

Referential Dependencies of the Mondial Database



The relational schema of the Mondial database

Country: the countries (and similar areas) of the world with some data.

name: the country name

code: the car code

capital: the name of the capital

province: the province where the capital belongs to

area: the total area

population: the population number

Economy: economical information about the countries.

country: the country code

GDP: gross domestic product (in million \$)

agriculture: percentage of agriculture of the GDP

service: percentage of services of the GDP

industry: percentage of industry of the GDP

inflation: inflation rate (per annum)

unemployment: unemployment rate

Politics: political information about the countries.

country: the country code

independence: date of independence (if independent)

wasdependent: the political body where the area was dependent of; usually a country (but not always).

dependent: the country code where the area belongs to

government: type of government

Population: information about the population of the countries.

country: the country code

population_growth: population growth rate (per annum)

infant_mortality: infant mortality (per thousand)

Countrypops: information about the population number of the countries in different years.

country: the country code

population: number of inhabitants

year: in which year

CountryLocalName: information about the local name of the country.

country: the country code

localname: the local name, usually in a local alphabet (UTF-8)

Language: information about the languages spoken in a country

country: the country code

name: name of the language

percentage: percentage of the language in this country

Religion: information about the religions in a country

country: the country code

name: name of the religion

percentage: percentage of the language in this country

EthnicGroup: information about the ethnic groups in a country

country: the country code

name: name of the religion

percentage: percentage of the language in this country

borders: informations about neighboring countries. Note that in this relation, for every pair of neighboring countries (A,B), only one tuple is given – thus, the relation is *not* symmetric.

country1: a country code

country2: a country code

length: length of the border between country1 and country2

Continent: Information about continents.

name: name of the continent

area: total area of the continent

encompasses: information to which continents a country belongs.

country: the country code

continent: the continent name

percentage: percentage, how much of the area of a country belongs to the continent

City: information about cities.

name: the name of the city

country: the code of the country where it belongs to

province: the name of the province where it belongs to

population: population of the city

elevation: the elevation (above sea level) of the city

latitude: geographic latitude

longitude: geographic longitude

Citypops: information about the population number of the cities in different years.

city: the name of the city

province: the name of the province

country: the code of the country where it belongs to

population: number of inhabitants

year: in which year

CityLocalName: information about the local name of the city.

city: the name of the city

province: the name of the province

country: the code of the country where it belongs to localname:
the local name, usually in a local alphabet (UTF-8)

Province: information about administrative divisions.

name: the name of the administrative division

country: the country code where it belongs to

area: the total area of the province

population: the population of the province

capital: the name of the capital

capprov: the name of the province where the capital belongs to

note that *capprov* is not necessarily equal to *name*. E.g., the municipality of *Bogota (Colombia)* is a province of its own, and *Bogota* is also the capital of the surrounding province *Cundinamarca*.

Provpops: information about the population number of the provinces in different years.

province: the name of the province

country: the code of the country where it belongs to

population: number of inhabitants

year: in which year

ProvinceLocalName: information about the local name of the province.

province: the name of the province

country: the code of the country where it belongs to localname:

the local name, usually in a local alphabet (UTF-8)

Organization: information about political and economical organizations.

name: the full name of the organization

abbreviation: its abbreviation

city: the city where the headquarter is located

country: the code of the country where the headquarter is located

province: the name of the province where the headquarter is located

established: date of establishment

isMember: memberships in political and economical organizations.

organization: the abbreviation of the organization

country: the code of the member country

type: the type of membership

Lake: information about lakes.

name: the name of the lake

area: the total area of the lake

depth: the depth of the lake

elevation: the elevation (above sea level) of the lake

river: the river that flows out of the lake (may be null)

type: the type of the lake, e.g., salt, caldera, ...

coordinates: its geographical coordinates as (latitude, longitude)

Sea: information about seas.

name: the name of the sea

depth: the maximal depth of the sea

area: the total area of the sea

River: information about rivers.

name: the name of the river

length: the length of the river

area: the size of its catchment area

river: the river where it finally flows to

lake: the lake where it finally flows to

sea: the sea where it finally flows to;

(note that at most one out of {river,lake,sea} can be non-null)

source: the coordinates of its source

sourceElevation: the elevation (above sea level) of its source

mountains: the mountains where its source is located

estuary: the coordinates of its estuary

estuaryElevation: the elevation (above sea level) of its estuary

RiverThrough: information about rivers flowing through lakes.

river: the name of the river

lake: the lake where it flows through

Mountain: information about mountains

name: the name of the mountain

mountains: the mountains where it belongs to

elevation: the maximal elevation of the summit of the mountain

type: the type of the mountain, e.g. volcanic, (active) volcano, ...

coordinates: its geographical coordinates as (latitude, longitude)

Island: information about islands

name: the name of the island

islands: the group of islands where it belongs to

area: the area of the island

elevation: the maximal elevation of the island

type: the type of the island, e.g. volcanic, coral, atoll, ...

coordinates: its geographical coordinates as (latitude, longitude)

Desert: information about deserts.

name: the name of the desert

area: the total area of the desert

coordinates: its geographical coordinates as (latitude, longitude)

mergesWith: information about neighboring seas. Note that in this relation, for every pair of neighboring seas (A,B), only one tuple is given – thus, the relation is *not* symmetric.

sea1: a sea

sea2: a sea

located: information about cities located at rivers, lakes, and seas.

city: the name of the city

country: the country code where the city belongs to

province: the province where the city belongs to

river: the river where it is located at

lake: the lake where it is located at

sea: the sea where it is located at

Note that for a given city, there can be several lakes/seas/rivers where it is located at.

locatedOn: information about cities located in islands.

city: the name of the city

country: the country code where the city belongs to

province: the province where the city belongs to

island: the island it is (maybe only partially) located on

Note that for a given city, there can be several islands where it is located on.

islandIn: information the waters where the islands are located in.

island: the name of the island

sea: the sea where the island is located in

lake: the lake where the island is located in

river: the river where the island is located in

Note that an island can have coasts to several seas.

MountainOnIsland: information which mountains are located on islands.

mountain: the name of the mountain

island: the name of the island

RiverOnIsland: information which rivers are located on islands.

river: the name of the river

island: the name of the island

LakeOnIsland: information which lakes are located on islands.

lake: the name of the lake

island: the name of the island

Airport: information about airports

iatacode: the IATA code of the airport

name: the name of the airport

country: the country code where the airport is located

city: in case the airport is associated with a city, the name of the city

province: the province where the city belongs to

island: if it is located on an island, the name of this island

latitude: geographic latitude

longitude: geographic longitude

elevation: the elevation (above sea level) of the city

gmtOffset: the GMT offset of the local time

geo_desert: geographical information about deserts.

desert: the name of the desert

country: the country code where it is located

province: the province of this country

geo_estuary: geographical information about the estuary of rivers.

river: the name of the river

Country: the country code where it is located

province: the province of this country

geo_island: geographical information about islands.

island: the name of the island

country: the country code where it is located

province: the province of this country

geo_lake: geographical information about lakes.

lake: the name of the lake

country: the country code where it is located

province: the province of this country

geo_mountain: geographical information about mountains.

mountain: the name of the mountain

country: the country code where it is located

province: the province of this country

geo_river: geographical information about rivers.

river: the name of the river

country: the country code where it is located

province: the province of this country

geo_sea: geographical information about seas.

sea: the name of the sea

country: the code of the country to which the sea is adjacent

province: the province of this country

geo_source: geographical information about sources of rivers.

river: the name of the river

country: the country code where it is located

province: the province of this country

Incrementally extended version: Mondial-III, 2017

2017-11-17: Underlined primary keys

2017-11-21: Added geo_* explicitly