# Synthesis

## TP0

### Preliminary

* Relational Database Management Systems (RDBMS)
  + Supports transactions with ACID properties
    - Atomicity: A transaction block is either fully executed or completely canceled
    - Consistency (or Correctness): The resulting database is valid w.r.t to the integrity constraints.
    - Isolation: The effect of two concurrent transactions is the same as if one was scheduled before the other.
    - Durability: Once confirmed, a transaction cannot be rolled back.
  + Inefficient for sparse relations
  + Not suitable for aggregations
  + Need query optimization
  + Queries can be time expensive
* Relational model
  + Tables or relations, each of which have several columns or attributes, each of which have a type
  + Data is stored as records or tuples
  + Sets
    - of labels
    - of values
    - of types (for each )
  + Relation schema is a n-tuple where each is a pair with and

### Questions

* In this zeroth exercise, we will query a single the following tables (with its fields):
  + Room (Name, Time, Movie Title)
  + Movie (MovieTitle, Director, Actor)
  + Producer (ProducerName, MovieTitle)
  + Seen (Spectator, MovieTitle)
  + Like (Spectator, MovieTitle)
* 1: Where and when can we see the movie “Mad Max…”?
* DROP(FILTER(Room, movieTitle="Mad Max"), MovieTitle)
* 2: What are the movies directed by Welles?
* DROP(DROP(FILTER(Movie, Director="Welles"), Director), Actor)
* 3: Who are the actors of “Ran”?
* DROP(DROP(FILTER(Movie, MovieTitle="Ran"), Director), MovieTitle)
* 4: Where can we see a movie in which Signoret plays?
* moviesWithS = DROP(DROP(FILTER(Movie, Actor="Signoret"), Actor), Director)  
  whereMovie = RENAME(DROP(Movie, Time), MovieTitle, MovieTitleBis)  
  DROP(FILTER(PRODUCT(moviesWithS, whereMovie), movieTitle=movieTitleBis), [MovieTitle, MovieTitleBis])
* 5: Among the actors who produced at least one movie?
* actors = DROP(DROP(Movie, MovieTitle), Director)  
  producers = DROP(Producers, MovieTitle)  
  DROP(FILTER(PRODUCT(actors, producers), ProducerName=Actor), ProducerName)
* 6: Among the actors who directed a movie that they played in?
* DROP(FILTER(Movie, Director=Actor), [MovieTitle, Director])
* 7: Who plays in one (or more) movie from Varda?
* DROP(FILTER(Movie, Director="Varda"), [MovieTitle, Director])
* 8: Who are the actors playing in all the films from Chloé Zhao?
* ZhaoMovies = DROP(FILTER(Movie, Director="Zhao"), [Director, Actor])  
  allPairsA\_ZM = PRODUCT(actors, ZhaoMovies)  
  missingPairs = DIFFERENCE(allPairsA\_ZM, Movie)  
  actorsInAllZhao = DIFFERENCE(actors, missingPairs)
* 9: Who produces all the movies from Kurosawa?
* KurosawaMovies = DROP(FILTER(Movie, Director="Kurosawa"), [Director, Actor])  
  AllPairsKM\_P = PRODUCT(producers, KurosawaMovies)  
  missingPairs = DIFFERENCE(Producer, AllPairsKM\_P)  
  producersNotInAllK = DIFFERENCE(Producer, missingPairs)  
  DIFFERENCE(Producer, producersNotInAllK)
* 10: Who are the spectators watching all the movies?
* movieNames = DROP(Movie, [Director, Actor])  
  spectators = DROP(Seen, MovieTitle)  
  allPairs\_ms = PRODUCT(movieNames, spectators)  
  missingPairs = DIFFERENCE(allPairs\_ms, Seen)  
  DIFFERENCE(spectators, missingPairs)
* 11: Among the spectators, who like all the movies they see?
* DIFFERENCE(Seen, DIFFERENCE(Seen, Like))
* 12: Where can we see Adèle Haenel after 16:00?
* moviesRoomAfter16 = DROP(FILTER(Room, time>=16:00), Time)  
  moviesWithHaenel = RENAME(DROP(FILTER(Movie, Actor=Haenel), [Director, Actor]), MovieTitle, MovieTitleBis)  
  DROP(FILTER(PRODUCT(moviesRoomAfter16, moviesWithHaenel), movieTitle=movieTitleBis), [movieTitle, movieTitleBis])
* 13: What are the movies with no room projecting them?
* DIFFERENCE(Movie, Rooms)
* 14: Among the producers, who produce movies shown nowhere?
* DIFFERENCE(Producer, Room)
* 15: Among the producers who saw all the movies they directed?
* AllPairs\_P\_M = PRODUCT(movieNames, producers)  
  missingPairs = DIFFERENCE(AllPairs\_P\_M, Producer)  
  DIFFERENCE(producers, missingPairs)
* 16: Among the spectators, who saw all the movies from Kurosawa
* moviesK = DROP(FILTER(Movie, Director="Kurosawa"), [Director, Actor])  
  allPairs\_ms = PRODUCT(moviesK, spectators)  
  missingPairs = DIFFERENCE(allPairs\_ms, Seen)  
  DIFFERENCE(spectators, missingPairs)
* 17: Among the spectators, who liked a movie they did not watch?
* DROP(DIFFERENCE(Like, Seen), MovieTitle)
* 18: Among the spectators, who liked 0 movies?
* DIFFERENCE(DROP(Seen, MovieTitle), DROP(Like, MovieTitle))

## TP1

### Preliminary

* SQL types
  + BOOLEAN
  + INT: Signed range is from -2147483648 to 2147483647
  + SERIAL for an auto-incrementing identifier
  + REAL for floating-point numbers (4-byte)
  + NUMERIC for high-precision numbers (1000 digits)
  + TEXT or VARCHAR: text
  + BLOB for binary strings: up to 65,535 bytes
  + TIMESTAMP for date and time
* The subquery must have an alias (here, M1), even if it is not used

### Questions

* In this first exercise, we will query a single world table. The table contains information about the world’s countries. The table has the following fields:
  + name varchar(50): the English name of the country (primary key)
  + continent varchar(60): the English name of the continent
  + area decimal(10,0): the area in square kilometers (note: it is sometimes missing)
  + population decimal(11,0): the population (note: it is sometimes missing)
  + capital varchar(60): the English name of the capital of the country

#### Basics: countries and continents

* 1: Modify the query below to select the name of all countries in Europe. Hint: use the WHERE clause:
* SELECT  
   name  
  FROM  
   world  
  WHERE  
   continent = 'Europe';
* 2: Select all DISTINCT continents in the database:
* SELECT  
   DISTINCT continent  
  FROM  
   world;

#### Filtering names

* 3: Select all country names that start with an ‘F’. Hint: use LIKE. Note: with PostgreSQL, the LIKE operator is case-sensitive.
* SELECT  
   name  
  FROM  
   world  
  WHERE  
   name LIKE 'F%';
* 4: Select all country names containing the letter ‘z’:
* SELECT  
   name  
  FROM  
   world  
  WHERE  
   name LIKE '%z%';

#### Population

* 5: Select all country names having at most one million inhabitants:
* SELECT  
   name  
  FROM  
   world  
  WHERE  
   population <= 1000000;
* 6: Write a query returning the name and population of all European countries
* SELECT  
   name,  
   population  
  FROM  
   world  
  WHERE  
   continent = 'Europe';
* 7: Write a query returning the country name and population of France, Germany, and Italy (in that order):
* SELECT  
   name,  
   population  
  FROM  
   world  
  WHERE  
   name IN ('France', 'Germany', 'Italy')  
  ORDER BY  
   name;
* 8: Write a query returning the population of France:
* SELECT  
   population  
  FROM  
   world  
  WHERE  
   name = 'France';

#### Boolean conditions

* 9: Write a query selecting the name, population, and area of all countries that have at least 100 million inhabitants or have an area of at least 3 million square kilometers (or both). Order the results by name.
* SELECT  
   name,  
   population,  
   area  
  FROM  
   world  
  WHERE  
   population >= 100000000  
   OR area >= 3000000  
  ORDER BY  
   name;
* 10: Write a query selecting the name, population, and area of all European countries which have population greater than 50 million or area greater than 500000 square kilometers (or both), ordered by name. Hint: be careful!
* SELECT  
   name,  
   population,  
   area  
  FROM  
   world  
  WHERE  
   continent = 'Europe'  
   AND (  
   population >= 50000000  
   OR area >= 500000  
   )  
  ORDER BY  
   name;
* 11: Write a query selecting the name, population, and area of all countries that are either large in population or large in size, but not both. Specifically, select countries that have at least 100 million inhabitants OR have an area of at least 3 million square kilometers, but not both. Order the results by name
* SELECT  
   name,  
   population,  
   area  
  FROM  
   world  
  WHERE  
   (  
   population >= 100000000  
   AND NOT area >= 3000000  
   )  
   OR (  
   area >= 3000000  
   AND NOT population >= 100000000  
   )  
  ORDER BY  
   name;

#### String operations

* 12: Write a query selecting the country names that are the same as that country’s capital, sorted by name.
* SELECT  
   name  
  FROM  
   world  
  WHERE  
   capital = name  
  ORDER BY  
   name;
* 13: Write a query selecting the country names and capital names where both names have the same length. E.g., “Greece” and “Athens” both have length 6. Exclude the cases from the previous question, i.e., those where the country name is the same as the capital, and sort the results by country name. Hint: search the Internet for a function to compute the length of a string
* SELECT  
   name,  
   capital  
  FROM  
   world  
  WHERE  
   CHAR\_LENGTH(capital) = CHAR\_LENGTH(name)  
   AND capital <> name  
  ORDER BY  
   name;
* 14: Write a query returning the country name with the least population, along with its population.
* SELECT  
   name,  
   population  
  FROM  
   world  
  ORDER BY  
   population  
  LIMIT  
   1;
* 15: Write a query returning the name and population of the 5 countries with the greatest population (ordered by descending population, i.e., from greatest to least population), where this population is known. Note: use IS NOT NULL.
* SELECT  
   name,  
   population  
  FROM  
   world  
  WHERE  
   population IS NOT NULL  
  ORDER BY  
   population DESC  
  LIMIT  
   5;
* 16: Write a query returning the name, population, and area of the 100th country in alphabetical order
* SELECT  
   name,  
   population,  
   area  
  FROM  
   world  
  ORDER BY  
   name  
  LIMIT  
   1 OFFSET 99;
* 17: Write a query returning the name and population of the 10 most populous countries among the 20 countries with the greatest area. Hint 1: remember that the population and the area may be NULL! Hint 2: use a subquery, and remember to give an alias to its result!
* SELECT  
   \*  
  FROM  
   (  
   SELECT  
   name,  
   population  
   FROM  
   world  
   WHERE  
   area IS NOT NULL  
   AND population IS NOT NULL  
   ORDER BY  
   area DESC  
   LIMIT  
   20  
   ) AS T  
  ORDER BY  
   population DESC  
  LIMIT  
   10;

#### Computation and aggregation

* 18: Write a query returning the name and population density of the Asian countries. The population density is the population divided by the area. Order the results by name.
* SELECT  
   name,  
   population / NULLIF(area, 0) AS density  
  FROM  
   world  
  WHERE  
   continent = 'Asia'  
  ORDER BY  
   name;
* 19: Write a query returning the name of Asian countries with a new column dense containing “yes” for countries with density at least 100 inhabitants per square kilometer, and “no” otherwise. Sort the results to have first the dense countries, then the non-dense countries, and then order the countries in each group by name. Hint: use UNION.
* SELECT  
   name,  
   'yes' AS dense  
  FROM  
   world  
  WHERE  
   continent = 'Asia'  
   AND population / NULLIF(area, 0) >= 100  
  UNION  
  SELECT  
   name,  
   'no' AS t  
  FROM  
   world  
  WHERE  
   continent = 'Asia'  
   AND population / NULLIF(area, 0) < 100  
  ORDER BY  
   dense DESC,  
   name;
* SELECT  
   name,  
   CASE  
   WHEN (density > 100) THEN 'yes'  
   ELSE 'no'  
   END AS dense  
  FROM  
   (  
   SELECT  
   name,  
   population / area AS density  
   FROM  
   world  
   WHERE  
   (  
   area IS NOT NULL  
   AND area != 0  
   AND continent = 'Asia'  
   )  
   ORDER BY  
   name  
   );
* 20: Write a query returning the number of countries in the database. Hint: use COUNT(\*).
* SELECT  
   COUNT(\*)  
  FROM  
   world;
* 21: Write a query returning the total world population. Hint: use SUM.
* SELECT  
   SUM(population)  
  FROM  
   world;
* 22: Write a query returning the continent names and, for each continent, the number of countries in that continent. Hint: use aggregation. In this question and the next ones, unless otherwise stated, order the results by continent.
* SELECT  
   continent,  
   COUNT(\*)  
  FROM  
   world  
  GROUP BY  
   continent  
  ORDER BY  
   continent;
* 23: Write a query returning the continent names and, for each continent, the total population and total area of the countries in that continent.
* SELECT  
   continent,  
   SUM(population),  
   SUM(area)  
  FROM  
   world  
  GROUP BY  
   continent  
  ORDER BY  
   continent;
* 24: Show the name and total population of continents having total population at least 100 million.
* SELECT  
   continent,  
   SUM(population) AS total  
  FROM  
   world  
  GROUP BY  
   continent  
  HAVING  
   SUM(population) >= 100000000  
  ORDER BY  
   continent;
* 25: Show the name of continents and the number of countries in that continent that have population at least 1 million.
* SELECT  
   continent,  
   COUNT(\*)  
  FROM  
   world  
  WHERE  
   population >= 1000000  
  GROUP BY  
   continent  
  ORDER BY  
   continent;
* 26: Compute the average population of the world’s countries.
* SELECT  
   AVG(population) AS average  
  FROM  
   world;
* 27: Compute, for every continent name, three columns containing the following three values, rounded to the nearest integer with the ROUND function. Do you understand why the three values are different?
  + its population density (i.e., the continent’s population divided by the continent’s area, taking all countries into account),
  + the average of the population densities of its countries (for the countries with a non-NULL and non-zero area),
  + the sum of the population densities of its countries (for the countries with a non-NULL and non-zero area), divided by the number of countries
* SELECT  
   continent,  
   ROUND(SUM(population) / SUM(area)) AS total,  
   ROUND(AVG(population / NULLIF(area, 0))) AS average,  
   ROUND(SUM(population / NULLIF(area, 0)) / COUNT(\*)) AS fake\_average  
  FROM  
   world  
  GROUP BY  
   continent;
* 28: Return a table with a column alpha containing the first letter of country names (ordered alphabetically) and a column total with the total population of countries whose name starts with that letter. Hint: use SUBSTR as in the example.
* SELECT  
   SUBSTR(name, 1, 1) AS alpha,  
   SUM(population)  
  FROM  
   world  
  GROUP BY  
   alpha  
  ORDER BY  
   alpha;
* 29: Compute, for every continent name, its country having the largest area, and its country having the greatest population. Order the result by continent. Hint: this is a complicated task, use subqueries and/or join the world table with itself. Note: It is normal that Kazakhstan appears as a country in Europe.
* SELECT  
   W1.continent,  
   W1.name AS largest,  
   W2.name AS most\_populous  
  FROM  
   world AS W1,  
   world AS W2  
  WHERE  
   W1.continent = W2.continent  
   AND W1.area IS NOT NULL  
   AND W2.population IS NOT NULL  
   AND NOT EXISTS (  
   SELECT  
   1  
   FROM  
   world AS W  
   WHERE  
   W.continent = W1.continent  
   AND W.area > W1.area  
   )  
   AND NOT EXISTS (  
   SELECT  
   1  
   FROM  
   world AS W  
   WHERE  
   W.continent = W2.continent  
   AND W.population > W2.population  
   )  
  ORDER BY  
   W1.continent;
* 30: Compute, for every continent name, a count of how many countries in the continent are strictly more populous that the continent’s largest country (by area). Order the results by continent, and do not omit results where the count is zero.
* SELECT  
   continent,  
   SUM(total)  
  FROM  
   (  
   SELECT  
   continent,  
   COUNT(\*) AS total  
   FROM  
   world AS W1  
   WHERE  
   population > (  
   SELECT  
   population  
   FROM  
   world AS W2  
   WHERE  
   W1.continent = W2.continent  
   AND W2.area IS NOT NULL  
   ORDER BY  
   area DESC  
   LIMIT  
   1  
   )  
   GROUP BY  
   continent  
   UNION  
   SELECT  
   DISTINCT continent,  
   0 AS TOTAL  
   FROM  
   world  
   ) AS T  
  GROUP BY  
   continent  
  ORDER BY  
   continent;

## TP2

### Questions

* In this second exercise, we will query three tables: movie, casting, and actor.
  + The movie table describes movies, and contains the following fields:
    - id int(11), an identifier
    - title varchar(50), the title of the movie
    - yr int(11), the year the movie was released
    - director int(11), the identifier of the director
* The actor table contains people (actors and directors) and contains the following fields:
  + id int(11), an identifier
  + name varchar(50), the name of the person
* The casting describes actors starring in movies. It contains the following fields:
  + movieid int(11), the identifier of the movie in the movie table
  + actorid int(11), the identifier of the actor in the actor table
  + ord int(11), an integer describing the position of the actor in the film’s starring list. The first actor (called the leading actor) has ord value 1, the second actor has ord value 2, and so on.

#### Basic joins

* 1: Compute the title of every Star Wars movie (starting with “Star Wars”) and the name of its director. Sort the result by title.
* SELECT  
   title,  
   name  
  FROM  
   movie,  
   actor  
  WHERE  
   title LIKE 'Star Wars%'  
   AND director = actor.id  
  ORDER BY  
   title;
* 2: Compute the list of the names of the actors starring in the movie “Jurassic Park” (1993), in the order in which they starred (i.e., by increasing ord value).
* SELECT  
   name  
  FROM  
   movie,  
   actor,  
   casting  
  WHERE  
   movie.id = casting.movieid  
   AND actor.id = casting.actorid  
   AND title = 'Jurassic Park'  
  ORDER BY  
   ord;
* 3: Compute the list of the titles of the movies where “George Clooney” appeared, ordered by title.
* SELECT  
   title  
  FROM  
   movie,  
   actor,  
   casting  
  WHERE  
   movie.id = casting.movieid  
   AND actor.id = casting.actorid  
   AND name = 'George Clooney'  
  ORDER BY  
   title;
* 4: Compute the list of the titles of all movies released in 1920 together with the name of their leading actor (the one with ord value of 1), ordered by title.
* SELECT  
   title,  
   name  
  FROM  
   movie,  
   casting,  
   actor  
  WHERE  
   yr = 1920  
   AND casting.movieid = movie.id  
   AND casting.actorid = actor.id  
   AND ord = 1  
  ORDER BY  
   title;
* 5: Compute the list of the titles of all movies released in 1920 together with the name of their leading actor and with the name of their director, ordered by title.
* SELECT  
   title,  
   SA.name,  
   D.name  
  FROM  
   movie,  
   casting,  
   actor as SA,  
   actor as D  
  WHERE  
   yr = 1920  
   AND casting.movieid = movie.id  
   AND casting.actorid = SA.id  
   AND ord = 1  
   AND D.id = director  
  ORDER BY  
   title;
* 6: Compute the five movies in the database having the highest number of participating actors, and this number of participating actors, sorted by decreasing number of actors. Warning: beware of titles like “The Hunchback of Notre Dame” that are the titles of multiple movies!
* SELECT  
   title,  
   COUNT(\*) AS cnt  
  FROM  
   movie,  
   casting  
  WHERE  
   casting.movieid = movie.id  
  GROUP BY  
   movie.id,  
   title  
  ORDER BY  
   cnt DESC  
  LIMIT  
   5;
* 7: Compute the years where the actor “Rock Hudson” participated to strictly more than one movie, along with the number of movies to which he participated on that year, sorted by decreasing number of movies, then by ascending year.
* SELECT  
   yr,  
   COUNT(\*) AS count  
  FROM  
   movie,  
   casting,  
   actor  
  WHERE  
   movie.id = casting.movieid  
   AND actor.id = casting.actorid  
   AND name = 'Rock Hudson'  
  GROUP BY  
   yr  
  HAVING  
   COUNT(\*) > 1  
  ORDER BY  
   COUNT(\*) DESC,  
   yr;
* 8: Compute the names of actors who were the leading actor in a movie where Harrison Ford appeared (and were not Harrison Ford himself). Order the results by actor name.
* SELECT  
   DISTINCT A2.name  
  FROM  
   actor AS A1,  
   actor AS A2,  
   casting AS C1,  
   casting as C2  
  WHERE  
   A1.name = 'Harrison Ford'  
   AND A1.id = C1.actorid  
   AND C1.movieid = C2.movieid  
   AND C2.actorid = A2.id  
   AND C2.ord = 1  
   AND A2.name <> 'Harrison Ford'  
  ORDER BY  
   A2.name;
* 9: Compute the titles of movies which were both directed by Woody Allen and had Woody Allen appear as an actor, sorted in alphabetical order.
* SELECT  
   title  
  FROM  
   movie,  
   actor,  
   casting  
  WHERE  
   casting.movieid = movie.id  
   AND casting.actorid = actor.id  
   AND actor.name = 'Woody Allen'  
   AND movie.director = actor.id  
  ORDER BY  
   title;
* 10: Compute the titles of movies which were directed by Woody Allen or had Woody Allen appear as an actor (or both), sorted in alphabetical order.
* SELECT  
   title  
  FROM  
   movie,  
   actor,  
   casting  
  WHERE  
   casting.movieid = movie.id  
   AND casting.actorid = actor.id  
   AND actor.name = 'Woody Allen'  
  UNION  
  SELECT  
   title  
  FROM  
   movie,  
   actor  
  WHERE  
   movie.director = actor.id  
   AND actor.name = 'Woody Allen'  
  ORDER BY  
   title;

#### Trick questions

* 11: In which movie title did Alain Delon and Catherine Deneuve appear together?
* SELECT  
   title  
  FROM  
   movie,  
   casting AS C1,  
   casting AS C2,  
   actor AS A1,  
   actor AS A2  
  WHERE  
   A1.id = C1.actorid  
   AND A2.id = C2.actorid  
   AND C1.movieid = movie.id  
   AND C2.movieid = movie.id  
   AND A1.name = 'Alain Delon'  
   AND A2.name = 'Catherine Deneuve';
* 12: Find the only actor who appeared in all Star Wars movies.
* SELECT  
   name  
  FROM  
   actor  
  WHERE  
   NOT EXISTS (  
   SELECT  
   id  
   FROM  
   movie  
   WHERE  
   TITLE LIKE 'Star Wars%'  
   AND NOT EXISTS (  
   SELECT  
   \*  
   FROM  
   casting  
   WHERE  
   movieid = movie.id  
   AND actorid = actor.id  
   )  
   );
* 13: Find the only actor which only appeared in movies where Harrison Ford appeared, and appeared in strictly more than one such movie.
* SELECT  
   name  
  FROM  
   actor,  
   casting,  
   movie  
  WHERE  
   casting.actorid = actor.id  
   AND casting.movieid = movie.id  
   AND name <> 'Harrison Ford'  
   AND NOT EXISTS (  
   SELECT  
   1  
   FROM  
   movie AS M1,  
   casting AS C1  
   WHERE  
   C1.movieid = M1.id  
   AND C1.actorid = actor.id  
   AND NOT EXISTS (  
   SELECT  
   1  
   FROM  
   casting AS C2,  
   actor AS A2  
   WHERE  
   A2.name = 'Harrison Ford'  
   AND C2.actorid = A2.id  
   AND C2.movieid = C1.movieid  
   )  
   )  
  GROUP BY  
   name  
  HAVING  
   COUNT(movie.id) > 1;
* 14: For performance reasons, we limit ourselves to the movies released no later than 1930, and to the actor names starting with A, B, or C. We say that two actors X and Y are challengers if X was the leading actor in a movie where Y appeared and Y was the leading actor in a movie where X appeared. Compute all pairs X, Y of challengers (with X < Y, in alphabetical order of X and then of Y).
* WITH oldmov AS (  
   SELECT  
   id,  
   yr  
   FROM  
   movie  
   WHERE  
   yr <= 1930  
  ),  
  aactor AS (  
   SELECT  
   id,  
   name  
   FROM  
   actor  
   WHERE  
   name LIKE 'A%'  
   OR name LIKE 'B%'  
   OR name LIKE 'C%'  
  )  
  SELECT  
   DISTINCT A1.name,  
   A2.name  
  FROM  
   aactor AS A1,  
   aactor AS A2,  
   casting AS C1a,  
   casting AS C1b,  
   casting AS C2a,  
   casting AS C2b,  
   oldmov AS Ma,  
   oldmov AS Mb  
  WHERE  
   A1.id = C1a.actorid  
   AND A1.id = C1b.actorid  
   AND A2.id = C2a.actorid  
   AND A2.id = C2b.actorid  
   AND C1a.movieid = C2a.movieid  
   AND C1b.movieid = C2b.movieid  
   AND C1a.ord = 1  
   AND C2b.ord = 1  
   AND C1a.movieid = Ma.id  
   AND C1b.movieid = Mb.id  
   AND A1.name < A2.name  
  ORDER BY  
   A1.name,  
   A2.name;

## TP3

### Preliminary

* A relation satisfies the First Normal Form (1NF) if the data of every cell is an atomic type
  + There are only Single Valued Attributes
  + Attribute Domain does not change
  + There is a unique name for every Attribute/Column
  + The order in which data is stored does not matter
* Functional dependency (FD) on a relation is an assertion of the form where and are attributes of
  + Constraint that always holds
  + Always holds if
    - (trivial FD)
    - at least are in the left-hand side if are a key
    - if is an FD for each
* A relation is in Boyce-Codd Normal Form (BCNF) if for every non-trivial FD that it satisfies, then is a superkey
  + Disallows
    - FDs between non-key attributes (attributes outside the key)
    - FDs from a strict subset of the key attributes
  + Non-BCNF
    - Many-to-many relationship
* Schema design
  + Being complete, i.e., can represent everything that is needed
  + Being clear to developers and as simple as possible
  + Being precise: clear how to map actual business needs to data
  + Not being too broad, i.e., correctly reflect constraints that are assumed
  + Avoiding redundancy: make sure every data item is in one place
  + Ensuring good performance (often linked to simplicity)
* Entity-relationship diagrams
  + Attribute
    - Single-valued attributes
    - Multi-valued attributes
    - Derived attributes (dashed circle) are derivable from stored attributes
    - Key of weak entities (dash-underlined)
  + Entity
    - Weak entity (double lined square)
  + Relationship
    - Identifying relationships (double lined diamond)
    - is-A
      * Specialization: top-down
      * Generalization: bottom-up
    - Can be entities for other relationships
  + Roles are written above the lines connecting the relationship and entity
  + Cardinality constraints
    - One-to-one: functional and injective
    - One-to-many: injective but not functional
    - Many-to-one: functional but not injective
    - Many-to-many arbitrary and not functional
  + Below the role the minimal and maximal number of relationships in which an entity can participate ( means no limit)

### Questions

#### PostgreSQL

* List databases \l, \l+
  + \x on for narrower settings
* Connect to a database \c < databaseName >
* Display tables \dt
* Display schema \d < tableName >, \d+ < tableName >
* Display user roles \du
* Create a database $ createdb -U postgres < databaseName >
* Create a database $ dropdb -U postgres < databaseName >
* To alter a table ALTER TABLE < table >…
  + RENAME to < newTableName >;
  + ADD < columnName > < dataType >;
  + DROP < columnName >;
  + RENAME < oldColumnName > to < newColumnName >;
  + ALTER < columnName > < dataType >;

#### Schema and views

* Schema
* CREATE TABLE IF NOT EXISTS player (  
   id SERIAL,  
   name TEXT NOT NULL,  
   money INT CHECK (money >= 0),  
   PRIMARY KEY (id)  
  );  
    
  CREATE TABLE IF NOT EXISTS type (  
   id SERIAL,  
   name TEXT NOT NULL,  
   PRIMARY KEY (id)  
  );  
    
  CREATE TABLE IF NOT EXISTS possession (  
   id SERIAL,  
   type INT NOT NULL,  
   owner INT NOT NULL,  
   price INT,  
   PRIMARY KEY (id),  
   FOREIGN KEY (owner) REFERENCES player(id),  
   FOREIGN KEY (type) REFERENCES type(id)  
  );
* Create a new type of object or add new players,
* INSERT INTO  
   player (id, name, money)  
  VALUES  
   (DEFAULT, < name >, < money >);
* Change the name of a type of objects or the name of a player,
* UPDATE  
   player  
  SET  
   name = < name >  
  WHERE  
   id = < id >;
* Attribute an object of a given type to a player,
* INSERT INTO  
   possession (type, owner, price)  
  VALUES  
   (< typeId >, < ownerId >, NULL);
* Increase or decrease the amount of money a player has,
* UPDATE  
   player  
  SET  
   money = money + < diff >  
  WHERE  
   id = < id >;
* Retrieve the list of all the items that a player has,
* SELECT  
   type,  
   owner  
  FROM  
   possession  
  WHERE  
   owner = < playerId >;
* Compute the current balance of a player,
* SELECT  
   money  
  FROM  
   player  
  WHERE  
   id = < id >;
* Allow a player to mark one of their item as buyable with a given price,
* UPDATE  
   possession  
  SET  
   price = < price >  
  WHERE  
   id = < id >  
   AND owner = < playerId >;
* Allow a player to buy the cheapest item of a given type from the marketplace.
* START TRANSACTION;  
    
  SELECT  
   id,  
   price,  
   owner as curOwner  
  FROM  
   possession  
  WHERE  
   price IS NOT NULL  
   AND type = < desired\_type >  
  ORDER BY  
   price ASC  
  LIMIT  
   1;  
    
  UPDATE  
   player  
  SET  
   money = money + < price >  
  WHERE  
   id = < curOwner >;  
    
  UPDATE  
   player  
  SET  
   money = money - < price >  
  WHERE  
   id = < buyerId >;  
    
  UPDATE  
   possession  
  SET  
   owner = < buyerId >,  
   price = NULL  
  WHERE  
   id = < objectId >;  
    
  COMMIT;

#### Normalization & advanced exercices

* Create a department whose employees are located in different buildings using multivalued attributes.
* CREATE TABLE IF NOT EXISTS department (  
   dnumber INT NOT NULL,  
   dname TEXT,  
   d\_head INT,  
   d\_building TEXT,  
   PRIMARY KEY (dnumber)  
  );
* Retrieve information about a department based on the location,
* SELECT  
   \*  
  FROM  
   department  
  WHERE  
   d\_building LIKE '%BuildingB%';  
    
  SELECT  
   \*  
  FROM  
   department  
  WHERE  
   d\_building = 'BuildingB';
* Normalize the department information to comply with 1NF.
* CREATE TABLE IF NOT EXISTS department\_normalized (  
   id SERIAL,  
   dnumber INT,  
   dname TEXT,  
   d\_head INT,  
   d\_building TEXT,  
   PRIMARY KEY (id)  
   );
* Retrieve information about a department based on the location using exact match.
* SELECT  
   \*  
  FROM  
   department\_normalized  
  WHERE  
   d\_building = 'BuildingB';
* Create courses taught by the professors and attended by the students.
* CREATE TABLE IF NOT EXISTS teach (  
   student TEXT,  
   course TEXT,  
   professor TEXT  
   );
* Define possible decompositions of the courses.
* CREATE TABLE teach\_1\_1 AS  
  SELECT  
   student,  
   professor  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_1\_1  
  ADD  
   PRIMARY KEY (student, professor);  
    
  CREATE TABLE teach\_1\_2 AS  
  SELECT  
   student,  
   course  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_1\_2  
  ADD  
   PRIMARY KEY (student, course);  
    
  CREATE TABLE teach\_2\_1 AS  
  SELECT  
   course,  
   professor  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_2\_1  
  ADD  
   PRIMARY KEY (professor);  
    
  CREATE TABLE teach\_2\_2 AS  
  SELECT  
   course,  
   student  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_2\_2  
  ADD  
   PRIMARY KEY (course, student);  
    
  CREATE TABLE teach\_3\_1 AS  
  SELECT  
   course,  
   professor  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_3\_1  
  ADD  
   PRIMARY KEY (professor);  
    
  CREATE TABLE teach\_3\_2 AS  
  SELECT  
   student,  
   professor  
  FROM  
   teach;  
    
  ALTER TABLE  
   teach\_3\_2  
  ADD  
   PRIMARY KEY (student, professor);
* Reconstruct the courses taught by the professors and attended by the students.
* SELECT  
   course,  
   professor,  
   t1.student  
  FROM  
   teach\_1\_1 AS t1,  
   teach\_1\_2 AS t2  
  WHERE  
   t1.student = t2.student;  
    
  SELECT  
   t1.course,  
   professor,  
   student  
  FROM  
   teach\_2\_1 AS t1,  
   teach\_2\_2 AS t2  
  WHERE  
   t1.course = t2.course;  
    
  SELECT  
   course,  
   t1.professor,  
   student  
  FROM  
   teach\_3\_1 AS t1,  
   teach\_3\_2 AS t2  
  WHERE  
   t1.professor = t2.professor;
* Create information about specific employees by taking into account the properties of the employees in general.
* CREATE TABLE IF NOT EXISTS employee (  
   id SERIAL,  
   name TEXT,  
   salary INT,  
   PRIMARY KEY (id)  
  );  
    
  CREATE TABLE IF NOT EXISTS professor (  
   pid INT NOT NULL,  
   field TEXT,  
   PRIMARY KEY (pid)  
  ) INHERITS (employee);  
    
  CREATE TABLE IF NOT EXISTS secretary (  
   sid INT NOT NULL,  
   building TEXT,  
   PRIMARY KEY (sid)  
  ) INHERITS (employee);
* Retrieve information about all the types of employees.
* SELECT  
   \*  
  FROM  
   employee;  
    
  SELECT  
   \*  
  FROM  
   ONLY employee;  
    
  SELECT  
   \*  
  FROM  
   professor;  
    
  SELECT  
   \*  
  FROM  
   secretary;
* Create courses having dependent types of information which would not exist otherwise.
* CREATE TABLE IF NOT EXISTS course (  
   id SERIAL,  
   name TEXT,  
   PRIMARY KEY (id)  
  );  
    
  CREATE TABLE IF NOT EXISTS session (  
   course INT,  
   num INT,  
   name TEXT,  
   PRIMARY KEY (course, num),  
   FOREIGN KEY (course) REFERENCES course(id)  
  );

## TP4

### Preliminary

### Questions

* SeqScan: exploring the full table
* SELECT  
   \*  
  FROM  
   unicode;
* Index Scan: index to retrieve the position and fetch data
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   codepoint = '0000';
* Index Only Scan: index to retrieve position
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   codepoint < '0000';
* Bitmap Index Scan and Bitmap Heap Scan: bitmap using an index
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   charname = '<control>';
* BitmapOr: two bitmaps created and then another bitmap is built with the OR condition
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   numeric IS NOT NULL  
   or codepoint = '0000';
* BitmapAnd: two bitmaps created and then another bitmap is built with the AND condition
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   numeric IS NOT NULL  
   AND charname < 'b';
* Filter: filter when no indexes are present
* SELECT  
   \*  
  FROM  
   unicode  
  WHERE  
   comment IS NOT NULL;
* Nested Loop: join that cannot be efficiently processed
* SELECT  
   \*  
  FROM  
   unicode u1,  
   unicode u2;
* Merge Join and Hash Join:
* SELECT  
   \*  
  FROM  
   unicode u1,  
   unicode u2  
  WHERE  
   u1.comment = u2.comment;  
    
  SELECT  
   \*  
  FROM  
   unicode u1,  
   unicode u2  
  WHERE  
   u1.codepoint = u2.lowercase;

## TP5

### Preliminary

### Questions

* 1: Finds all the Shakespeare performances at Newcastle’s Theatre Royal.
* MATCH (theater:Venue {name:'Theatre Royal'}),  
   (newcastle:City {name:'Newcastle'}),  
   (bard:Author {lastname:'Shakespeare'}),  
   (newcastle)<-[:STREET|CITY\*1..2]-(theater)  
   <-[:VENUE]-()-[:PERFORMANCE\_OF]->()  
   -[:PRODUCTION\_OF]->(play)<-[:WROTE\_PLAY]-(bard)  
  RETURN DISTINCT play.title AS play
* 2: Finds all the Shakespeare performances at Newcastle’s Theatre Royal after 1608.
* MATCH (theater:Venue {name:'Theatre Royal'}),  
   (newcastle:City {name:'Newcastle'}),  
   (bard:Author {lastname:'Shakespeare'}),  
   (newcastle)<-[:STREET|CITY\*1..2]-(theater)  
   <-[:VENUE]-()-[:PERFORMANCE\_OF]->()  
   -[:PRODUCTION\_OF]->(play)<-[w:WROTE\_PLAY]-(bard)  
  WHERE w.year > 1608  
  RETURN DISTINCT play.title AS play
* 3: How many Shakespeare performances were at Newcastle’s Theatre Royal?
* MATCH (theater:Venue {name:'Theatre Royal'}),  
   (newcastle:City {name:'Newcastle'}),  
   (bard:Author {lastname:'Shakespeare'}),  
   (newcastle)<-[:STREET|CITY\*1..2]-(theater)  
   <-[:VENUE]-()-[:PERFORMANCE\_OF]->()  
   -[:PRODUCTION\_OF]->(play)<-[w:WROTE\_PLAY]-(bard)  
  RETURN count(play)
* 4: Rank plays by number of performances.
* MATCH (theater:Venue {name:'Theatre Royal'}),  
   (newcastle:City {name:'Newcastle'}),  
   (bard:Author {lastname:'Shakespeare'}),  
   (newcastle)<-[:STREET|CITY\*1..2]-(theater)  
   <-[:VENUE]-()-[p:PERFORMANCE\_OF]->()  
   -[:PRODUCTION\_OF]->(play)<-[:WROTE\_PLAY]-(bard)  
  RETURN play.title AS play, count(p) AS performance\_count  
  ORDER BY performance\_count DESC
* 5: Find the plays written by Shakespeare, and order them based on the year in which they were written. (HINT: Use WITH and collect())
* MATCH (bard:Author {lastname:'Shakespeare'})-[w:WROTE\_PLAY]->(play)  
  WITH play  
  ORDER BY w.year DESC  
  RETURN collect(play.title) AS plays