ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

FS 2016 Prof. R. Wattenhofer

Computer Engineering II

Solution to Exercise Sheet 7

1 Quiz Questions

a) Both {`ID,TR-ID`} and {`ID,title,TR-ID`} uniquely identify any row within the table. There are generally a number of possible candidate keys. By definition, only one candidate key is choosen to be the table's primary key. Generally, the choice falls to the candidate key that has the smallest set of columns, which in this case is {`ID,TR-ID`}.

b) Query 6. results in:

```
ERROR 1111 (HY000):  Invalid use of group function.
```

SQL's WHERE clause does not work with aggregate functions like SUM, AVG, MAX, COUNT and so on. Instead, the HAVING keyword was introduced to SQL in order to quantitatively compare aggregateed values. A correct query would look like this:

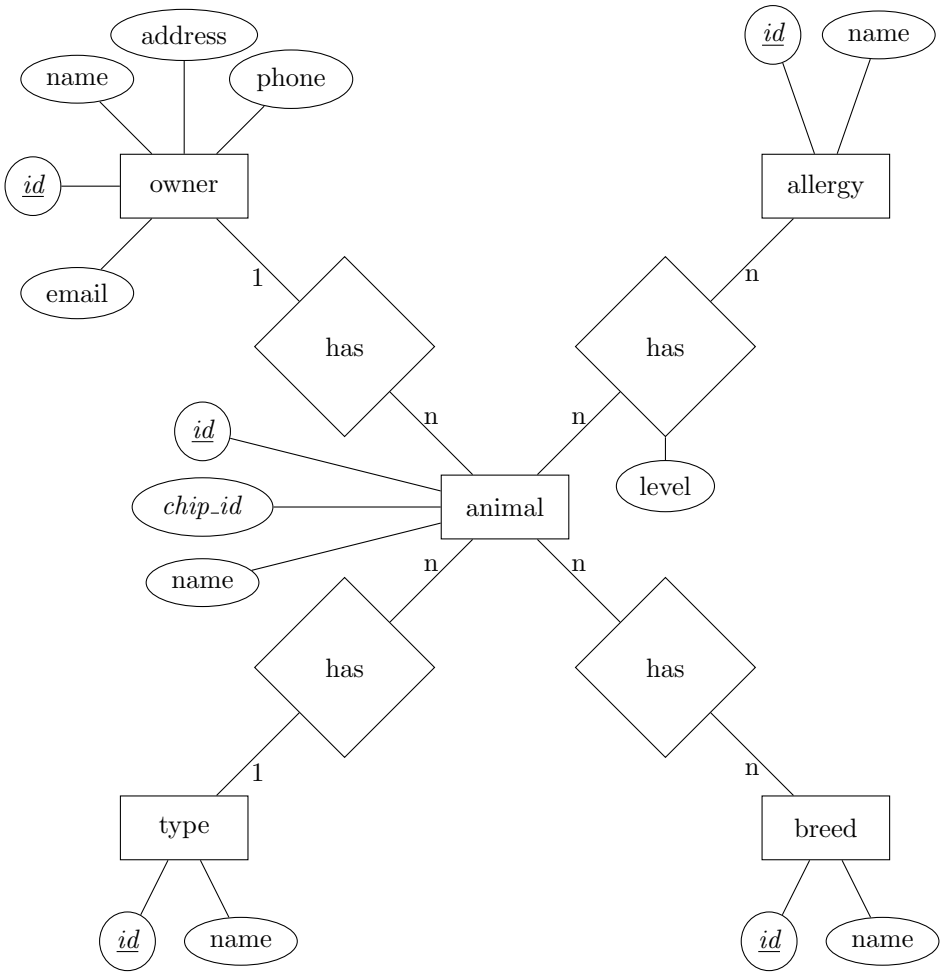
```
SELECT year,COUNT(*) FROM Articles GROUP BY year HAVING COUNT(*) > 10;
```

c) 3 rows

d) 3. is correct

2 Database Design

a) The following Entity Relationship Diagram describes the animals database. Owners and animals are in a 1-to-n relation. Each owner may own multiple animals, but every animal can have exactly one registered owner in the database. Animals and animal types are in a n-to-1 relation. Any animal cannot be both a cat and a dog, but the animal type table may very well contain multiple cats or dogs. Animals and breeds are in a n-to-n relation. Any animal can be a mixed breed and there may be multiple animals of the same breed in the database. Animals and allergies are in a n-to-n relation. Any animal may have multiple allergies and any allergy may afflict more than one animal in the database. For every animal allergy, we reserve a level field that denotes how strongly allergic the animal is to the allergy in question. Notice that we underline primary key attributes and we use an italic font to label unique attributes. The id field is chosen as the primary key for the animal table because even though the chip_id is a unique value, we would like to allow for the possibility that the unique chip_id is changed, for example, when the animal chip breaks or if it's updated due to a change in chip standard.



b)

```
DROP TABLE IF EXISTS Owner;
CREATE TABLE Owner (
id INT PRIMARY KEY,
name CHARACTER VARYING(255),
email CHARACTER VARYING(255),
phone CHARACTER VARYING(15),
address CHARACTER VARYING(255)
);

DROP TABLE IF EXISTS Type;
CREATE TABLE Type (
id INT PRIMARY KEY,
animal_type CHARACTER VARYING(50),
);


DROP TABLE IF EXISTS Animal;
CREATE TABLE Animal (
id INT PRIMARY KEY,
chip_id CHARACTER VARYING(50) UNIQUE,
name CHARACTER VARYING(50),
```

```
CONSTRAINT fk_animal_owner FOREIGN KEY (owner_id) REFERENCES Owner(id),
CONSTRAINT fk_animal_type FOREIGN KEY (type_id) REFERENCES Type(id)
);

DROP TABLE IF EXISTS Allergy;
CREATE TABLE Allergy (
id INT PRIMARY KEY,
allergy CHARACTER VARYING(255)
);

DROP TABLE IF EXISTS AllergicAnimals;
CREATE TABLE AllergicAnimals (
id INT PRIMARY KEY,
allergy_degree INT,
CONSTRAINT fk_allergic_animal_id FOREIGN KEY (animal_id) REFERENCES Animal(id),
CONSTRAINT fk_allergy_id FOREIGN KEY (allergy_id) REFERENCES Allergy(id)
);

DROP TABLE IF EXISTS Breed;
CREATE TABLE Breed (
id INT PRIMARY KEY,
animal_breed CHARACTER VARYING(50)
);

DROP TABLE IF EXISTS AnimalBreed;
CREATE TABLE AnimalBreed (
id INT PRIMARY KEY,
CONSTRAINT fk_animal_breed_animal_id FOREIGN KEY (animal_id) REFERENCES Animal(id),
CONSTRAINT fk_animal_breed_breed_id FOREIGN KEY (breed_id) REFERENCES Breed(id)
);
```

3 Database Queries

1. `SELECT id,title FROM movie LIMIT 5;`
2. `SELECT * FROM movie ORDER BY title DESC LIMIT 2;`
3. `SELECT COUNT(*) FROM movie WHERE year > 2000;`
4. `SELECT title,tomatometer FROM movie WHERE title = 'The Matrix' LIMIT 5;`
5.

```
SELECT COUNT(*) FROM movie
  WHERE tomatometer > ALL (
    SELECT tomatometer FROM movie
      WHERE title = 'The Matrix'
    );
```

6. `SELECT MAX(tomatometer),MIN(tomatometer),AVG(tomatometer) FROM movie LIMIT 1;`
7.

```
SELECT title FROM movie
  WHERE title LIKE 'X%'
  ORDER BY title DESC;
```

8.

```
SELECT COUNT(*) FROM movie
  WHERE title LIKE '%fight%';
```

or case insensitive:

```
SELECT COUNT(*) FROM movie
  WHERE title COLLATE UTF8_GENERAL_CI LIKE '%fight%';
```

4 More Database Queries

1.

```
SELECT person.name,cast_info.role_id,person.gender FROM cast_info
    JOIN person ON person.id = cast_info.person_id
    JOIN movie ON movie.id = cast_info.movie_id
  WHERE person.gender = 'f'
    AND cast_info.role_id = 2
    AND movie.title = 'The Matrix';
```

2.

```
SELECT COUNT(DISTINCT person.name) FROM cast_info
    JOIN role_type ON role_type.id = cast_info.role_id
    JOIN person ON person.id = cast_info.person_id
  WHERE role_type.role = 'director'
  AND person.gender = 'f';
```

3.

```
SELECT DISTINCT person.name FROM cast_info
    JOIN person ON person.id = cast_info.person_id
    JOIN movie ON movie.id = cast_info.movie_id
  WHERE (cast_info.role_id = 2 or cast_info.role_id = 1)
  AND EXISTS (
    SELECT DISTINCT ci.person_id FROM cast_info AS ci
      WHERE ci.role_id = 8
      AND cast_info.person_id = ci.person_id
      GROUP BY ci.person_id
      HAVING COUNT(ci.person_id) > 20
  );
```

4.

```
SELECT movie.title,COUNT(*) AS cnt FROM movie_keyword
    JOIN movie ON movie_keyword.movie_id = movie.id
    JOIN keyword ON keyword.id = movie_keyword.keyword_id
  GROUP BY movie_keyword.movie_id
  ORDER BY cnt DESC;
```

5.

```
SELECT AVG(cnt),MAX(cnt),MIN(cnt) FROM (
  SELECT movie.title,COUNT(*) AS cnt FROM movie_keyword
      JOIN movie ON movie_keyword.movie_id = movie.id
      JOIN keyword ON keyword.id = movie_keyword.keyword_id
    GROUP BY movie_keyword.movie_id
  ) AS countaverages;
```

6.

```
SELECT
    person.name,
    AVG(movie.tomatometer) AS average,
    count(ci.person_id) AS cnt,
    MAX(movie.year) AS maxyear
  FROM cast_info AS ci
    JOIN movie ON movie.id = ci.movie_id
    JOIN person ON person.id = ci.person_id
  WHERE ci.role_id = 1
  GROUP BY ci.person_id
  HAVING average > 85
  AND cnt > 30
  AND maxyear > 2000
  ORDER BY
    maxyear DESC,
    average DESC;
```

7.

```
SELECT DISTINCT person.name FROM cast_info
    JOIN person ON person.id = cast_info.person_id
  WHERE cast_info.role_id = 8
  AND EXISTS (
      SELECT
          ci.person_id,
          COUNT(ci.person_id) AS cnt
        FROM cast_info AS ci
          JOIN movie ON movie.id = ci.movie_id
        WHERE ci.role_id = 8
        AND cast_info.person_id = ci.person_id
        AND movie.tomatometer > '90%'
        GROUP BY ci.person_id
        HAVING cnt > 10
  );
```