DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Databases Project – Spring 2020

Team No: 27

Members: Atle Wiig-Fisketjøn, Sophie Du Couédic De Kergoualer, Corentin Junod

Contents

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 1

Assumptions

Businesses

We supposed there are 6 groups of attributes and that this number won't change, otherwise we would need to create a new relationship in the ER diagram (we use one many-to-many relationship per group of attribute). We estimated "False" and no-value for an attribute in the CSV file as the same, so only the attributes set as "True" are registered in the attributes relationships. Even if the number of groups of attributes won't change, we assume that the attributes inside a group itself can change (for example we can change a music type in Music, or add a new type). Therefore, the simplest way to model an attribute is a String.

One given business has exactly one location, but there can be multiple businesses at the same location (imagine two businesses in the same building for example), and a location can exist independently of any business, that's why we didn't create a one-to-one relationship.

We suppose that a business can only opens once in a day and close once is a day (no lunch interruption). This mean that there is only an opening time and a closing time per day in the week. The business schedules do not need to be set (the IsOpen relationship do not need to have the information for every business).

We expected that a business ID is never longer than 22 characters, as all business IDs in the data are exactly 22 characters long.

Tips

A user can write more than one tips to a business (therefore User + Business can't be primary key). Two different users can write a tip at the same time (even for the same business), so Date can't be the primary key neither. But we can easily assume that one single user can't write two tips at the same time (he is human!), so we choose Date + User as the primary key.

Reviews

We set the "Text" entry in the "Reviews" table as a field of type "Text" so many databases will authorize very long reviews, but there is a limit (on many databases the limit is above 16'000 UTF-8 characters so it seems reasonable). Concerning the primary key, same logic as for Tips : Date + User constitutes the primary key.

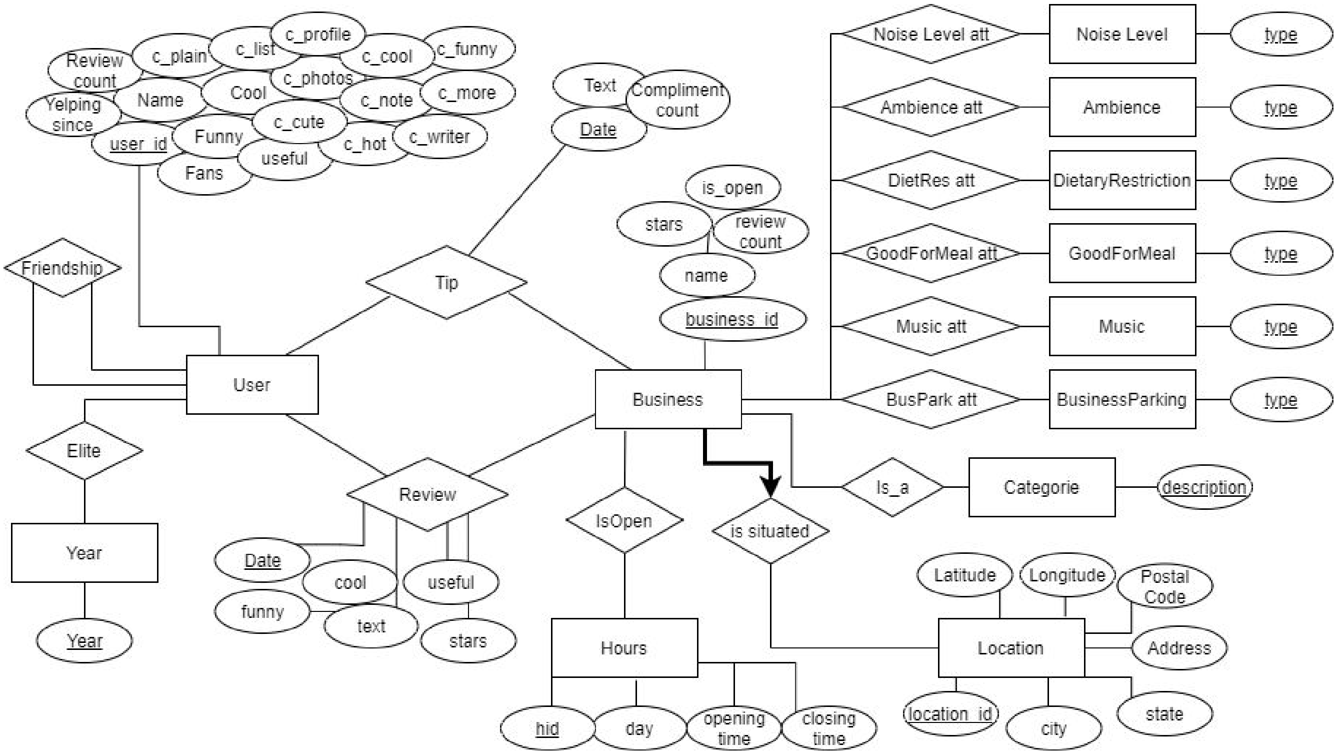
Users

We assume that friendships can be only one-directional. The situation where A is friend with B, but B is not friend with A is admitted to be possible.

A user can never be elite (empty list of elite years). And for a given year, it is admitted to be possible that no user is elite.

Entity Relationship Schema

Schema



Description

Businesses

The first adjustment we made is to separate the location information from the businesses. To do so, we created a “Location” entity, and one business has exactly location. As seen in class, we can combine Business and IsSituated because each Business has exactly one Location.

A business can be opened many days in a week, so we need to create a separate entity for Hours in order to keep the 1NF. We can imagine that many businesses have the same opening and closing time for a given day in the week, so one entry in Hours can be involved in many businesses. Therefore, we have a many-to-many relationship between Hours and Businesses, with no participation constraint, as explained above a business can have no schedule information and of course one given schedule may be not used for any business.

For the attributes, we created one separated entity per attribute type, with a relationship between that entity and business, with an entry for each attribute associated to as business, in order to respect the first normal form. This represents a many-to-many relationship between each attribute type with business, with no participation constraint (for example a business can have no Music type and a given music type can be associated with no business).

We have also a many-to-many relationship between Categories and Businesses : one business has potentially many categories, and one given category is probably shared by many businesses. One Category can exist without being assigned to any business, and a business can have no category. So there is no participation constraint.

Tips

Tips represents a many-to-many relationship between Businesses and Users (users can write many tips and business receive many tips as well). We set the (User_ID + Date) together as the primary key for a Tip (it is difficult to reflect this in the ER-Schema, but you can see it in the Relational Schema). Please note that User ID + Business ID is not sufficient to denote a Tip because one user can tip multiple times the same business. Also, as explained before, we can remove the Business ID from the PK because it is assumed that one user can't write two tips at the same time, so a given tip can be entirely determined by User ID + Date.

Reviews

We treated the reviews' data the same way we treated "Tips", as a many-to-many relationship between the users and the businesses, one user can write many reviews to many businesses, and many reviews to the same business as well. Therefore, same logic applies for Reviews than for Tips : User_ID + Date constitute the PK.

Users

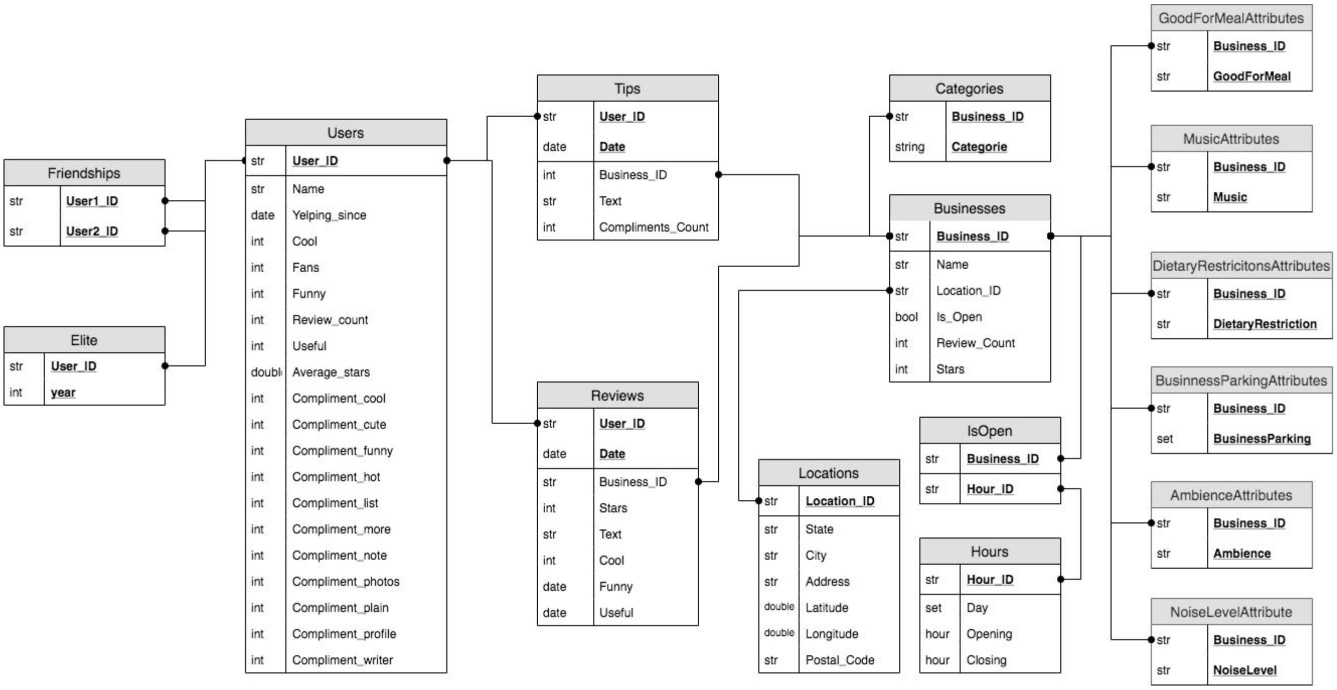
We have chosen to treat "Users" as an entity, this seems logical in the context of Yelp.

"Friendship" represents a many-to-many relationship between Users and Users itself. As assumed, the friendship is not bi-directional, therefore (U1_ID, U2_ID) is not the same entry as (U2_ID, U1_ID).

One user can elite for more than one year, and every year many users are elite. Therefore, Elite is a many-to-many relationship between User and Year, with no participation constraint (a user can never be elite, and for a given year we can have no elite user).

Relational Schema

ER schema to Relational schema



Businesses

We created one table per entity, so we have the following tables : Hours, Businesses, Locations and Categories. We also create the table "IsOpen" to reflect the relation between "Businesses" and "Hours".

In the table "Hours" all fields are required (NOT NULL) otherwise it makes no sense to have partial opening/closing hours. The day is represented as a string of length 9 because it is the longer day of the week (Wednesday). We could have created another table assigning for each day a number, but we found this solution overcomplicated.
The field Hour_ID is an int, we found this solution much simpler than a string ID as Business_ID for example.

The table "Businesses" is pretty straightforward, we have set the max business name length to 255, which seems reasonable and does not create any problem during the import. The Location_ID references for each business its location, as the ER diagram is showing.

For the table "IsOpen" we set the two foreign keys as the ER diagram suggests, the table itself only contains links between Business_ID and Hour_ID.

In the "Location" table all fields can be NULL, because partial Locations can be acceptable. Each location has an ID, because we need to be able to reference them in the "Businesses" table.

The categories table has a simple design, each row associate a Business_ID with one of its categories.

To create the attributes, we created a table for each kind of attribute, and bonded businesses IDs with the attributes. We could have created tables containing the attributes values and then create foreign keys to those tables, but we thought it would have been cumbersome. All attributes are stored in fields of length 255, as the attributes are mostly words we thought this is a good choice.

Tips

We translated Tips into one table, having User_ID and Date as primary key. We choose to make the Text field NOT NULL, because otherwise there is no point to have a tip.

Reviews

This table follows the same pattern as the table "Tips". There is no NOT NULL fields besides the IDs and the date because a review containing only a text, or only a star-rate is possible.

Users

We created three tables based on the Users data : Users, Friendships and Elite.
Elite is similar to Categories for a business, each row assigns to a User_ID a year during which the user was an elite.
Friendships, as the ER diagram shows, links two User_ID together, the first one being friend with the second, so both columns are a foreign key to Users(User_ID).

Finally, the table Users itself contains all compliment-like attributes as integers among the others attributes. We choose to set the name length to 255, it seems reasonable for names.

DDL

```
CREATE TABLE Hours (
    Hour_ID          int           NOT NULL,
    Day              char(9)      NOT NULL,
    Opening         time        NOT NULL,
    Closing         time        NOT NULL,

    PRIMARY KEY(Hour_ID)
);

CREATE TABLE Locations (
    Location_ID     int         NOT NULL,
    State           char(255),
    City            char(255),
    Address         char(255),
    Latitude        double,
    Longitude       double,
    Postal_Code     char(255),

    PRIMARY KEY(Location_ID),
);


CREATE TABLE Businesses (
    Business_ID     char(22)     NOT NULL,
    Location_ID     int          NOT NULL,
    Name            char(255)    NOT NULL,
    is_Open         tinyint(1)   NOT NULL,
    Review_Count    int          NOT NULL,
    Stars           double       NOT NULL,

    PRIMARY KEY(Business_ID)
    FOREIGN KEY (Location_ID) REFERENCES Locations(Location_ID)
);

CREATE TABLE IsOpen (
    Business_ID     char(22)    NOT NULL,
    Hour_ID         int            NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID),
    FOREIGN KEY (Hour_ID)     REFERENCES Hours(Hour_ID)
);

CREATE TABLE Categories (
    Business_ID     char(22)    NOT NULL,
    Categorie       char(255)   NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);
```

```
/*-------------------------------------------
----------- ATTRIBUTES --------------------
-------------------------------------------*/

CREATE TABLE BusinessParkingAttributes (
    Business_ID     char(22)    NOT NULL,
    BusinessParking char(255)  NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);

CREATE TABLE NoiseLevelAttributes (
    Business_ID char(22)    NOT NULL,
    NoiseLevel  char(255)      NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);


CREATE TABLE AmbienceAttributes (
    Business_ID char(22)  NOT NULL,
    Ambience    char(255) NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);

CREATE TABLE DietaryRestrictionsAttributes (
    Business_ID         char(22)    NOT NULL,
    DietaryRestrictions char(255)    NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);

CREATE TABLE GoodForMealAttributes (
    Business_ID char(22)    NOT NULL,
    GoodForMeal   char(255)    NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);

CREATE TABLE MusicAttributes (
    Business_ID char(22)    NOT NULL,
    Music       char(255)      NOT NULL,

    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID)
);
```

```
/*-------------------------------------------
-------------------------------------------
-------------------------------------------*/

CREATE TABLE Users (
    User_ID             char(22)    NOT NULL,
    Name                char(255)   NOT NULL,
    Yelping_since       timestamp   NOT NULL,
    Cool                int         NOT NULL,
    Fans                int         NOT NULL,
    Funny               int         NOT NULL,
    Review_count        int         NOT NULL,
    Useful              int         NOT NULL,
    Average_stars       double      NOT NULL,
    Compliment_cool     int         NOT NULL,
    Compliment_cute     int         NOT NULL,
    Compliment_funny    int         NOT NULL,
    Compliment_hot      int         NOT NULL,
    Compliment_list     int         NOT NULL,
    Compliment_more     int         NOT NULL,
    Compliment_note     int         NOT NULL,
    Compliment_photos   int         NOT NULL,
    Compliment_plain    int         NOT NULL,
    Compliment_profile  int         NOT NULL,
    Compliment_writer   int         NOT NULL,

    PRIMARY KEY(User_id)
);

CREATE TABLE Friendships (
    User1_ID            char(22)    NOT NULL,
    User2_ID            char(22)    NOT NULL,

    FOREIGN KEY (User1_ID) REFERENCES Users(User_ID),
    FOREIGN KEY (User2_ID) REFERENCES Users(User_ID)
);

CREATE TABLE Elite (
    User_ID             char(22)    NOT NULL,
    year                int         NOT NULL,

    FOREIGN KEY (User_ID) REFERENCES Users(User_ID)
);

/*-------------------------------------------
```

```
-------------------------------------------
-------------------------------------------*/

CREATE TABLE Reviews (
    Business_ID char(22) NOT NULL,
    User_ID     char(22) NOT NULL,
    Date        date     NOT NULL,
    Stars       int      NOT NULL,
    Text        text     NOT NULL,
    Cool        int      NOT NULL,
    Funny       int      NOT NULL,
    Useful      int      NOT NULL,

    PRIMARY KEY(User_ID, Date),
    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID),
    FOREIGN KEY (User_ID)     REFERENCES Users(User_ID)
);


CREATE TABLE Tips (
    User_ID          char(22) NOT NULL,
    Date             datetime NOT NULL,
    Business_ID      char(22) NOT NULL,
    Text             text     NOT NULL,
    Compliment_Count int      NOT NULL,

    PRIMARY KEY(User_ID, Date),
    FOREIGN KEY (Business_ID) REFERENCES Businesses(Business_ID),
    FOREIGN KEY (User_ID)     REFERENCES Users(User_ID)
);
```

General Comments

Please do not be confused between B.IS_Open which indicates if the business is currently open, and the relation IsOpen which contains the list of opening/closing hours per day for all the businesses. The second name is maybe not really well-chosen.

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 2

Assumptions

In general : We assume that the letter case is unimportant when considering the City (ig. PHOENIX is the same city as Phoenix or phoenix, we consider them as the same unique city).

Query 4 : We assumed that by "label" you are talking about the categories. We do not check wether the name of the Business contains "Dry Cleanining" or "Dry Cleaners". Indeed, you can imagin that a restaurant contains "Dry Cleaners" in his name (certainly a strange idea for a name, but why not?)

Query 12 : We assumed that if a business opens before 19h and closes after 23h then it is opened "between 19h and 23h" (no need to open at 19h and close at 23h). Note also that "after midnight" is also after 23h.

Data Loading/Cleaning

For the data cleaning, we used python scripts to format the data and import it in the database.
The most recurring problem was the string values handling. Sometimes they were with single quote, sometimes with double quotes, and those symbols were also used inside the strings. The most straightforward way to handle those case was to demarcate the strings with "<" and ">", they are recognized by Oracle SQLDevelopper.

Businesses

The hardest part was to extract the attributes' information, because they were JSON-like format inside the CSV file and to fit our database design we need to remove all "FALSE" attributes.
Data about the hours were also in JSON-like format, we applied the same principle and separated the opening and closing time.

Once one line was correctly retrieved the script separates and write the attributes to different files, one per table. Due to our schema for hours, for every line read in the input file we need to check if the entry already exists in the output hours file.
If this is the case we only have to assign the current business to the found entry (in the table IsOpen).
If this is not the case we add a new entry in the Hours table and create a new ID. Then we link the current business to this new entry in IsOpen.

Users

Formatting the users data was way simpler than businesses, to handle the friendships and the elite the scripts had, for a given line, to split the values and insert inside the corresponding files one entry per tuple (user1, user2) or (user, year). All the remaining data were just copy-pasted are reordered.

Tips

The tips were formatted in a very special way, in a JSON-like format, so the script has to recompose them, but this aside this problem the only difficulty in Tips data was the strings, sometimes beginning with a double or single quote, so this case must be handled.

Reviews

The texts in the reviews were pretty messy, with many carriage returns inside the file itself, but we manage to recompose them and import them.

Query Implementation

Query 1

Descriptions of logic

Takes the average of the column Review_count and round it for convenience

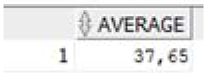
SQL statement

SELECT ROUND(AVG(U.Review_count), 2) AS average FROM Users U

Runtime

974 millisecs

Query result



	AVERAGE
1	37,65

Query 2

Descriptions of logic

We join Businesses and Locations and select only the distinct businesses where their stat is Québec or Alabama

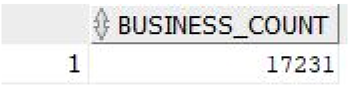
SQL statement

SELECT COUNT (DISTINCT B.Business_ID)
FROM Businesses B, Locations L
WHERE B.Location_ID = L.Location_ID AND L.State IN ('QC', 'AB');

Runtime

170 millisecs

Query result



	BUSINESS_COUNT
1	17231

Query 3

Descriptions of logic

We join Businesses and Categories in an inner query, then we group the selection by Business_ID and select the number of rows grouped for a given Business (which is the number of categories for this business). Then we sort the result and retain only the first result, this gives the ID of the searched business. Finally, we extract the name in the outer query.

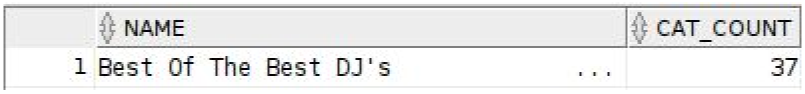
SQL statement

```
SELECT B2.Name, Cat_Count
FROM Businesses B2, (
        SELECT B.Business_ID AS ID, COUNT(*) AS Cat_Count
        FROM Businesses B, Categories C
        WHERE B.Business_ID = C.Business_ID
        GROUP BY B.Business_ID
)
WHERE B2.Business_ID = ID
ORDER BY cat_count DESC
FETCH FIRST 1 ROWS ONLY
```

Runtime

709 millisecs

Query result



	NAME	CAT_COUNT
1	Best Of The Best DJ's ...	37

Query 4

Descriptions of logic

We select the number of distinct entries in Categories where the categorie is one of the two searched.

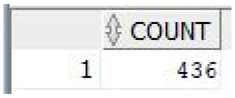
SQL statement

```
SELECT COUNT(DISTINCT C.Business_ID)
FROM Categories C
WHERE C.Categorie IN ('Dry Cleaners', 'Dry Cleaning')
```

Runtime

146 millisecs

Query result



	COUNT
1	436

Query 5

Descriptions of logic

We first select in a subquery the businesses having at least 2 dietary restriction. Those result are used to retain only the businesses having more than 150 reviews and belonging to that subquery. We then sum their review count.

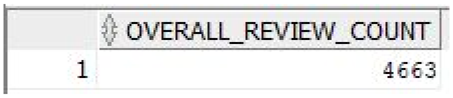
SQL statement

```
SELECT SUM(Review_Count) as overall_review_count
FROM (
  SELECT DISTINCT B.Business_ID, B.Review_Count as Review_Count
  FROM Businesses B
  WHERE B.Review_Count >= 150 AND B.Business_ID IN (
    SELECT DR.Business_ID
    FROM DietaryRestrictionsAttributes DR
    GROUP BY DR.Business_ID
    HAVING COUNT(*) > 1
    )
  )
```

Runtime

149 millisecs

Query result



	OVERALL_REVIEW_COUNT
1	4663

Query 6

Descriptions of logic

We group the table Friendships by the first user, and count the number of grouped user in the second column, then we order by this second column, retaining only the first 10 users.

SQL statement

```
SELECT F.User1_ID as User_ID, COUNT(*) AS Friends
```

FROM Friendships F

GROUP BY F.User1_ID

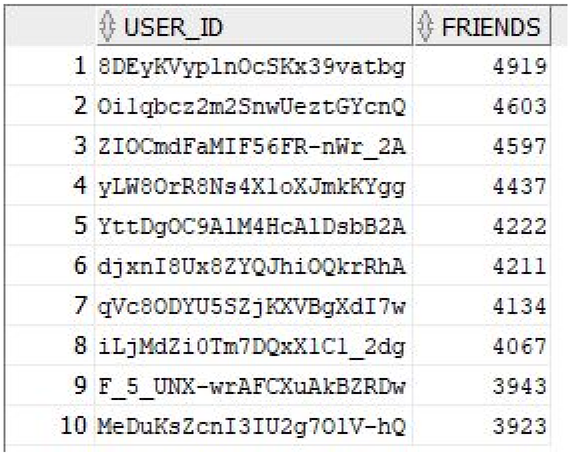
ORDER BY Friends DESC

FETCH FIRST 10 ROWS ONLY

Runtime

2812 millisecs

Query result



	USER_ID	FRIENDS
1	8DEyKVyplnOcSKx39vatbg	4919
2	0ilqbcz2m2SnwUeztGYcnQ	4603
3	ZIOCmdFaMIF56FR-nWr_2A	4597
4	yLW8OrR8Ns4X1oXJmkKYgg	4437
5	YttDgOC9A1M4HcA1DsbB2A	4222
6	djxnI8Ux8ZYQJhiOQkrRhA	4211
7	qVc8ODYU5SZjKXVBgXdI7w	4134
8	iLjMdZi0Tm7DQxX1C1_2dg	4067
9	F_5_UNX-wrAFCXuAkBZRDw	3943
10	MeDuKsZcnI3IU2g7O1V-hQ	3923

Query 7

Descriptions of logic

We join Businesses and Locations and retain only the city of Sans Diego where the business is open. Then we order by the review count and select the required columns. We only retrieve the first 5 rows.
In this case, there was only one business corresponding to the criteria.

SQL statement

SELECT B.Name, B.Stars, B.Review_Count

FROM Businesses B, Locations L

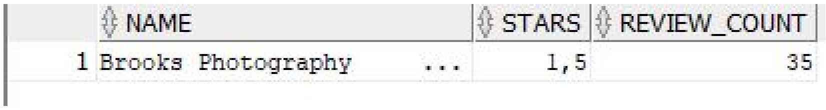
WHERE B.Location_ID = L.Location_ID AND LOWER(L.City) = 'san diego' AND B.Is_Open = 1

ORDER BY B.Review_Count DESC;

Runtime

1416 millisecs

Query result



	NAME	STARS	REVIEW_COUNT
1	Brooks Photography ...	1,5	35

Query 8

Descriptions of logic

We join Locations and Businesses, then group by the state and order by this column, so the first result is the one we are searching for, so we keep only this one.

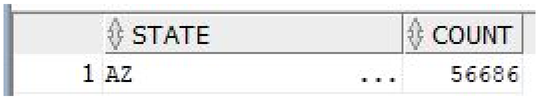
SQL statement

```
SELECT L.State, COUNT(*) as Count
FROM Locations L, Businesses B
WHERE L.Location_ID =  B.Location_ID
GROUP BY L.State
ORDER BY COUNT(*) DESC
FETCH FIRST 1 ROWS ONLY;
```

Runtime

1282 millisecs

Query result



	STATE	COUNT
1	AZ ...	56686

Query 9

Descriptions of logic

We join Elite and Users, and group by the year while taking the average stars, then we sort the result

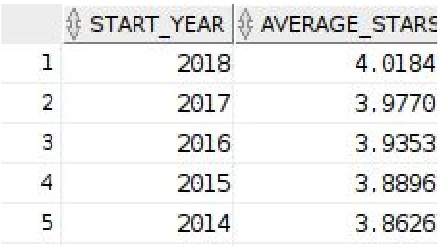
SQL statement

```
SELECT Start_Year, ROUND(AVG(U2.Average_stars),5) as Average_Stars
FROM Users U2, (
            SELECT U.User_ID as U_ID, MIN(E.Year) as Start_Year
            FROM Users U, Elite E
            WHERE U.User_ID = E.User_ID
            GROUP By U.User_ID )
WHERE U2.User_ID = U_ID
GROUP BY Start_Year
ORDER BY Start_Year DESC
```

Runtime

472 millisecs

Query result



	START_YEAR	AVERAGE_STARS
1	2018	4.0184
2	2017	3.9770
3	2016	3.9353
4	2015	3.8896
5	2014	3.8626

Query 10

Descriptions of logic

We filter Businesses with the requirements (must be open and in New York) and select the first 10 Businesses when ordered by the Stars attribute.

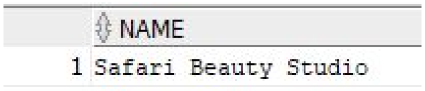
SQL statement

```
SELECT B.name
FROM Locations L, Businesses B
WHERE L.Location_ID = B.Location_ID AND B.Is_Open = 1 AND LOWER(L.City) = 'new york'
ORDER BY B.Stars DESC
FETCH FIRST 10 ROWS ONLY;
```

Runtime

938 millisecs

Query result



	NAME
1	Safari Beauty Studio

Query 11

Descriptions of logic

We have two subqueries, the first selects the maximum, minimum and average of categories grouped by business.

The second computes the median of the number of categories per business. To do so, we run the query as if we would find the maximum value, then we offset the results by number of business divided by 2.

The outermost query wrap up those results on a single result row.

SQL statement

```
SELECT first1, first2, first3, second1
FROM (
   SELECT MAX(curCount) as first1, MIN(curCount) as first2, ROUND(AVG(curCount),2) as first3
   FROM ( SELECT COUNT(*) as curCount FROM Categories C GROUP BY C.Business_ID )
 ),(
   SELECT COUNT(Categorie) as second1
   FROM Categories
   GROUP BY Business_ID
   ORDER BY COUNT(Categorie) DESC
   OFFSET (SELECT FLOOR(COUNT(*)/2) FROM Businesses) ROWS
   FETCH FIRST 1 ROWS ONLY
  );
```

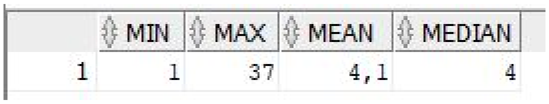
Alternatively, we could have used the function MEDIAN, if available

```
SELECT MIN(count) as min , MAX(count) as max, ROUND(AVG(count),1) as mean, MEDIAN(count) as median
FROM ( SELECT COUNT(*) as count
   FROM Categories C
   GROUP BY C.Business_ID
   );
```

Runtime

524 millisecs

Query result



	MIN	MAX	MEAN	MEDIAN
1	1	37	4,1	4

Query 12

Descriptions of logic

As the filter is more complicated on this query, we need to join BusinessParkingAttributes, Locations, Hours, Businesses and IsOpen. Then we apply all the filters with AND clauses.

As the times are stored as string, to ensure the results are within a precise period of time we cut the strings only to have the hours number and then compare it.

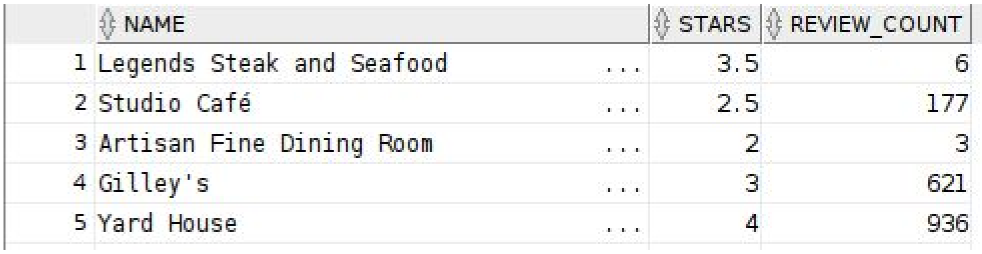
SQL statement

```
SELECT B.Name, B.Stars, B.Review_Count
FROM BusinessParkingAttributes P, Locations L, Hours H, Businesses B, IsOpen IO
WHERE IO.Hour_ID = H.Hour_ID AND IO.Business_ID = B.Business_ID AND L.Location_ID = B.Location_ID AND
P.Business_ID = B.Business_ID
   AND P.BusinessParking = 'valet'
   AND LOWER(L.City) = 'las vegas'
   AND H.day = 'Friday'
   AND ( TO_NUMBER(SUBSTR(H.opening, 0,2)) < 19 OR SUBSTR(H.opening, 0,4) = '19:00' )
   AND ( TO_NUMBER(SUBSTR(H.closing, 0,2)) >= 23
     OR TO_NUMBER(SUBSTR(H.closing,0,2)) < TO_NUMBER(SUBSTR(H.opening, 0,2)))
```

Runtime

622 millisecs

Query result



	NAME	STARS	REVIEW_COUNT
1	Legends Steak and Seafood ...	3.5	6
2	Studio Café ...	2.5	177
3	Artisan Fine Dining Room ...	2	3
4	Gilley's ...	3	621
5	Yard House ...	4	936

General Comments

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

Deliverable 3

Assumptions

General note about Cities : we still ignore case when considering a city (ig. Phoenix, PHOENIX, phoenix are the same cities)

For the review count : unless it is explicitely asked, we always use B.Review_count to count the number of reviews.

Query 12 : we do not count when the user gave to the same business the positive tip : we understand "some business" as "some other business".

Query 15 : By "weekend" we understand that the business must be open both on saturday and sunday.

Query 16 : If the business is opened only one day in the week between 14h,16h it is okay. I mean that you didn't say every day between 14h and 16h.

Query 19 : We do not exclude cities that have less than 100 businesses : if a city has less than 100 business, then the condition that the number of reviews of the top100 businesses are 2* the number of reviews of non-top100 businesses automatically satisfies.

Query Implementation

Query 1

Descriptions of logic

We join Businesses to Locations and apply the required filters, then count the number of results.

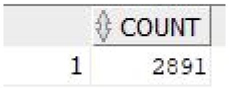
SQL statement

```
SELECT COUNT(*) as count
FROM Businesses B, Locations L
WHERE B.Location_ID = L.Location_ID AND L.state = 'ON' AND B.Review_Count > 5 AND B.stars > 4.2;
```

Runtime

201 millisecs

Query result



	COUNT
1	2891

Query 2

Descriptions of logic

Subtract average for quiet/average from average for loud/very loud. A negative output implies a higher average number of stars for the businesses with noise levels “quiet” or “average”, and the opposite for a positive output.

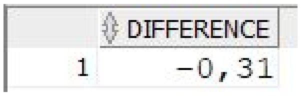
SQL statement

```
WITH first1 as (
        SELECT B.Business_ID as ID, B.Stars as Stars
        FROM Businesses B, Goodformealattributes G
        WHERE B.Business_ID = G.Business_ID AND G.Goodformeal = 'dinner'
)
SELECT ROUND((SELECT AVG(F1.Stars) FROM first1 F1, Noiselevelattributes N1
WHERE F1.ID = N1.Business_ID AND N1.Noiselevel IN ('loud', 'very_loud')
) - (
SELECT AVG(F2.stars)
FROM first1 F2, Noiselevelattributes N2 WHERE F2.ID = N2.Business_ID AND N2.Noiselevel IN ('average',
'quiet')), 2) as difference
FROM DUAL
```

Runtime

949 millisecs

Query result



	DIFFERENCE
1	-0,31

Query 3

Descriptions of logic

We joined Business, Categories and MusicAttributes then apply the required filters

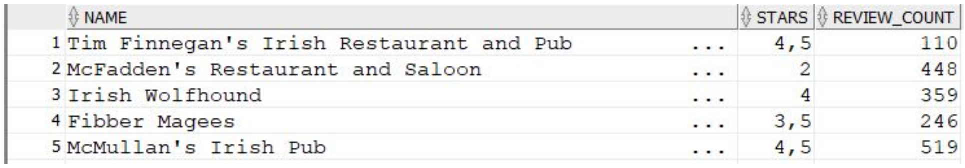
SQL statement

```
SELECT B.name, B.stars, B.review_count
FROM Businesses B, Categories C, MusicAttributes M
WHERE B.Business_ID = C.Business_ID AND B.Business_ID = M.Business_ID
        AND M.music = 'live' AND C.categorie = 'Irish Pub'
```

Runtime

257 millisecs

Query result



	NAME	STARS	REVIEW_COUNT
1	Tim Finnegan's Irish Restaurant and Pub ...	4,5	110
2	McFadden's Restaurant and Saloon ...	2	448
3	Irish Wolfhound ...	4	359
4	Fibber Magees ...	3,5	246
5	McMullan's Irish Pub ...	4,5	519

Query 4

Descriptions of logic

As the description states, we need to retrieve 4 values, represented by the 4 subqueries. All of them retrieve the data from Users and apply the filters. The two queries for the Elite need to join this table before applying the filters.

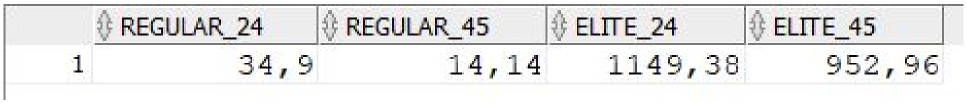
SQL statement

```
SELECT
        (
        SELECT ROUND( AVG(U1.useful),2)
        FROM Users U1
        WHERE U1.average_stars>=2 AND U1.average_stars < 4
        AND U1.User_ID NOT IN (SELECT User_ID FROM Elite))
        as regular_24 ,
        (
        SELECT ROUND( AVG(U2.useful),2)
        FROM Users U2
        WHERE (U2.average_stars BETWEEN 4 AND 5)
        AND U2.User_ID NOT IN (SELECT User_ID FROM Elite)) as regular_45,
        (
        SELECT ROUND( AVG(U3.useful),2)
        FROM Users U3, Elite E
        WHERE U3.average_stars>=2 AND U3.average_stars < 4
        AND U3.User_ID = E.User_ID) as elite_24,
        (
        SELECT ROUND( AVG(U4.useful),2)
        FROM Users U4, Elite E
        WHERE (U4.average_stars BETWEEN 4 AND 5)
        AND U4.User_ID = E.User_ID) as elite_45
FROM DUAL;
```

Runtime

1285 millisecs

Query result



	REGULAR_24	REGULAR_45	ELITE_24	ELITE_45
1	34,9	14,14	1149,38	952,96

Query 5

Descriptions of logic

We first compute two tables, one containing the businesses having more than 1 categories, the other one containing the businesses having more than 1 parking type. Then we join those two tables with Businesses and extract the requested values.

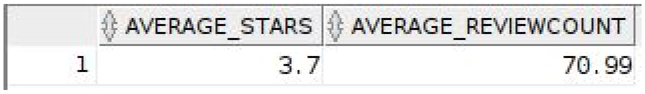
SQL statement

```
SELECT ROUND(AVG(B.stars),2) as average_stars,
        ROUND(AVG(B.review_count),2) as average_reviewcount
FROM Businesses B,
        (
        SELECT C.Business_ID as ID
        FROM Categories C GROUP BY C.Business_ID
        HAVING COUNT(*) >= 2) C1,
        (
        SELECT P.Business_ID as ID
        FROM BusinessParkingAttributes P GROUP BY P.Business_ID
        HAVING COUNT(*) >= 1) P1
WHERE B.Business_ID = C1.ID AND B.Business_ID = P1.ID;
```

Runtime

494 millisecs

Query result



	AVERAGE_STARS	AVERAGE_REVIEWCOUNT
1	3.7	70.99

Query 6

Descriptions of logic

For this query we simply join Businesses and GoodForMealAttributes and count the number of businesses being “latenight”. Then we divide this number by the total number of businesses.

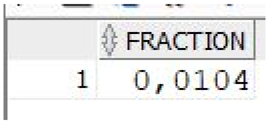
SQL statement

```
SELECT ROUND( (SELECT COUNT(DISTINCT B.Business_ID)
        FROM Businesses B, GoodForMealAttributes G
        WHERE B.Business_ID = G.Business_ID AND G.Goodformeal = 'latenight') /
         (SELECT COUNT(*) FROM Businesses B), 4) as Fraction
FROM DUAL
```

Runtime

174 millisecs

Query result



	FRACTION
1	0,0104

Query 7

Descriptions of logic

For this query we perform a table subtraction. We first take all the cities, then take all the cities where a business does have an hour with the day "Sunday". The subtraction gives us all the cities where no business is opened on Sundays.

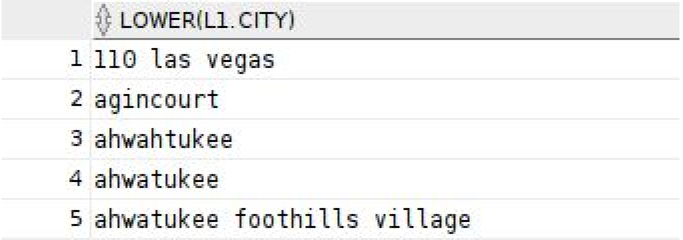
SQL statement

```
SELECT DISTINCT LOWER(L1.City)
FROM Locations L1
MINUS
SELECT DISTINCT LOWER(L2.City)
FROM Locations L2
WHERE L2.Location_ID IN (
        SELECT B.Location_ID
        FROM Businesses B, Hours H
        WHERE B.Business_ID = H.Hour_ID AND H.day = 'sunday')
```

Runtime

851 millisecs

Query result



	LOWER(L1.CITY)
1	110 las vegas
2	agincourt
3	ahwahtukee
4	ahwatukee
5	ahwatukee foothills village

Query 8

Descriptions of logic

We first compute a table with each business id and the number of distinct users having left a review, then we only keep those where this number is greater than 1030.

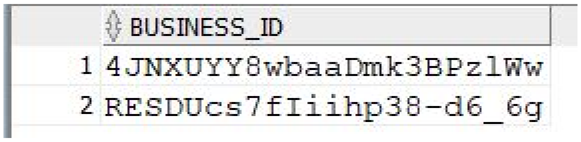
SQL statement

```
SELECT Rev.Business_ID
FROM (SELECT Business_ID, COUNT (DISTINCT User_ID) as number1
        FROM Reviews
        GROUP BY Business_ID) Rev
WHERE Rev.Number1 > 1030
```

Runtime

5319 millisecs

Query result



	BUSINESS_ID
1	4JNXUYY8wbaaDmk3BPzlWw
2	RESDUcs7fIiihp38-d6_6g

Query 9

Descriptions of logic

This query is pretty straightforward, we only need to join Businesses and Locations and apply the corresponding filter

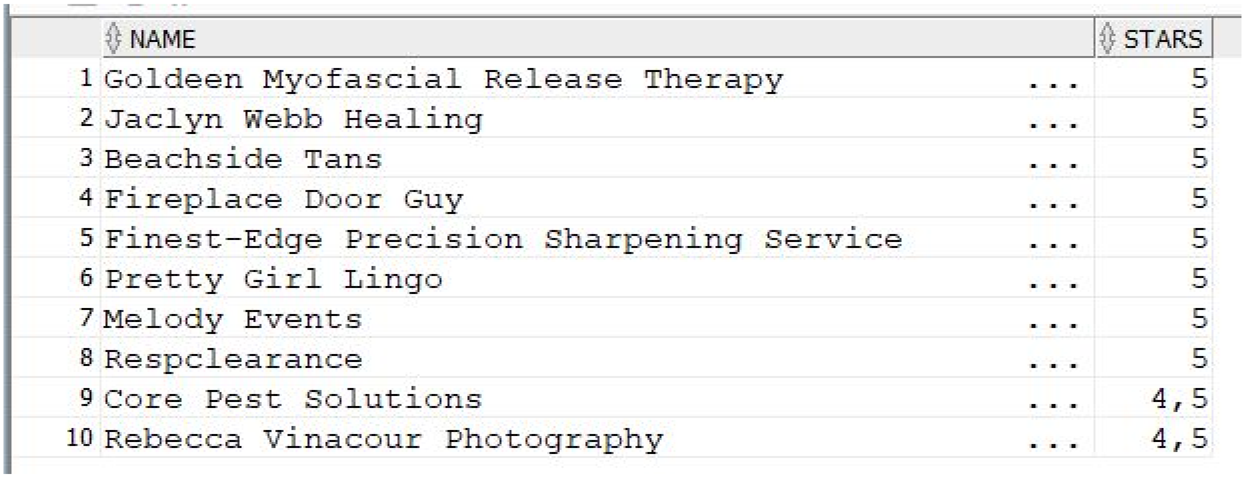
SQL statement

```
SELECT B.Name, B.Stars
FROM Businesses B, Locations L
WHERE B.Location_ID = L.Location_ID AND L.State = 'CA'
ORDER BY B.Stars DESC
FETCH FIRST 10 ROWS ONLY
```

Runtime

347 millisecs

Query result



	NAME	STARS
1	Goldeen Myofascial Release Therapy ...	5
2	Jaclyn Webb Healing ...	5
3	Beachside Tans ...	5
4	Fireplace Door Guy ...	5
5	Finest-Edge Precision Sharpening Service ...	5
6	Pretty Girl Lingo ...	5
7	Melody Events ...	5
8	Respclearance ...	5
9	Core Pest Solutions ...	4,5
10	Rebecca Vinacour Photography ...	4,5

Query 10

Descriptions of logic

First, in the nested selection, we order the businesses of each state by their number of stars, along with their rank in the state. Then we display all the businesses with rank < 10 (meaning the top ten businesses), it is naturally displayed by state, because in the inner search we partition by state.

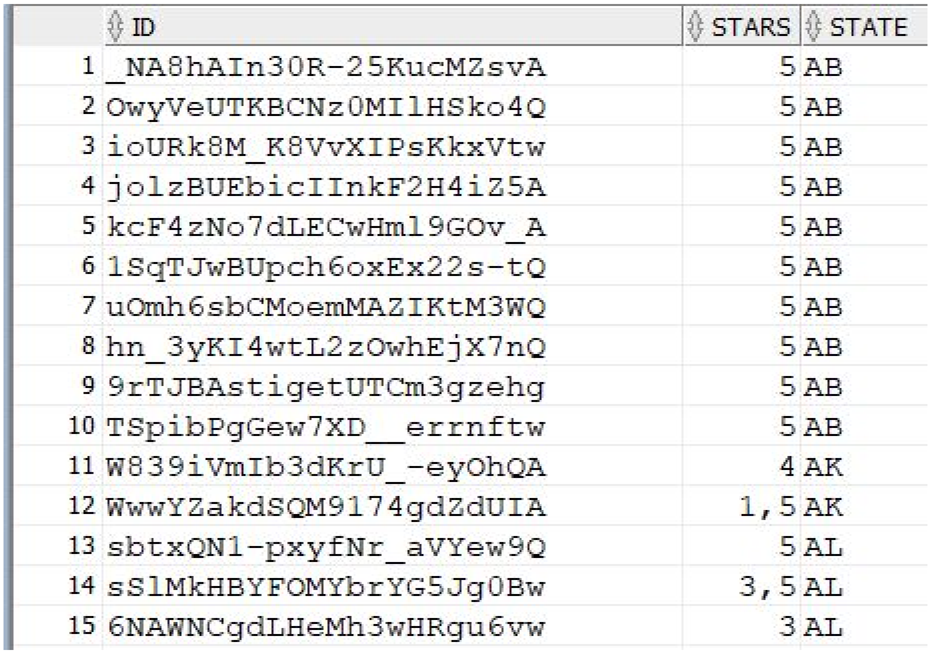
SQL statement

```
SELECT Ranks.ID, Ranks.Stars, Ranks.State
FROM (
        SELECT B.Business_ID as ID, B.Stars as Stars,
                L.State as State,
                ROW_NUMBER() OVER (PARTITION BY L.State ORDER BY B.Stars DESC) AS Rank
        FROM Locations L, Businesses B
        WHERE B.Location_ID = L.Location_ID
        ) Ranks
WHERE Ranks.Rank <= 10;
```

Runtime

1087 millisecs

Query result



	ID	STARS	STATE
1	_NA8hAIn30R-25KucMZsvA	5	AB
2	OwyVeUTKBCNz0MIlHSko4Q	5	AB
3	ioURk8M_K8VvXIPsKkxVtw	5	AB
4	jolzBUEbicIInkF2H4iZ5A	5	AB
5	kcF4zNo7dLECwHml9GOv_A	5	AB
6	1SqTJwBUpch6oxEx22s-tQ	5	AB
7	uOmh6sbCMoemMAZIKtM3WQ	5	AB
8	hn_3yKI4wtL2zOwhEjX7nQ	5	AB
9	9rTJBAstigetUTCm3gzehg	5	AB
10	TSpibPgGew7XD__errnftw	5	AB
11	W839iVmIb3dKrU_-eyOhQA	4	AK
12	WwwYZakdSQM9174gdZdUIA	1,5	AK
13	sbtxQN1-pxyfNr_aVYew9Q	5	AL
14	sSlMkHBYFOMYbrYG5Jg0Bw	3,5	AL
15	6NAWNCgdLHeMh3wHRgu6vw	3	AL

Query 11

Descriptions of logic

We perform a table subtraction, we first take all the cities then remove from it all the cities where there is a business with less than 2 reviews.

SQL statement

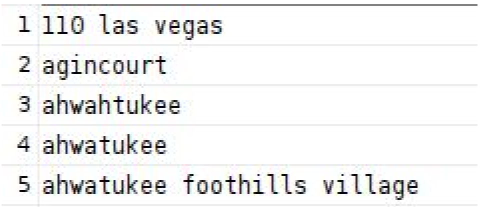
```
SELECT DISTINCT LOWER(L1.City)
FROM Locations L1
```

```
MINUS
SELECT DISTINCT LOWER(L2.City)
FROM Locations L2,
        (SELECT B.Location_ID as ID
        FROM Businesses B
        WHERE B.Review_Count < 2
        ) Counter
WHERE L2.Location_ID = Counter.ID
```

Runtime

698 millisecs

Query result



```
1 110 las vegas
2 agincourt
3 ahwahtukee
4 ahwatukee
5 ahwatukee foothills village
```

Query 12

Assuming the positive tip should be to some other business (not the same twice) this day or the day before

Descriptions of logic

We first consider all tips containing awesome, then for those tips we retain only the one where the user leaves a comment containing “awesome” the previous date and that are not on the same business.
From those tips we count the number of business id.

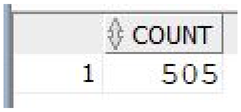
SQL statement

```
SELECT COUNT(DISTINCT Business_ID) AS Count
FROM Tips T1
WHERE LOWER(T1.Text) LIKE '%awesome%' AND User_ID IN
        (SELECT User_ID
        FROM Tips T2
        WHERE LOWER(T2.Text) LIKE '%awesome%'
        AND T2.Business_ID != T1.Business_ID
        AND T2.Tipdate < T1.Tipdate
        AND T2.Tipdate+0 BETWEEN T1.Tipdate-1 AND T1.Tipdate+0
        );
```

Runtime

6984 millisecs

Query result



	COUNT
1	505

Query 13

Descriptions of logic

We group the reviews by user, then retain the maximum value among the number of business reviewed for each user.

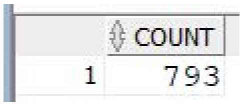
SQL statement

```
SELECT MAX(COUNT(DISTINCT R.business_id)) as count
        FROM Reviews R
        GROUP By R.User_ID
```

Runtime

3179 millisecs

Query result



	COUNT
1	793

Query 14

Descriptions of logic

We compute the averages of useful ratings for elite users and also the non-elite users, then we subtract the second one from the first one.

Note :Positive number implies a higher average for elite users compared to regular users.

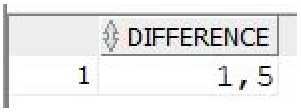
SQL statement

```
SELECT ROUND(
        (SELECT AVG(R1.useful)
        FROM Reviews R1
        WHERE R1.User_ID IN (SELECT User_ID FROM Elite)
        ) -
        (SELECT AVG(R2.useful)
        FROM Reviews R2
        WHERE R2.User_ID NOT IN (SELECT User_ID FROM Elite)),
        2) as Difference
FROM DUAL;
```

Runtime

2733 millisecs

Query result



	DIFFERENCE
1	1,5

Query 15

Descriptions of logic

We select the businesses which are open, has B.stars >= 4.5 and for which the relation GoodformealAttributes has an entry associated with 'brunch', to this we add the condition that the business is opened both saturday and sunday : the business needs to have two entries in the relation IsOpen : one that is associated to a 'saturday' opening day, and the other associated to a 'sunday' opening day.

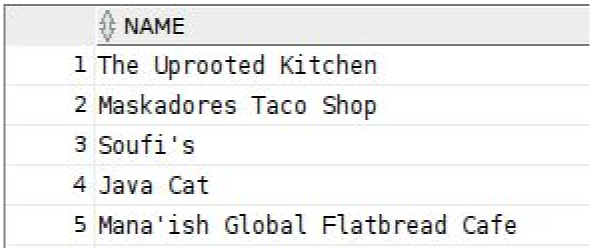
SQL statement

```
SELECT DISTINCT B.Name
FROM Businesses B, GoodformealAttributes G
WHERE G.Business_ID = B.Business_ID
        AND B.Is_Open = 1
        AND B.Stars >= 4.5
        AND LOWER(G.GoodForMeal) = 'brunch'
        AND B.Business_ID IN (
                SELECT DISTINCT IO.Business_ID
                FROM IsOpen IO, Hours H
                WHERE H.Hour_ID = IO.Hour_ID AND LOWER(H.Day) = 'saturday')
        AND B.Business_ID IN (
                SELECT DISTINCT IO2.Business_ID
                FROM IsOpen IO2, Hours H2
                WHERE H2.Hour_ID = IO2.Hour_ID AND LOWER(H2.Day) = 'sunday');
```

Runtime

1335 millisecs

Query result



	NAME
1	The Uprooted Kitchen
2	Maskadores Taco Shop
3	Soufi's
4	Java Cat
5	Mana'ish Global Flatbread Cafe

Query 16

Descriptions of logic

First, note that to be able to compute the average stars for each Business, we need group the entries in the cross products with Reviews by the Business_ID, and compute the mean value for each of this group. However, to be able to print B.Name and B.Reviews_Count, we need to integrate B.Name and B.Reviews_Count in the outer GROUP_BY expression (this is not a problem because a given business can't have more than one name and one Reviews_count value). We order the result by the computed average stars and return only the five first lines.

The other conditions are pretty straightforward : 1. the businesses must be situated at los angeles (trivial), 2. the business must have two entries in DietaryRestrictionsAttributes associated respectively with 'vegan' and 'vegetarian'. 3. it has to be opened between 14h and 16h at least one day (this is how we understood it), so the business needs to have an entry in IsOpen associated with an Hour which the opening hour is before or equal to 14h and closing hour after 16h (to compute after midnight we need closing time < opening time).

SQL statement

```
SELECT B.Name, ROUND(AVG(R.stars),2) as average_stars, B.Review_Count
FROM Businesses B, Locations L, Hours H, IsOpen IO, Reviews R
WHERE B.Location_ID = L.Location_ID AND B.Business_ID = IO.Business_ID AND
        B.Business_ID = R.Business_ID AND IO.Hour_ID = H.Hour_ID
        AND LOWER(L.City) = 'los angeles'
        AND (TO_NUMBER(SUBSTR(H.Opening, 0, 2)) < 14 OR SUBSTR(H.Opening,0,4) = '14:00')
        AND (TO_NUMBER(SUBSTR(H.Closing, 0, 2)) >= 16
                OR TO_NUMBER(SUBSTR(H.Closing, 0, 2)) < TO_NUMBER(SUBSTR(H.Opening, 0, 2)) )
        AND B.Business_ID IN (
                SELECT D.Business_ID
                FROM DietaryRestrictionsAttributes D
                WHERE LOWER(D.Dietaryrestrictions) = 'vegan')
        AND B.Business_ID IN (
                SELECT D2.Business_ID
                FROM DietaryRestrictionsAttributes D2
                WHERE LOWER(D2.Dietaryrestrictions) = 'vegetarian')
GROUP BY (B.Business_ID, B.Name, B.Review_Count)
ORDER BY AVG(R.stars) DESC
FETCH FIRST 5 ROWS ONLY;
```

Runtime

346 millisecs

Query result

Observed that 0 rows were fetched. By making a query selecting all businesses that serve "vegan" and "vegetarian" food, we note that only 9 businesses offer both, whereas none of them are in Los Angeles.

Query 17

Descriptions of logic

First we computed the list of businesses that have an entry in the relation DinnerBusinnesses associated with ‘good for dinner’. After that we computed the two averages of R.stars, based on the two separated cross products of (Reviews, Businnesses that are good for dinner, and AmbianceAttributes). The first cross product is conditionned by the businesses associated with an ‘upscale’ ambiance, the second conditionned by ‘divey). Finally we subtracted the two average values.

A positive difference implies a higher average star rating for businesses with “upscale” ambience.

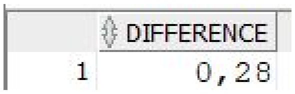
SQL statement

```
WITH DinnerBusinesses AS (
        SELECT G.Business_ID as ID
        FROM GoodForMealAttributes G
        WHERE G.Goodformeal = 'dinner'
)
SELECT ROUND((
        SELECT AVG(R.Stars)
        FROM Reviews R, DinnerBusinesses Dinner, Ambienceattributes A
        WHERE R.Business_ID = Dinner.ID AND A.Business_ID = Dinner.ID
        AND  A.Ambience = 'upscale'
        ) - (
        SELECT AVG(R2.Stars)
        FROM Reviews R2, DinnerBusinesses Dinner2,  Ambienceattributes A2
        WHERE R2.Business_ID = Dinner2.ID AND A2.Business_ID = Dinner2.ID
        AND A2.Ambience = 'divey')
,2) as difference
FROM DUAL;
```

Runtime

18’251 millisecs

Query result



	DIFFERENCE
1	0,28

Query 18

Descriptions of logic

In the most inner selection, we are selecting cities that contain at least five businesses (note : we cannot juste look for cities that have five locations, since we admitted that there is no bijection between a location and a business). In the middle selection, we select the cities from those prementionned cities, that have at least five distinct businesses with more than or equal to 100 reviews. Finally we count the number of those cities. Note of logic : if a city has at least 5 businesses with 100+ reviews, then the top-5 businesses (by number of reviews) of that city will necessarily have 100+ reviews

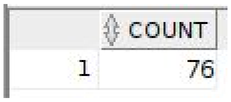
SQL statement

```
SELECT COUNT(*) as count
FROM (SELECT DISTINCT LOWER(L.City)
        FROM Businesses B, Locations L
        WHERE B.Location_ID = L.Location_ID AND B.Review_Count >= 100
                AND LOWER(L.City) IN (
                        SELECT LOWER(L2.City)
                        FROM Locations L2, Businesses B2
                        WHERE L2.Location_ID = B2.Location_ID
                        GROUP by LOWER(L2.City)
                        HAVING COUNT(DISTINCT B2.Business_ID) >=5
                )
        GROUP BY LOWER(L.City)
        HAVING COUNT(DISTINCT(B.Business_ID)) > 5
) subquery
```

Runtime

2541 millisecs

Query result



	COUNT
1	76

Query 19

Descriptions of logic

First, we select for each city the businesses ordered by number of review, along with their rank in that city and name this selection 'Ranks'. This makes the selection of top 100 and not top 100 businesses for a city easier. Secondly, for each city we compute the sum of review counts for the two separation (top 100 and not top 100) of businesses, and select only the cities for which the first sum is at least 2 times the second sum.

Note : that we do LOWER(L.City) only in the Ranks computation, after that the cities are all in lower latter case.

Important note : we do not exclude cities that have less than 100 businesses, because if a city has less than 100 businesses, then all of them are in the top 100 and it automatically satisfies the condition (with an empty non-top-100 list of businesses)-

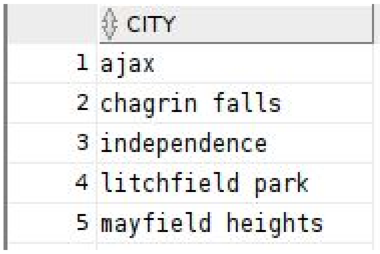
SQL statement

```
WITH Ranks AS (
        SELECT LOWER(L.City) AS City, B.Review_Count AS Review_Count,
                ROW_NUMBER() OVER (PARTITION BY LOWER(L.City)
                                        ORDER BY B.Review_Count DESC) AS Tops
        FROM Locations L, Businesses B
        WHERE B.Location_ID = L.Location_ID
)
SELECT TRIM(Tops100.City) AS City
FROM (
        SELECT R1.City, SUM(R1.Review_Count) AS Rev_Count
        FROM Ranks R1
        WHERE R1.Tops <= 100
        GROUP BY R1.city) Tops100,
        (
        SELECT R2.City, SUM(R2.Review_Count) AS Rev_Count2
        FROM Ranks R2
        WHERE R2.Tops > 100 GROUP BY R2.city) NotTops100
WHERE Tops100.City = NotTops100.City AND Tops100.Rev_Count >= (2 * NotTops100.Rev_Count2)
```

Runtime

2714 millisecs

Query result



	CITY
1	ajax
2	chagrin falls
3	independence
4	litchfield park
5	mayfield heights

Query 20

Descriptions of logic

First in Topten, we select the top-10 businesses by the number of reviews (based on B.Review_Count, the the Reviews relation), after that we work on this selection. After that, in the innermost selection, for each of those top-10 businesses, we list the users that reviewed them, ordered by their review_count, along with their rank in the list. In the outermost selection, we display the top10 businesses with their top-3 users by review-cout (= user with Rank <= 3).

SQL statement

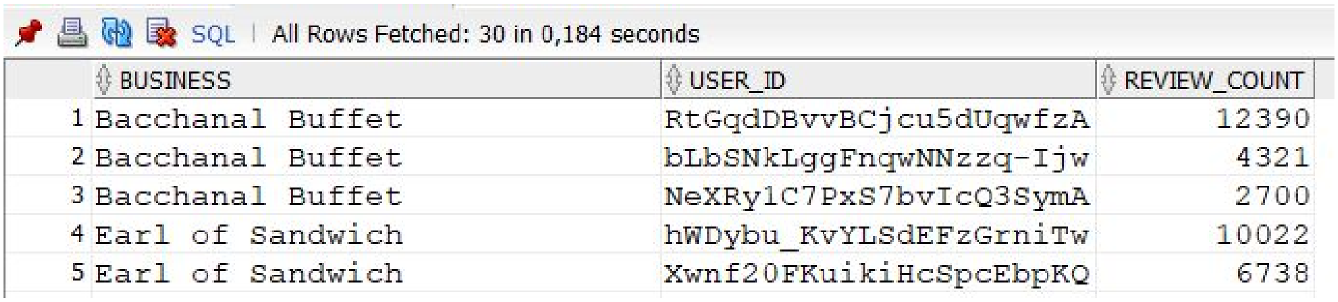
```
WITH Topten AS (
        SELECT Business_ID, Review_Count
        FROM Businesses
        ORDER BY Review_Count DESC
        FETCH FIRST 10 ROWS ONLY
)
SELECT TRIM(B.Name) as Business, Ranks.User_ID, Ranks.Review_Count
```

```
FROM (SELECT T.Business_ID, U.User_ID, U.Review_Count,
                    ROW_NUMBER() OVER (PARTITION BY T.Business_ID
                                                 ORDER BY U.Review_Count DESC) AS Rating
          FROM Users U, Reviews R, Topten T
          WHERE R.Business_ID = T.Business_ID AND U.User_ID = R.User_ID) Ranks,
          Businesses B
WHERE B.Business_ID = Ranks.Business_ID AND Ranks.Rating <= 3
ORDER BY B.Name, Review_Count DESC;
```

Runtime

7931 millisecs

Query result



SQL | All Rows Fetched: 30 in 0,184 seconds

	BUSINESS	USER_ID	REVIEW_COUNT
1	Bacchanal Buffet	RtGqdDBvvBCjcu5dUqwfzA	12390
2	Bacchanal Buffet	bLbSNkLggFnqwNNzzq-Ijw	4321
3	Bacchanal Buffet	NeXRy1C7PxS7bvIcQ3SymA	2700
4	Earl of Sandwich	hWDybu_KvYLSdEFzGrniTw	10022
5	Earl of Sandwich	Xwnf20FKuikiHcSpcEbpKQ	6738

Note that 30 rows are fetched, which corresponds to top 3 users for top 10 businesses.

Query Performance Analysis – Indexing

<In this section, for 6 selected queries explain in detail why do you see given improvements (or not). For example, why building an index on certain field changed the plan and IO.>

Query 1

<Initial Running time/IO:
Optimized Running time/IO:
Explain the improvement:
Initial plan
Improved plan>

Query 2

<Initial Running time/IO:
Optimized Running time/IO:
Explain the improvement:
Initial plan
Improved plan>

DIAS: Data-Intensive Applications and Systems Laboratory
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Building BC, Station 14
CH-1015 Lausanne
URL: http://dias.epfl.ch/

General Comments

<In this section write general comments about your deliverable (comments and work allocation between team members>