

# Databases Project – Spring 2020

---

Team No: 41

Members: Abdrabo, Yassmine Bahaaeldin Gaber Hafez; Di, Yao; Ebrahimi, Mohammadreza

## Contents

<b>Deliverable 1</b>	2
Assumptions	2
Entity Relationship Schema	3
Schema	3
Description	4
Relational Schema	5
ER schema to Relational schema	5
DDL	8
General Comments	14
<b>Deliverable 2</b>	15
Assumptions	15
Data Loading/Cleaning	15
Query Implementation	19
General Comments	24
<b>Deliverable 3</b>	24
<b>Assumptions</b>	25
Query Implementation	25
Query Performance Analysis – Indexing	36

## **Deliverable 1**

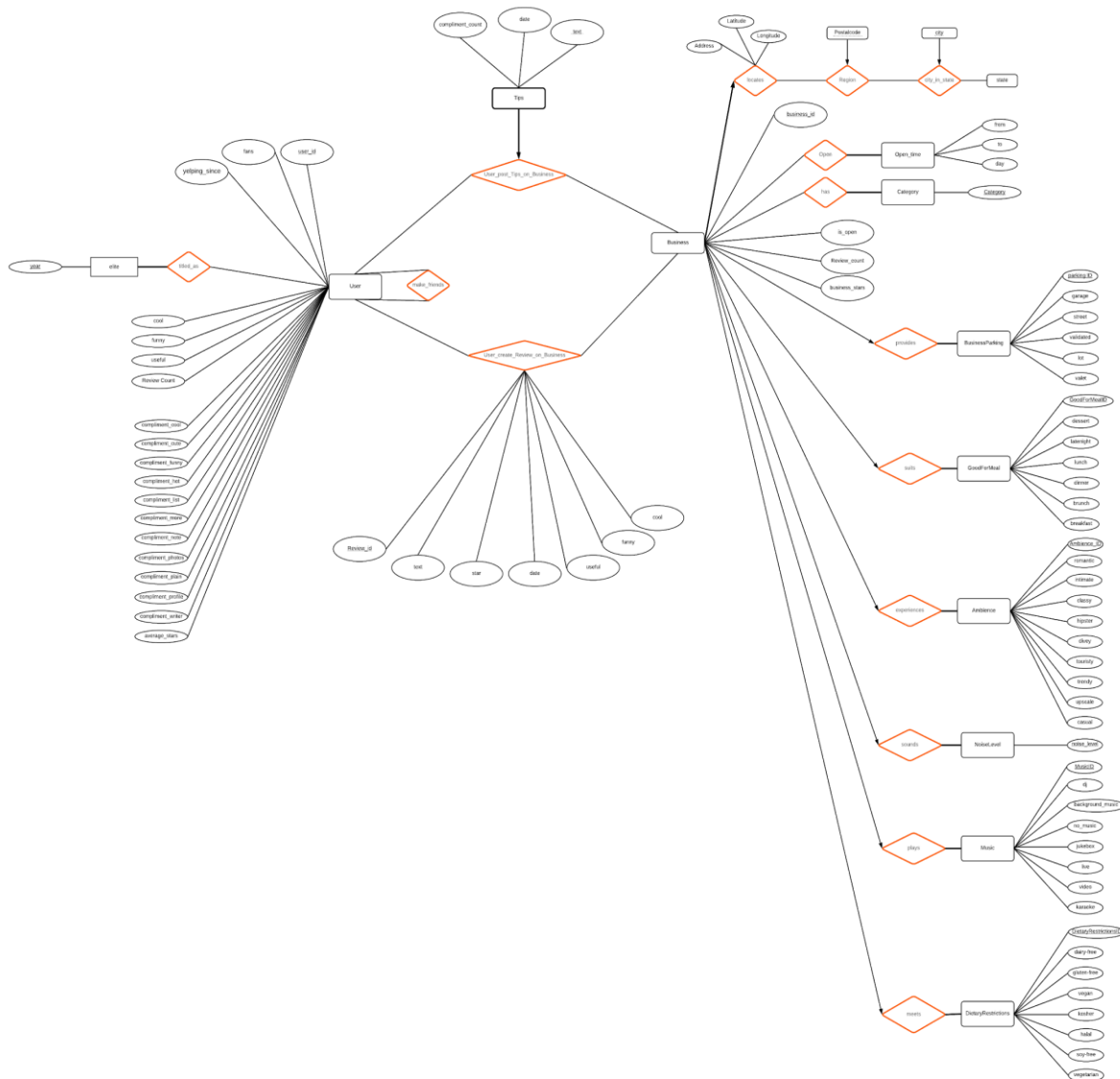
### ***Assumptions***

We have assumed that a user can write only one review for a business, but we have also allowed a user to submit several tips for a business, which is allowed by Yelp. We have assumed that no two businesses can have the exact same address, differing at least in their postal box addresses. We have also assumed that a business may operate at different intervals during a weekday (i.e., a business's operating hours for a day are not necessarily a single interval).

## Entity Relationship Schema

Schema

Copy of Intro2DB ER model Deliverable 1  
 Introduction to Database Systems, 1st Edition, 1997, 1998



## Description

### User-Business relationship:

We made the user-business relationship many-to-one, with attributes of the relationship the fields of a review, for two reasons: 1. A user can write only one review for a business (as assumed), while a business has many users review it. 2. The review is what characterizes a unique user-business relationship, so it is more intuitive in terms of design to have it as an attribute of a user-business relationship (i.e., since a relationship is uniquely characterized by its participating entities, and since a user-business pair already uniquely identifies a relationship instance of a relationship set, there is no need to have the review as a participating entity nor any other participating entity).

### User-Elite relationship

We created an entity for the Elite status for the reason that a user may be an elite user for an unlimited number of years, which may or may not be consecutive. Having an Elite entity with a single attribute of “year” thus simplifies capturing this information, since a user will then simply be in a many-to-many relationship with Elite entities with the appropriate “year” values. In addition, users who will never have been elite users will not need to be included in this relationship set.

### Business-Category relationship and Category entity

We created a separate category entity, since the categories a business can have are almost unlimited, so a separate entity is necessary to capture this information on a business. The business-category relationship is thus many-to-many, since a business can have many categories, and many businesses can have the same category.

### User-Business-Tip relationship, Tip weak entity set

We chose to make tips a weak entity set, since a user can leave several tips for a business, which in turn implies that a user-business pair cannot be used alone to identify a tip. At the same time, a tip has an underlying “structure” given by the text, date, and its complement count, but which cannot be used to uniquely identify a tip for a specific business by a specific user. Therefore, we decided to make a tip a weak entity that can be uniquely identified once given a business and a user.

### Is-open relationship, Open-hours entity set

We chose to create a separate entity for opening hours, since the opening hours of a business exhibit an underlying structure (day of the week and opening intervals). In addition, we assign to the opening\_hours entity attributes for day of the week and from and to hours, since a business can operate at several intervals during a weekday, and not necessarily operate continuously (for example, a business can be open on Monday from 8h-12h, and from 14h-18h).

### set <business attribute> relationships, and <business attribute> entities

We chose to have a separate entity for each type of business attribute (6 in total), since each attribute exhibited a structure, independent of other attribute structures. In addition, we do not enforce the constraint of full participation on businesses in this relationship, to avoid cases where a business does not have any of the sub-

attributes of an attribute (i.e., for the sake of efficiency). By inspecting the CSV file for businesses, we see that many businesses are indeed missing some of the attributes. However, we do enforce a many-to-one relationship on the businesses, since a business should not be allowed to be in a relationship with several entities of the same type of attribute (i.e., one entity of an attribute entity set will exclusively determine the business's sub-attributes for that attribute type).

#### Business entity

We chose to have a business entity in our database, since a business exhibits several qualities which do not change frequently, seem independent of any other object, and are integral to its representation. We chose as the primary key for a business its `business_id`, since it is a unique identifier which all businesses must have. The remaining attributes of our business entity are `review_count`, `business_stars`, and `is_open`. We chose these data fields to be attributes since each of them is a single value which does not change frequently.

#### User entity

We chose to have a user entity in our database, following a reasoning similar to that for a business entity. A user represents an object with qualities independent of any other object and which do not change frequently. We chose as the primary key for a user their `user_id`, since it is unique, and all users must have one. The remaining attributes of a user are: `fans`, `yelping_since`, `cool`, `funny`, `useful`, `review_count`, all types of compliment counts given by `compliment_<type>`, and `average_stars`. These attributes represent simple values which do not change frequently, and which do not exhibit an important underlying structure.

## ***Relational Schema***

### ER schema to Relational schema

Once we have finalized the ER model, we can move on to converting that to a Relational Schema. For that, we have different tables capturing the entity sets, relationship sets and the ICs involved in them. In the following, we provide the explanations of the DDL and address the possible ambiguities.

#### User entity set

For this table, we have several attributes that require some elaboration. As it makes sense, the `user_id` should be the primary key. `name` attribute should participate entirely; otherwise, it doesn't make sense to have that in the dataset, which translates to NOT NULL in the relational schema. Also, all users should have the attribute `yelping_since` for the same reason. This fact also works for the `fans` attribute as well. For the compliments that users receive like `cool` or the compliments they leave, it is allowed to have Null (zero) in this case.

We should have the `average_star` attribute since we have access to a slice of the data, and we can't compute this from other attributes.

#### Elite relationship set

For elite users, we considered a separate table that keeps track of the user's elite status. The reason is that it can be more than one period for which a user is elite. For that, we specified `user_id` and `year` as the primary key,

which allows for many to many relationships but disallows redundant tuples. It makes sense to have the user\_id as the foreign key from the User table.

#### Friends relationship set

Similar to Elite, this relationship is many to many as with the difference that we should have two different user\_ids as the primary key and specify both as a foreign key.

#### Business attribute set

Since we want to have a clean and organized table, it makes sense not to have many attributes specific to this table. Instead, as we will explain later, we decided to have different tables with respective attributes, and they will be related to this entity set by the business\_id key. Intuitively, business\_id is the primary key. Some of the attributes have their separate table due to the importance of future queries (e.g., categories).

We use the UNIQUE command for the latitude, longitude pair since we assume that each business should have a unique location. In the section related to locate, we provide more explanations on how we ensured the one to one relationship between a business and its location. Also, each business must have a name (NOT NULL).

For the category attribute, we will later define a separate table for category relationship and entity, which assigns one label to each business category. Hence, all the possibilities of the category names can be covered, and in the business table, we will have cleaner tuples.

For the opening hours entity, we decided to define the attribute hours\_key, which is unique, and it's a foreign key. This way, we ensure that each business should have opening hours in a separate table. Later on, we describe more details on this.

#### locate relationship set

As we mentioned, we have a separate table for locations to make future queries convenient. The primary key for this table is again, business\_id that makes sure we will have unique business\_ids, and each of them will be associated with exactly one latitude, longitude pair. Utilizing this design combined with the business table, the one to one relationship is obliged. -- Changes were made since the previous deliverable to accommodate the change in the ER diagram, where we made the city a weak entity that depends on state, and postal code a weak entity that depends on city and state. For the Region relationship, we added an extra table and made the primary key (postal\_code, city, state), since postal code is a weak entity that is determined by the combination of city and state. We did not add additional tables for city or postal code, since being weak entities, they depend on a defining relationship. We only added an extra table for states, since they are an independent entity, and represent logically different objects from a region relationship or a city-state relationship.

#### hours key table

To address the problem of opening hours, we created an intermediate table which contains "hours\_key", "day", "from" and "to" attributes. For this table, the primary key is (hours\_key, day, from), and where we use the hours\_key as a link to the business table. Then, the foreign key is defined to be (day, from, to) that corresponds to the same attributes in the set\_business\_hours table. Consequently, each key can be in a relationship with several values of (day, from, to) attributes. This way, we avoid repetitive information.

### set\_business\_hours relationship set

For this table, we chose the primary key to be the (day, from, to). This means that we first generate this table with all the opening period possibilities. Then, by generating hours\_key\_table, each key can be assigned with multiple tuples in the set\_business\_hours table. Finally, each business\_id will be associated with one key that represents all the opening hours period of that business. The benefit of this approach is that if some businesses share the same opening hours (which is usually the case), we can use the same hours\_key for them.

### Business to Business Attributes

In our ER model, there are six <business attributes> entities linked to the entity Business with one-to-many <set\_business\_attribute> relationship. According to the one-to-many key constraint, the <business attributes> should be combined with the corresponding relationship during the ER model translating process.

In the combined table, all the sub-attributes of the <business attributes> should be recorded as well as the foreign key business\_id, which refers to the <business entity>. The primary key should be set as business\_id, which enforces the key constraint that a business cannot associate to more than one entity of a given attribute.

The six combined tables include:

- set\_business\_parking
- set\_good\_for\_meal
- set\_ambiance
- set\_noise\_level
- set\_music
- set\_dietary\_restrictions

### User create review on business

According to our assumption about the user-business relationship, one table should be created for the <user\_create\_review\_on\_business> relation. The table should include all the review related attributes including review\_id, text, stars, date, cool, funny, useful. As it is a table for relation, the keys user\_id and business\_id should be imported as the foreign key.

There are two sets of key in the dataset, one is the review\_id and the other set is the combination of (user\_id, business\_id). In order to make sure that our assumption always holds that each user could only review once the same business, the (user\_id, business\_id) should be defined as the primary key.

### tip

The table <tip> is a combination of the tip entity and its relation with <user> and <business> entities. The combination is the result of the one-to-many constraint.

- All the tip-related attributes should be defined as columns, including text, date and comp\_count.
- The keys user\_id and business\_id should be imported as the foreign key
- As <tip> is a weak entity, the primary key should be a combination of text (for tips), user\_id and business\_id (which allows a user to submit several different tips for the same business)

### Set\_categories table

In this table, we define unique labels with attribute name `label1`. We also include `business_id` information from the `business` table, which is a foreign key. The primary key is, in this case, (`business_id`, `label1`) that provides many to many relationship for `business` and `category`.

#### DDL

```
CREATE TABLE Userr(  
    namee VARCHAR(70) NOT NULL,  
    yelping_since DATE NOT NULL,  
    user_idd VARCHAR(25),  
    fans INTEGER NOT NULL,  
    review_count INTEGER,  
    cool INTEGER,  
    funny INTEGER,  
    useful INTEGER,  
    comp_cool INTEGER,  
    comp_cute INTEGER,  
    com_funny INTEGER,  
    comp_hot INTEGER,  
    comp_list INTEGER,  
    comp_more INTEGER,  
    comp_note INTEGER,  
    comp_photos INTEGER,  
    comp_plain INTEGER,  
    comp_profile INTEGER ,  
    comp_writer INTEGER,  
    average_stars FLOAT,  
    PRIMARY KEY (user_idd)  
);  
  
CREATE TABLE Elite(  
    yearr INTEGER,  
    user_idd VARCHAR(25),  
    PRIMARY KEY (user_idd, yearr),  
    FOREIGN KEY (user_idd) REFERENCES Userr  
);  
  
CREATE TABLE Friends(  
    user_id1 VARCHAR(25),  
    user_id2 VARCHAR(25),  
    PRIMARY KEY (user_id1, user_id2),  
    FOREIGN KEY (user_id1) REFERENCES Userr(user_idd),
```



**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/>

```
FOREIGN KEY (user_id2) REFERENCES Userr(user_idd)
);
```

```
CREATE TABLE Business(
    business_id VARCHAR(25),
    namee VARCHAR(70) NOT NULL ,
    review_count INTEGER,
    business_stars FLOAT,
    openn INTEGER,
    PRIMARY KEY (business_id)
);
```

```
CREATE TABLE Statee(
    statee VARCHAR2(5),
    PRIMARY KEY (statee)
);
```

```
CREATE TABLE City_in_statee(
    city VARCHAR2(50),
    statee VARCHAR2(5),
    PRIMARY KEY (city, statee),
    FOREIGN KEY (statee) REFERENCES Statee
);
```

```
CREATE TABLE Region(
    postal_code VARCHAR(10),
    city VARCHAR(50),
    statee VARCHAR(5),
    PRIMARY KEY (postal_code, city, statee),
    FOREIGN KEY (city, statee) REFERENCES City_in_statee
);
```

```
CREATE TABLE locate(
    address VARCHAR(120),
    city VARCHAR(50),
    statee VARCHAR(5),
    postal_code VARCHAR(10),
    latitude FLOAT,
    longitude FLOAT,
    business_id VARCHAR(25),
```

**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/>

```
        PRIMARY KEY (business_id),
        FOREIGN KEY (postal_code, city, statee) REFERENCES Region
    );
```

```
CREATE TABLE business_hours(
    dayy INTEGER,
    fromm FLOAT,
    too FLOAT,
    business_id VARCHAR(25),
    PRIMARY KEY (business_id, dayy, fromm, too),
    CONSTRAINT CHK_DAY CHECK (dayy>=1 AND dayy<=7),
    FOREIGN KEY (business_id) REFERENCES Business
);
```

```
CREATE TABLE has_business_parking(
    parking_id CHAR(5),
    business_id VARCHAR(25),
    PRIMARY KEY (business_id),
    FOREIGN KEY (business_id) REFERENCES Business,
    FOREIGN KEY (parking_id) REFERENCES Parking
);
```

```
CREATE TABLE Parking(
    parking_id CHAR(5),
    garage INTEGER,
    street INTEGER,
    validated INTEGER,
    lot INTEGER,
    valet INTEGER,
    UNIQUE (garage,street,validated,lot,valet),
    PRIMARY KEY parking_id
);
```

```
CREATE TABLE suit_good_for_meal(
    meal_id CHAR(6),
    business_id VARCHAR(25),
    PRIMARY KEY (business_id),
    FOREIGN KEY (business_id) REFERENCES Business,
    FOREIGN KEY (meal_id) REFERENCES Meal
);
```

**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/>

```
CREATE TABLE Meal(  
    meal_id CHAR(6),  
    dessert INTEGER,  
    latenight INTEGER,  
    lunch INTEGER,  
    dinner INTEGER,  
    brunch INTEGER,  
    breakfast INTEGER,  
    UNIQUE (dessert, latenight, lunch, dinner, brunch, breakfast),  
    PRIMARY KEY(meal_id)  
);
```

```
CREATE TABLE experience_ambiance(  
    ambiance_id CHAR(9),  
    business_id VARCHAR,  
    PRIMARY KEY (business_id),  
    FOREIGN KEY (business_id) REFERENCES Business,  
    FOREIGN KEY (ambiance_id) REFERENCES Ambiance  
);
```

```
CREATE TABLE Ambiance(  
    ambiance_id CHAR(9),  
    romantic INTEGER,  
    intimate INTEGER,  
    classy INTEGER,  
    hipster INTEGER,  
    divey INTEGER,  
    touristy INTEGER,  
    trendy INTEGER,  
    upscale INTEGER,  
    casual INTEGER,  
    UNIQUE (romantic, intimate, classy, hipster, divey, touristy, trendy, upscale, casual),  
    PRIMARY KEY(ambiance_id)  
);
```

```
CREATE TABLE sound_noise_level(  
    noise_level INTEGER,  
    business_id VARCHAR,  
    PRIMARY KEY (business_id),  
    FOREIGN KEY (business_id) REFERENCES Business,  
);
```

**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/>

```
CREATE TABLE play_music(  
    music_id CHAR(7),  
    business_id VARCHAR(25),  
    PRIMARY KEY (business_id),  
    FOREIGN KEY (business_id) REFERENCES Business,  
    FOREIGN KEY (music_id) REFERENCES Music  
);
```

```
CREATE TABLE Music(  
    music_id CHAR(7),  
    dj INTEGER,  
    background_music INTEGER,  
    no_music INTEGER,  
    jukebox INTEGER,  
    live INTEGER,  
    video INTEGER,  
    karaoke INTEGER,  
    UNIQUE (dj, background_music, no_music, jukebox, live, video, karaoke),  
    PRIMARY KEY (music_id)  
);
```

```
CREATE TABLE meet_dietary_restrictions(  
    diet_id CHAR(5),  
    business_id VARCHAR(25),  
    PRIMARY KEY (business_id),  
    FOREIGN KEY (business_id) REFERENCES Business,  
    FOREIGN KEY (diet_id) REFERENCES Diet  
);
```

```
CREATE TABLE Diet(  
    diet_id CHAR(7),  
    dairy_free INTEGER,  
    gluten_free INTEGER,  
    vegan INTEGER,  
    kosher INTEGER,  
    halal INTEGER,  
    soy_free INTEGER,  
    vegetarian INTEGER,  
    UNIQUE (dairy_free, gluten_free, vegan, kosher, halal, soy_free, vegetarian),
```

**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/>

```
        PRIMARY KEY (diet_id)
    );

CREATE TABLE user_create_review_on_business(
    review_id VARCHAR(25),
    review_text VARCHAR(50),
    star INTEGER,
    date DATE,
    cool INTEGER,
    funny INTEGER,
    useful INTEGER,
    business_id VARCHAR(25),
    user_id VARCHAR(25),
    PRIMARY KEY (user_id, business_id),
    FOREIGN KEY (business_id) REFERENCES Business,
    FOREIGN KEY (user_id) REFERENCES User,
    UNIQUE (review_id)
);

CREATE TABLE tip(
    business_id VARCHAR(25),
    user_id VARCHAR(25),
    count INTEGER,
    date DATE,
    review_text VARCHAR(25),
    PRIMARY KEY (business_id, user_id, review_text),
    FOREIGN KEY (user_id) REFERENCES User,
    FOREIGN KEY (business_id) REFERENCES Business
);

CREATE TABLE category_labels(
    label VARCHAR(40),
    PRIMARY KEY (label)
);

CREATE TABLE has_categories(
    label VARCHAR(40),
    business_id VARCHAR(25),
    FOREIGN KEY (business_id) REFERENCES Business,
    FOREIGN KEY (label) REFERENCES category_labels,
    PRIMARY KEY (business_id, label)
```

);

## ***General Comments***

We all contributed to each of the ER model creation, sql commands, and writing the report, with each one of us having contributed with more emphasis to a specific sub-task. We found it particularly difficult to enforce the opening hours relationship in the ER schema, and we realized after having created our finalized ER model that perhaps sql is not well-suited for expressing many-to-many relationships with total participation constraints. We had a similar difficulty with location, where we wanted to find a way to enforce total participation on businesses without having to include the structure of an address in a business entity. In this case, however, it was easier to do, since locations and businesses are one-to-one. On the other hand, we found that it was sometimes easier to imagine how the ER schema should be (i.e., in terms of columns and tables), and then found it hard to infer an ER model that would correspond to such a schema. Finally, we realize that establishing an ER model and an ER schema is far from a trivial task!

## Deliverable 2

### *Assumptions*

We altered some of our assumptions based on the queries that we should have provided and the feedback from the TA that will be described here. First, we determined the maximum number of characters for the attributes of type VARCHAR otherwise we would encounter errors with parsing the data. This issue was addressed with the help of the Python and Pandas package. Also, we lifted the assumption that each business should have opening hours. Accordingly, we managed to get rid of hours\_key\_table in the relational schema. Moreover, we translated the “fromm” and “too” attributes to float numbers due to query requirements.

For the “locate” table, we relaxed the assumption of the UNIQUE (latitude, longitude) pair since we assumed that two businesses can share a building. For each of the business attributes, we generated a new table in which we keep all the different combinations of Booleans and assigned to each combination a key. Of course, many businesses can share this key which makes the data storage to be more minimized (no need to do the same for Noise Level attribute).

For the category attribute, we created a new table with the name “category”. In this table, we keep the list of all categories (which was at the end over 1300) to use in the table “has\_categories” that ensures many to many relationships between categories and businesses.

For all the data with BOOLEAN type, we changed the type to integer. The reason is that SQL server doesn’t have Boolean datatype.

### *Data Loading/Cleaning*

#### **Data parsed from “yelp\_academic\_dataset\_review”:**

The csv file includes the following columns: 'business\_id', 'cool', 'date', 'funny', 'review\_id', 'stars', 'text', 'useful', 'user\_id'.

1. Table **USER\_CREATE\_REVIEW\_ON\_BUSINESS**:
  - a) The table takes all the columns of the csv file.

#### **Data parsed from “yelp\_academic\_dataset\_tip\_transposed.csv”:**

The csv file includes the following columns: 'business\_id', 'compliment\_count', 'date', 'text', 'user\_id'.

2. Table **TIP**:
  - a) The table takes all the columns of the csv file.

### Data parsed from “yelp\_academic\_dataset\_user.csv”:

The csv file includes the following columns: 'average\_stars', 'compliment\_cool', 'compliment\_cute', 'compliment\_funny', 'compliment\_hot', 'compliment\_list', 'compliment\_more', 'compliment\_note', 'compliment\_photos', 'compliment\_plain', 'compliment\_profile', 'compliment\_writer', 'cool', 'elite', 'fans', 'friends', 'funny', 'name', 'review\_count', 'useful', 'user\_id', 'yelping\_since'.

#### 3. Table **USERR**:

- a) The table takes the following columns from the csv file: 'name', 'yelping\_since', 'user\_id', 'fans', 'review\_count', 'cool', 'funny', 'useful', 'compliment\_cool', 'compliment\_cute', 'compliment\_funny', 'compliment\_hot', 'compliment\_list', 'compliment\_more', 'compliment\_note', 'compliment\_photos', 'compliment\_plain', 'compliment\_profile', 'compliment\_writer', 'average\_stars'
- b) The row with no 'name' is removed.

#### 4. Table **ELITE**:

- a) The table takes the following columns from the csv file: 'user\_id', 'elite'.
- b) Empty cell in 'elite' as well as the corresponding cell in 'user\_id' is cleaned.
- c) The strings of years saved in 'elite' are splitted and saved in new cells.
- d) For each new cell containing single elite year, a new cell is created containing the string in corresponding 'user\_id' cell.

#### 5. Table **FRIENDS**:

- a) The table takes the following columns from the csv file: 'user\_id', 'friends'.
- b) Empty cell in 'friends' as well as the corresponding cell in 'user\_id' is cleaned.
- c) The strings of friend user\_ids saved in 'friends' are splitted and saved in new cells.
- d) For each new cell containing single friend user\_id, a new cell is created containing the string in corresponding 'user\_id' cell.

### Data parsed from “yelp\_academic\_dataset\_business.csv”:

The csv file includes the following columns: 'address', 'attributes', 'business\_id', 'categories', 'city', 'hours', 'is\_open', 'latitude', 'longitude', 'name', 'postal\_code', 'review\_count', 'stars', 'state'.

#### 6. Table **BUSINESS**:

- a) The table takes the following columns of the csv file: 'business\_id', 'name', 'review\_count', 'stars', 'is\_open'.

#### 7. Table **LOCATE**:



- a) The table takes the following columns from the csv file: 'address','city','state','postal\_code', 'latitude','longitude','business\_id'
- b) The empty cells in city column are replaced by “ctXXX” where XXX presents the cell is the XXXth cell which contains null value.
- c) The empty cells in postal\_code column are replaced by “pcXXX” where XXX presents the cell is the XXXth cell which contains null value.

8. Table **HAS\_CATEGORIES**:

- a) The table takes the following columns from the csv file: 'business\_id', 'categories'.
- b) Empty cell in 'categories' as well as the corresponding cell in 'business\_id' is cleaned.
- c) The strings of categories saved in 'friends' are splitted and saved in new cells.
- d) For each new cell containing a single category, a new cell is created containing the string in the corresponding 'business\_id' cell.

9. Table **CATEGORIES**:

- a) The table takes the following columns from the csv file: 'categories'.
- b) Empty cells in 'categories' are cleaned.
- c) The strings of category saved in 'categories' are splitted and saved in new cells.
- d) Convert the new array of single category cells to a new non-duplicate array/set

10. Table **Region and City\_in\_State**:

- a.) The Region table takes as columns postal\_code, city, and state
- b.) Postal\_code is the weak identifier for the postal\_code weak entity of Region, which is parsed from the business csv file by simply taking postal code entry and making its letters small and removing excess white space.
- c.) City, state pair is the primary key of city\_in\_State table, and they are parsed by taking these entries from the business csv file, making the letters small, and removing excess white space.

**Process for attributes:**

- A. The process takes the following columns from the csv file: 'business\_id', 'attributes'.
- B. Empty cell in 'attributes' as well as the corresponding cell in 'business\_id' is cleaned.
- C. The strings of attributes saved in 'attributes' are splitted based on attribute types and saved in new cells in new list 'Attributes\_tmp'.
- D. For each new cell in 'Attributes\_tmp' containing a single attribute series, a new cell is created containing the string in the corresponding 'business\_id' cell in 'Business\_ids\_tmp'.
- E. Separate 'Attributes\_tmp' and 'Business\_ids\_tmp' based on the attribute group and regroup into 6 attributes sets.

10. Table **PARKING**:

- a) The table takes the 'bp\_attributes' columns generate in E for the business\_parking group
- b) Create new items columns: 'garage', 'street', 'validated', 'lot', 'valet'

- c) Fill the cell according to the string in 10.a). 'T' for TRUE, 'F' for False, 'C' if the value is unknown
- d) Create new column: 'parking\_id'
- e) Fill the combination of values in c) to 'parking\_id' e.g. 'TFTFF'

**11. Table HAS\_BUSINESS\_PARKING:**

- a) The table takes the 'bp\_attributes' and 'bp\_bid' columns generate in **E** for the business\_parking group,
- b) For each new cell in 'bp\_bid' containing a single business\_id, the value in 'bp\_attributes' is replaced by the corresponding 'parking\_id'.

**12. Table MEAL:**

- a) The table takes the 'meal\_attributes' columns generate in **E** for the good\_for\_meal group
- b) Create new items columns: 'dessert', 'latenight', 'lunch', 'dinner', 'brunch', 'breakfast'
- c) Fill the cell according to the string in 12.a). 'T' for TRUE, 'F' for False, 'C' if the value is missing
- d) Create new column: 'meal\_id'
- e) Fill the combination of values in c) to 'meal\_id' e.g. 'TFTFF'

**13. Table SUIT\_GOOD\_FOR\_MEAL:**

- a) The table takes the 'meal\_attributes' and 'meal\_bid' columns generate in **E** for the good\_for\_meal group,
- b) For each new cell in 'meal\_bid' containing a single business\_id, the value in 'meal\_attributes' is replaced by the corresponding 'meal\_id'.

**14. Table AMBIANCE:**

- a) The table takes the 'ambiance\_attributes' columns generate in **E** for the ambiance group
- b) Create new items columns: 'romantic', 'intimate', 'classy', 'hipster', 'divey', 'touristy', 'trendy', 'upscale', 'casual'.
- c) Fill the cell according to the string in 15.a). 'T' for TRUE, 'F' for False, 'C' if the value is unknown
- d) Create new column: 'ambiance\_id'
- e) Fill the combination of values in c) to 'ambiance\_id' e.g. 'TFTFF'

**15. Table EXPERIENCE\_AMBIANCE:**

- a) The table takes the 'ambiance\_attributes' and 'ambiance\_bid' columns generate in **E** for the ambiance group,
- b) For each new cell in 'ambiance\_bid' containing a single business\_id, the value in 'ambiance\_attributes' is replaced by the corresponding 'ambiance\_id'.

**16. Table SOUND\_NOISE\_LEVEL:**

- a) The table takes the 'noise\_attributes' and 'noise\_bid' columns generate in **E** for the sound\_noise\_level group

**17. Table MUSIC:**

- a) The table takes the 'music\_attributes' columns generate in **E** for the music group

- b) Create new items columns: 'dj', 'background\_music', 'no\_music', 'jukebox', 'live', 'video', 'karaoke'.
- c) Fill the cell according to the string in 17.a). 'T' for TRUE, 'F' for False, 'C' if the value is unknown
- d) Create new column: 'music\_id'
- e) Fill the combination of values in c) to 'music\_id' e.g. 'TFTFF'

18. Table **PLAY\_MUSIC**:

- a) The table takes the 'music\_attributes' and 'music\_bid' columns generate in **E** for the music group,
- b) For each new cell in 'music\_bid' containing a single business\_id, the value in 'music\_attributes' is replaced by the corresponding 'music\_id'.

19. Table **DIET**:

- a) The table takes the 'diet\_attributes' columns generate in **E** for the diet group
- b) Create new items columns: 'dairy\_free', 'gluten\_free', 'vegan', 'kosher', 'halal', 'soy\_free', 'vegetarian'.
- c) Fill the cell according to the string in 18.a). 'T' for TRUE, 'F' for False, 'C' if the value is unknown
- d) Create new column: 'diet\_id'
- e) Fill the combination of values in c) to 'diet\_id' e.g. 'TFTTTFF'

20. Table **MEET\_DIETARY\_RESTRICTIONS**:

- a) The table takes the 'diet\_attributes' and 'diet\_bid' columns generate in **E** for the diet group,
- b) For each new cell in 'diet\_bid' containing a single business\_id, the value in 'diet\_attributes' is replaced by the corresponding 'diet\_id'.

## ***Query Implementation***

<For each query>

**Query:**

**Description of logic:**

1. For this query, we simply should calculate the overall average of the number of reviews per user which is quite straightforward.
2. This query requires to find the number of businesses located in Québec or Alberta. We simply need to use the COUNT command and apply WHERE with the combination of OR to check for both states.
3. For this query, we first create a new table H2 which is has\_categories table that is grouped by business\_id with another attribute nc (number of categories) which counts how many times that business id has been repeated and sort H2 in descending order based on that. At the end, we compare the business\_ids between H2 and Business table and take only the first row.

4. This query is also convenient to implement. We should count the number of tuples that are Dry Cleaners or Dry Cleaning.
5. For this query, we should combine tables "Business", "meet\_dietary\_restrictions" and "Diet" to access review count and the number of TRUE values in Diet table that should be more than one. Since we defined diet\_id in a meaningful way, we can simply count number of 'T' in the diet id to find businesses that have at least 2 dietary restrictions.
6. For this query, we create a new table F in which we have the first attribute the user\_id and the second attribute, number of times that user\_id has appeared in the friends table. This table is grouped by user\_id and ordered by 2<sup>nd</sup> attribute in descending order.
7. We should combine "locate" and "Business" tables to access business names, business\_stars, review counts, and the city they are located in. The location and open attributes are simply checked in WHERE clause and as before, the business table is sorted based on the review count with ORDER BY command. In the end, we pick the first 5 tuples of the most recent relation. However, we found only one tuple verifying the conditions of this query.
8. The idea of this query is similar to that of query 3. We need to access the "state" attribute of the "locate" table and COUNT the repetition of each state and pick the maximum value.
9. For this query, we need to generate a new relation that contains the starting year of the elite user and the user id. The first time that a specific user became elite would be the minimum of the "yearr" attribute after we grouped the elite table based on user\_id. Then, we combine this new table with user table and group it by the starting year. Finally, for each group, we calculate the average and show the respected year.
10. In the beginning, we group the "user\_create\_review\_on\_business" based on business\_id which lists each business with all the review stars they have received. For each group, we sort the number of stars in descending order and take the middle one with the aid of PERCENTILE\_DISC(0.5) command and keep only this value. Finally, we sort this table based on the median\_stars and take the first 10 tuples. Verifying, the "openn" and the state of the business is easy to handle. However, there is only one business open New York
11. For this query, we generate a new table in which we keep the business\_id and the number of times they are repeated in the has\_category table. For finding the minimum, maximum, and average of categories per business, we simply use the respected commands with the count of each business id to be the argument. To find the median, we first sort this table based on the count of each business id and use PERCENTILE\_DISC(0.5) as before.
12. This query is only long because of the number of conditions that should be checked in WHERE. We should combine locate, Business, Parking, has\_business\_parking, and opening\_hours tables to access all the necessary attributes. Between Parking and has\_business\_parking tables the parking\_id should be checked and for the rest of the tables, we should match the business\_id. Then the requirements of the query are checked with AND in between. One interesting remark is that we translated opening hours to float numbers so that we can check the opening time to be before 19:00 and closing time to be after 23:00 and from DDL1, we already changed the day of the weeks to integers for easier query stage.

### SQL statements and Results

```
/* 1 */
SELECT AVG(U.review_count)
FROM User U
```

	AVG(U.REVIEW_COUNT)
1	37.65253451486547229178706736017466127272

```
/* 2 */
SELECT COUNT(L.business_id)
FROM locate L
WHERE L.state = 'QC' OR L.state = 'AB';
```

	COUNT(L.BUSINESS_ID)
1	17231

```
/* 3 */
SELECT B.business_id, B.name, H2.nc
FROM Business B, (SELECT H.business_id, count(H.business_id) as nc
                  FROM has_categories H
                  GROUP BY H.business_id
                  order by count(H.business_id) DESC) H2
WHERE B.business_id = H2.business_id AND ROWNUM = 1;
```

	BUSINESS_ID	NAME	NC
1	135486	Best Of The Best DJ's	37

```
/* 4 */
SELECT COUNT(DISTINCT H.business_id)
FROM has_categories H
WHERE H.label = 'DryCleaners' OR H.label = 'DryCleaning';
```

	COUNT(H.BUSINESS_ID)
1	436

```
/* 5 */
SELECT B.business_id, B.review_count
FROM Business B, meet_dietary_restrictions M, Diet D
```

```
WHERE B.review_count > 150 AND B.business_id = M.business_id AND M.diet_id = D.diet_id
      AND REGEXP_COUNT (M.diet_id, 'T') >= 2;
```

	BUSINESS_ID	REVIEW_COUNT
1	ghaj1MzZPW1qXYCZWfoTTw	195
2	SfhV8mF40RcsGrJjkuEvvw	258
3	ubD06BYjnPSIZIpWxXGd1Q	212
4	KskYqH1Bi7Z_6lpH60m8pg	3998

```
/* 6 */
```

```
SELECT F.user_id1, F.fr_cnt
FROM (SELECT F2.user_id1, COUNT(F2.user_id1) as fr_cnt
      FROM Friends F2
      GROUP BY F2.user_id1
      ORDER BY COUNT(F2.user_id1) DESC) F
WHERE ROWNUM <= 10;
```

	USER_ID1	FR_CNT
1	8DEyKvYp1n0cSFx39vatbg	4919
2	0ilqbcz2m2SnwUeZtGYcnQ	4603
3	ZIOCmdFaMIF56FR-nWr_2A	4597
4	yLW80rR8Ns4X1oXJmkKYgg	4437
5	YttDg0C9A1M4HcA1DsbB2A	4222
6	djxniI8Ux8ZYQJhi0QkrPhA	4211
7	qVc80DYU5SZjKXVBgXdI7w	4134
8	iLjMdZi0Tm7DQxX1C1_2dg	4067
9	F_5_UNX-wrAFCXuAkBZPDw	3943
10	MeDuKsZcnI3IU2g701V-hQ	3923

```
/* 7 */
```

```
SELECT B.nameee, B.business_stars, B.review_count
FROM locate L, Business B
WHERE B.business_id = L.business_id AND L.city = 'San Diego' AND B.openn = 1
ORDER BY B.review_count DESC;
```

	NAMEE	BUSINESS_STARS	REVIEW_COUNT
1	Brooks Photography	1.5	35

```
/* 8 */
```

```
SELECT L.statee, L.cnt
FROM (SELECT L2.statee, COUNT(L2.statee) as cnt
      FROM locate_2 L2
      GROUP BY L2.statee
      ORDER BY COUNT(L2.statee) DESC) L
WHERE ROWNUM = 1;
```

**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/>

	STATEE	CNT
1	az	53083

/\* 9 \*/

```

SELECT AVG(U.average_stars) AS total_average, E.start_year
FROM User U, (SELECT MIN(E1.yearr) as start_year, E1.user_idd
              FROM Elite E1
              GROUP BY E1.user_idd) E
WHERE U.user_idd = E.user_idd
GROUP BY E.start_year
ORDER BY E.start_year;

```

	TOTAL_AVERAGE	START_YEAR
1	3.86261168384879725308267636951687891652	2014
2	3.77914911720910444979791533716230589236	2011
3	3.74081834010446895264074289030760301799	2009
4	3.77247448979591837110969387755102040816	2007
5	3.75761206193969030558272208638956805216	2010
6	3.76123879810226674011597258829731154454	2008
7	3.80750841750841751279461279461279461279	2006
8	3.97701380897583429541618718833908707326	2017
9	3.78939248251748252068764568764568764569	2012

/\* 10 \*/

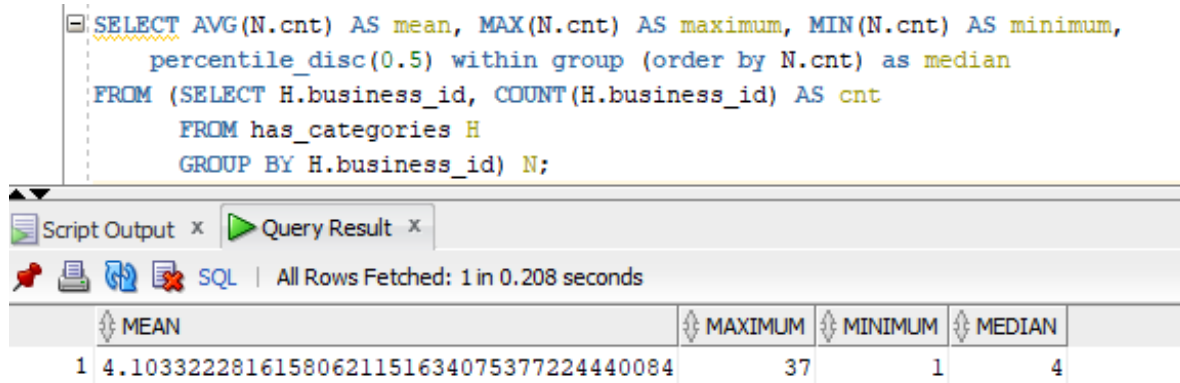
```

SELECT TOP 5 B.namee, (PERCENTILE_DISC(0.5) /* median review of each business */
                     WITHIN GROUP (ORDER BY R.starr) /* order by start rating */
                     OVER (PARTITION BY R.business_id)) AS median_star
FROM Business B, user_create_review_on_business R, locate L
WHERE B.openn = 1 AND L.city = "New York" AND B.business_id = R.business_id
      AND B.business_id = L.business_id /* conditions */
ORDER BY median_star /* final table should be ordered by median_star */

```

	NAMEE	MEDIAN_STAR
1	Safari Beauty Studio	1

/\* 11 \*/



```
SELECT AVG(N.cnt) AS mean, MAX(N.cnt) AS maximum, MIN(N.cnt) AS minimum,
percentile_disc(0.5) within group (order by N.cnt) as median
FROM (SELECT H.business_id, COUNT(H.business_id) AS cnt
FROM has_categories H
GROUP BY H.business_id) N;
```

Script Output x Query Result x

SQL | All Rows Fetched: 1 in 0.208 seconds

	MEAN	MAXIMUM	MINIMUM	MEDIAN
1	4.10332228161580621151634075377224440084	37	1	4

```
/* 12 */
```

```
SELECT B.name AS Name, B.business_stars AS Stars, B.review_count AS Reviews
FROM Business B, Parking P, has_business_parking H, business_hours T, locate_2 L
WHERE B.business_id = H.business_id AND T.business_id = B.business_id
AND L.business_id = B.business_id AND H.parking_id = P.parking_id
AND P.valet = 1 AND L.city = 'las vegas' AND T.day = 5 /* Friday */
AND T.fromm <= 19 AND (T.too >= 23 OR T.too = 0);
```

	NAME	STARS	REVEIEWS
1	The Shops at Crystals	3.5	346
2	Hachi	4	150
3	Ogden's Hops and Harvest	3.5	70
4	The Cosmopolitan of Las Vegas	4	4322
5	The Art of Shaving	4	37
6	Ca'd'Oro	3.5	3
7	Kalifano	4.5	12
8	Cantina Laredo	3.5	385
9	Tropical Smoothie Cafe	2.5	17
10	Tacos N' Ritas	3.5	176

## General Comments

In deliverable 2, except from the cases that we needed the median star of the business, we assumed that the query is asking the `business_star` and `review_count` attributes of the table `Business` rather than asking us to compute the mentioned attributes from the review table.



## Deliverable 3

### *Assumptions*

For the proceeding queries, we have some assumptions that we describe here. For every query that involves projection on the attributes, “business\_star” and “review\_count,” we directly generate the results from the Business table, and we didn’t assume that we are supposed to go through all the tuples in the review table, to calculate these attributes. Of course, for the queries that involve median\_star, we should go through the Review table, to access this quantity. Also, for the case of query 11, we realized that if we were to filter out the businesses that have less than two reviews, we wouldn’t filter out any business, and the query would be less interesting. Therefore, for this query, we count the number of repetitions of each business in the Review table.

### *Query Implementation*

<For each query>

#### **Description of logic:**

1. In this query, we use locate table to restrict the location state, use review\_count and business\_stars attributes in the business table to restrict the corresponding parameters. The two tables are linked with business id. In the end, we count the items under restriction.
2. The dinner restriction is done with the **business-suit\_good\_for\_meal-meal** link and it served as the common restriction. Two sub tables are generated based on the link to the noise level. The average rate is calculated in the sub tables based on the attribute in **business**. The two average rates are extracted from two subtables and the difference is calculated.
3. The live music restriction is done with the **business-play\_music-music** link and the Irish Pub criteria is achieved with the has\_categories table.
4. The definition of elite user in this query is that this user has been entitled as elite in one of the previous years. The table is separated to 4 sub tables according to the elite entitle and average\_stars received. Four separated average values are calculated from the useful attribute in the corresponding table.
5. For this query, we create table C1 that contains business\_id and the repetition in the category table as the 2nd attribute. Then, this count should be greater and equal to two, and one more critical WHERE condition is that in the parking\_id, we have one or 'T' characters with the function REGEXP\_COUNT. In the end, we find the corresponding business ids in the Business table and compute their average business stars and review\_count.
6. First, we create table B1 in which we have the repetition of each business\_id, and the second table is being generated with the same idea with the difference that we check if the business is good for late-night meals. Then we divide the second count by the first one to compute the ratio.
7. In this query, L2 table checks locate and business\_hours tables and find businesses for which the day is not equal to 7, i.e., are closed on Sundays. Then, we group by city name and show them.
8. For this query, we use the review table and group it by business\_id, and then, with the HAVING statement, we check if the count of the DISTINCT user ID is more than 1030. The result is only two businesses.

9. We create a new table in which we check the state to be 'ca', and order it by the business\_star attribute given in the business table. From this table, we only take the first ten tuples.
10. For this query, we use the command ROW\_NUMBER() OVER, PARTITION BY, ORDER BY. This structure is appropriate to find top attributes in grouped tables. With ROW\_NUMBER(), we assign each member of a partition to later use the ranks less or equal to ten. We partition by state and order by business\_star attribute from the Business table. In the new table T, we have business\_id, name of the state, business\_stars, and the business rank in that state named as b\_rank. Then in the WHERE clause, we use b\_rank <= 10.
11. In this query, we have a table r2 that contains business\_id and its count in the review table. For this question, we decided to use the review table since we would filter out no data if we were to use the review\_count attribute of the business table. We find all the businesses with less than two repetitions in the review table and are associated with the city. We use NOT IN to see all the cities for which no business in them has less than two review counts.
12. In this query, we have two critical tables. The first table, R, is a table that contains all the tips that include the 'awesome' string in their tip text with INSTR function. For this table, we keep all the business\_ids. Table P is the second table that inner-joins table tip, keeps tuples with the same user id and dates more difference in date between 0 and 1. Then, we filter out the ones that have the same text. This way, in table P, we have only business ids for which the same user has reviewed another business within the past 24 hours some business (could be the same business as well). Finally, we check the business\_ids from table R and P and count the number of DISTINCT business\_ids.
13. For query 13, we have a table that contains user\_id and the repetitions of DISTINCT business ids with name cnt and ordered by cnt, In case that a user has reviewed a business multiple times. From this table, we only take the first row.
14. For this query, we create two new tables from review and elite tables with names elite\_avg and non\_elite\_avg. For Elite\_avg, we isolate users from review and elite tables with the same id and then calculate the 'useful' attribute in the review table for all of them. For the second table, we choose all users in the review table except the first table with MINUS command and again calculate the average useful rating they gave. In the end, we subtract these two values and name it AVERAGE\_DIFF.
15. To find median stars, we use PERCENTILE\_DISC(0.5) WITHIN group that is ordered by the number of stars that is given to the business in the review table, and partitioned by the business\_id and name it median\_star. What is left to do is to verify business\_ids in the tables good\_for\_meal, business\_hours, and Meal and check if median\_star is >= 4.5, brunch in Meal table is 1. Finally, we should check that for this given business\_id, whether we can find day=7 or day=6, which means that they are open on weekends. No business could be found if we condition it to be open both on Saturday and Sunday.
16. We could only get an empty set based on our logic. The logic in this query is to generate the restricted set with the given location and diet restrictions, group the data in the restricted set by the name of business to calculate the average star and the review counts.
17. In this query, two sub tables are generated based on the ambience restriction. The ambience restriction is done with the **user\_create\_review\_on\_business-experience\_ambience-ambience link**. In each group, the star of the business is calculated by averaging all the review stars linked to the

business. The value is averaged again at the sub group level and the result is the difference between the two calculated values.

18. link and it served as the common restriction. Two sub tables are generated based on the link to the noise level. The average rate is calculated in the sub tables based on the attribute in **business**. The two average rates are extracted from two subtables and the difference is calculated.
19. The logic in this query is: if the city has more than 5 businesses and each top 5 business has more than 100 reviews, the fifth most reviewed business in this city must have more than 100 reviews. The ranking is done with the ROW\_NUMBER() function with partition of the city. The result is calculated by counting the cities matching the criterias.
20. For this query, we first do a simple rewrite of the condition that the sum of the top 100 is at least twice the rest:  $\text{sum\_top\_hundred} > 2 * \text{sum\_rest}$ , and using  $(\text{sum\_rest} = \text{sum\_all} - \text{sum\_top\_hundred})$  implies  $3 * \text{sum\_top\_hundred} > 2 * \text{sum\_all}$ . We do this simple rewrite since we find it easier to compute the sum of all the review counts of a city, rather than the sum of the remaining businesses. For the query itself, we first obtain a ranking of businesses within each city of a state to obtain its top 100 businesses, using oracle sql "partition by" function to partition over cities in their states and the "row\_number()" function to obtain the top 100 of each city in a state. We then use group by (city, state) to calculate the sum of the top 100 businesses of each (city, state). We join this result with a similar one, except we compute the sum of all businesses, and then use a condition to keep those results with  $3 * \text{sum\_top\_hundred} > 2 * \text{sum\_all}$ .
21. For this query, we use the business review count computed from the review table, and the user review count computed from the review table. We compute the review count for businesses separately and remove businesses with a rank less than 100 (using rownum to take exactly 100 businesses, if there are businesses with the same ranking). We then compute user review counts, and join with the previous result. We then use the "partition by" sql function to rank the users for a business based on their review count, and take only the top 3 users for each business.

#### **SQL statement**

*/\* 1 \*/*

```
SELECT COUNT(B.BUSINESS_ID)
FROM BUSINESS B, Locate_2 L
WHERE b.review_count > 5 AND b.business_stars > 4.2
      AND l.business_id = b.business_id AND l.statee = 'on';
```

	COUNT(B.BUSINESS_ID)
1	2891

*/\* 2 \*/*

```
SELECT b_quiet.avg_rate - b_loud.avg_rate
FROM (SELECT AVG(b1.business_stars) as Avg_rate
```

**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/>

```
FROM business b1, suit_good_for_meal s1, meal m1, sound_noise_level n1
WHERE b1.business_id = s1.business_id AND s1.meal_id = m1.meal_id
AND m1.dinner = 1 AND b1.business_id = n1.business_id
AND (n1.noise_level= 0 or n1.noise_level=1)) b_quiet,
(SELECT AVG(b2.business_stars) as Avg_rate
FROM business b2, suit_good_for_meal s2, meal m2, sound_noise_level n2
WHERE b2.business_id = s2.business_id AND s2.meal_id = m2.meal_id
AND m2.dinner = 1 AND b2.business_id = n2.business_id
AND (n2.noise_level= 2 or n2.noise_level=3)) b_loud;
```

	B_QUIET.AVG_RATE-B_LOUD.AVG_RATE
1	0.30845778345692861510452344938876654514

```
/* 3 */
```

```
SELECT b1.name AS name, b1.business_stars, b1.review_count
FROM business b1, play_music p1, music m1, has_categories h1
WHERE b1.business_id = p1.business_id AND p1.music_id = m1.music_id
AND m1.live = 1 AND b1.business_id = h1.business_id
AND h1.label = 'IrishPub'
ORDER BY name;
```

**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/>

NAME	BUSINESS_STARS	REVIEW_COUNT
Blarney's Gate	4	4
Dubliner	3.5	130
Fibber Magees	3.5	246
Fionn MacCool's Irish Pub	3	16
Grumpy's Bar	4	29
Hennessey's Tavern	3	384
Hurley's Irish Pub	4	108
Irish Wolfhound	4	359
J. Dublin's	3.5	19
McFadden's Restaurant and Saloon	2	448
McMullan's Irish Pub	4.5	519
McKibbin's Irish Pub	4.5	8
Mountain View Pub	2.5	115
Murphy's Law	3.5	156
Murphy's Tap Room	3.5	13
O'Connor's Pub	4	34
Public House	3.5	26
Rosie McCaffrey's Irish Pub & Restaurant	3.5	326
R... R...	3.5	298
The Harp And Crown Pub And Kitchen	4	123
The Skeptical Chymist	3.5	380
Tim Finnegan's Irish Restaurant and Pub	4.5	110
Tyber Creek Pub	4	153
Whelan's Gate Gastropub	4.5	24
Whelan's Gate Irish Pub	3	27
Ye Olde Orchard	4	62

```
/* 4 */
```

```
SELECT AVG(g1.useful) AS low_mark_elite
FROM (SELECT DISTINCT u1.user_idd, u1.useful
      FROM userr u1, elite e1
      WHERE u1.average_stars >= 2 AND u1.average_stars < 4 AND u1.user_idd in e1.user_idd) g1;
```

```
SELECT AVG(g2.useful) AS high_mark_elite
FROM (SELECT DISTINCT u2.user_idd, u2.useful
      FROM userr u2, elite e2
      WHERE u2.average_stars >= 4 AND u2.average_stars <= 5 AND u2.user_idd in e2.user_idd) g2;
```

```
SELECT AVG(g3.useful) AS low_mark_non_elite
FROM (SELECT DISTINCT u3.user_idd, u3.useful
      FROM userr u3, elite e3
```

WHERE u3.average\_stars >= 2 AND u3.average\_stars < 4 AND u3.user\_idd not in e3.user\_idd) g3;

```
SELECT AVG(g4.useful) AS high_mark_non_elite
FROM (SELECT DISTINCT u4.user_idd, u4.useful
      FROM userr u4, elite e4
      WHERE u4.average_stars >= 4 AND u4.average_stars <= 5 AND u4.user_idd not in e4.user_idd) g4;
```

LOW_MARK_ELITE	HIGH_MARK_ELITE
694.435303990636249199443475188268810318	516.555225888124060447820238942954347654
LOW_MARK_NON_ELITE	HIGH_MARK_NON_ELITE
118.854266935048578625404821878373515653	47.74405773169686095806671853416109620362

```
/* 5 */
SELECT AVG(B.review_count), AVG(B.business_stars)
FROM Business B, has_business_parking H, Parking P, (SELECT COUNT(c.business_id) AS cat_count, c.business_id
                                                    FROM has_categories C
                                                    GROUP BY c.business_id) C1
WHERE C1.cat_count >= 2 AND B.business_id = H.business_id AND H.parking_id = P.parking_id AND
      B.business_id = C1.business_id AND REGEXP_COUNT (P.parking_id, 'T') >= 1;
```

	AVG(B.REVIEW_COUNT)	AVG(B.BUSINESS_STARS)
1	72.90676883780332056194125159642401021711	3.70961456593640501686478697972950846514

```
/* 6 */
SELECT B2.cnt/B1.cnt as Ratio
FROM (SELECT COUNT(B.business_id) as cnt
      FROM Business B) B1, (SELECT COUNT(B.business_id) as cnt
                             FROM suit_good_for_meal S, Meal M, Business B
                             WHERE B.business_id = S.business_id AND M.meal_id = S.meal_id AND
                             M.latenight = 1) B2;
```

	RATIO
1	0.0103318121167754362464890010331812116775

```
/* 7 */
SELECT L2.city
```

**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/>

```

FROM (SELECT L.city
      FROM locate_3 L, Business_hours H
      WHERE H.day <> 7 AND L.business_id = H.business_id) L2
GROUP BY L2.city
ORDER BY L2.city;

```

	CITY
1	110 las vegas
2	agincourt
3	ahwahtukee
4	ahwatukee
5	airdrie
6	ajax

```

/* 8 */
SELECT U.business_id, COUNT(DISTINCT U.user_id) AS unique_ids
FROM user_create_review_on_business U
GROUP BY U.Business_id
HAVING COUNT(DISTINCT U.user_id) > 1030;

```

	BUSINESS_ID	UNIQUE_IDS
1	4JNXUY8wbaaDmk3BPz1Ww	1251
2	PESDUcs7fIihip38-d6_6g	1311

```

/* 9 */
SELECT *
FROM (SELECT B.business_id, B.namee, B.business_stars
      FROM Business B, locate_3 L
      WHERE L.statee = 'ca' AND L.business_id = B.business_id
      ORDER BY B.business_stars DESC)
WHERE ROWNUM <= 10;

```

	BUSINESS_ID	NAMEE	BUSINESS_STARS
1	d8QM8eV7dyjBTTPever-PA	Jaclyn Webb Healing	5
2	x_vRaIuDsvvRK5tQpV9hoA	Pretty Girl Lingo	5
3	UE_fvET7Aj16w7Cluj7ZGg	Beachside Tans	5
4	meK9y7zRVoN4uaf6sAC8KQ	Finest-Edge Precision Sharpening Service	5
5	UfUddig0a3jD1Irz3Jg-JQ	Fireplace Door Guy	5
6	FSeU1xc0m0zfNU2-IJnxWA	Melody Events	5
7	HUA_z4eTS7hNVqCh00FccQ	Respclearance	5
8	ZbZCTHxQBv0QGLEcL_BYEw	Goldeen Myofascial Release Therapy	5
9	f4SdMDdsCspFFom9J-pqg	Core Pest Solutions	4.5
10	gav10UJkI0ZSDzs_tHXQ9A	Rebecca Vinacour Photography	4.5

```

/* 10 */
SELECT *
FROM (SELECT B.business_id, L.statee, B.business_stars,
      ROW_NUMBER() OVER(PARTITION BY L.statee ORDER BY B.business_stars DESC) AS b_rank
      FROM locate_3 L, Business B
      WHERE L.business_id = B.business_id) T
WHERE T.b_rank <= 10;

```

52	LMwmAg_HZWDlynnIkGpNvQ il	5	2
53	X00bLb5dQcuaBvX9uMt08g il	5	3
54	TyfaHoggE2Z4S-_YgRx1pg il	5	4
55	pFvy0Yi105umrbzGVYSYKg il	5	5
56	i6HkbGtxKPyREtrrcqU4QA il	5	6
57	laAcAVEYFJZ05scr2PbDB9w il	5	7
58	egsTi5bjj7AzK0om_Vu-Eg il	5	8
59	Kq63eBzwMfZelntro8ASTw il	5	9
60	r6Jw8oRCeumxu7Y1WRxT7A il	5	10
61	Xd9Ajs93zQLsiBP5igjsDQ nc	5	1
62	PhA_nQr0VrksJ1Ng11NBMA nc	5	2
63	1bXYF9P0K3rbbERYCm_bdQ nc	5	3
64	JowN0taduZgemso-Zα zFw nc	5	4

```

/* 11 */
SELECT DISTINCT l2.city
FROM locate l2
WHERE l2.city not in
  (SELECT DISTINCT l1.city
   FROM locate l1, (SELECT count(r1.business_id) as cnt, r1.business_id
                     FROM user_create_review_on_business r1
                     GROUP BY r1.business_id) r2
   WHERE l1.business_id = r2.business_id AND r2.cnt<2);

```

1	marvin
2	perry twp
3	la salle
4	mesa az
5	mascouche

```

/* 12 */
SELECT COUNT(DISTINCT(R.idd))
FROM (SELECT T.business_id as idd
      FROM tip T
      WHERE INSTR(T.tip_text, 'awesome') > 0) R,

```



```
(SELECT T1.business_id as idd
FROM TIP T1, TIP T2
WHERE T1.user_idd = T2.user_idd AND (T1.datee - T2.datee) <= 1 AND
(T1.datee - T2.datee) > 0 AND T1.tip_text <> T2.tip_text) P
WHERE R.idd = P.idd;
```

	COUNT(DISTINCT(R.IDD))
1	9598

```
/* 13 */
```

```
SELECT *
FROM (SELECT U.user_idd, COUNT(DISTINCT(U.business_id)) as cnt
FROM user_create_review_on_business U
GROUP BY U.user_idd
ORDER BY cnt DESC)
WHERE ROWNUM = 1;
```

	USER_IDD	CNT
1	CxD0IDnH8gp9FKXzpBHJYXw	793

```
/* 14 */
```

```
SELECT R1.elite_avg - R2.non_elite_avg as average_diff
FROM (SELECT AVG(R.useful) as elite_avg
FROM USER_CREATE_REVIEW_ON_BUSINESS R, Elite E
WHERE R.user_idd = E.user_idd) R1,
(SELECT AVG(R.useful) as non_elite_avg
FROM USER_CREATE_REVIEW_ON_BUSINESS R
MINUS
SELECT R2.useful
FROM USER_CREATE_REVIEW_ON_BUSINESS R2, Elite E
WHERE R2.user_idd = E.user_idd) R2;
```

	AVERAGE_DIFF
1	1.4889051746941258233255853073772578938

```
/* 15 */
```

```
SELECT DISTINCT(B.namee)
FROM Business B, Meal M, suit_good_for_meal S, business_hours H,
(SELECT DISTINCT(R.business_id), PERCENTILE_DISC(0.5)
WITHIN GROUP (ORDER BY R.staar DESC) OVER (PARTITION BY R.business_id ) AS median_star
FROM USER_CREATE_REVIEW_ON_BUSINESS R) T
```

```
WHERE T.business_id = B.business_id AND H.business_id = B.business_id AND
      S.business_id = B.business_id AND S.meal_id = M.meal_id AND
      B.openn = 1 AND T.median_star >= 4.5 AND M.brunch = 1 AND
      (H.dayy = 7 OR H.dayy = 6);
```

	NAMEE
1	Great Harvest Bread co.
2	The Sow's Ear
3	MY Restaurant
4	Parry's Pizzeria & Bar
5	Cool Hand of a Girl

```
/* 16 */
SELECT b1.nameee, AVG(r1.staar) AS stars_count, AVG(b1.review_count)
FROM business b1, locate_3 l1, user_create_review_on_business r1,
      meet_dietary_restrictions m1, diet d1
WHERE b1.business_id = l1.business_id AND r1.business_id = b1.business_id
      AND b1.business_id = m1.business_id AND l1.city = 'los angeles'
      AND m1.diet_id = d1.diet_id AND d1.vegan = 1 AND d1.vegetarian = 1
GROUP BY b1.nameee
ORDER BY stars_count DESC;
```

NAMEE	STARS_C...	AVG(B1.R...
-------	------------	-------------

(empty result set)

```
/* 17 */
SELECT AVG(b_upscale.avg_star) - AVG(b_divey.avg_star)
FROM (
  SELECT AVG(r1.staar) as avg_star, r1.business_id
  FROM user_create_review_on_business r1, experience_ambience e1,
        ambience a1
  WHERE r1.business_id = e1.business_id
        AND e1.ambience_id = a1.ambience_id AND a1.upscale = 1
  GROUP BY r1.business_id) b_upscale,
(SELECT AVG(r2.staar) as avg_star, r2.business_id
 FROM user_create_review_on_business r2, experience_ambience e2,
        ambience a2
 WHERE r2.business_id = e2.business_id
        AND e2.ambience_id = a2.ambience_id AND a2.divey = 1
 GROUP BY r2.business_id) b_divey;
```

	AVG(B_UPSCALE.AVG_STAR)-AVG(B_DIVEY.AVG_STAR)
1	0.25170662281634291940442930091615550496

/\* 18 \*/

SELECT COUNT(\*)

FROM

(SELECT b1.review\_count,

ROW\_NUMBER() OVER (PARTITION BY l1.city ORDER BY b1.review\_count DESC) rank

FROM business b1, locate\_3 l1

WHERE b1.business\_id = l1.business\_id

ORDER BY l1.city) b\_ordered

WHERE b\_ordered.rank = 5 AND b\_ordered.review\_count &gt; 100;

	COUNT(*)
1	81

/\* 19 \*/

SELECT t1.city, t1.sum\_top\_hundred, t2.sum\_all from(SELECT city, statee, SUM(review\_count)

sum\_top\_hundred from (SELECT \* from (SELECT l.city, l.statee, b.business\_id, b.review\_count,

ROW\_NUMBER() OVER(

PARTITION BY l.city, l.statee

ORDER BY b.review\_count DESC

) as rankk

FROM

locate\_3 l, business b WHERE l.business\_id = b.business\_id) where rankk &lt;= 100) GROUP BY (city, statee))t1,

(SELECT city, statee, SUM(review\_count) sum\_all from business b, locate\_3 l where l.business\_id = b.business\_id group by (city, statee))t2 WHERE t1.city = t2.city and t1.statee = t2.statee and

3\*t1.sum\_top\_hundred&gt;2\*t2.sum\_all;

	CITY	SUM_TOP_HUNDRED	SUM_ALL
1	ahwatukee	1042	1042
2	ahwatukee foothills village	11	11
3	alburgh	3	3
4	auburn	4	4
5	auburn twp	10	10
6	avalon	95	95
7	banksville	3	3
8	bradfordwoods	8	8
9	brentwood	227	227
10	brunswick hills	28	28

/\* 20 \*/

```
SELECT * from (SELECT business_id, user_id, ROW_NUMBER() OVER(
                    PARTITION BY business_id
                    ORDER BY user_count DESC
                ) as user_rankk from (SELECT t.business_id, t.review_count, u.user_id, r.user_count
from (SELECT * from(SELECT r.business_id, COUNT(r.business_id) as review_count from
user_create_review_on_business r GROUP BY business_id ORDER BY review_count DESC)t WHERE
rownum<=10)t,
user_create_review_on_business u, (SELECT user_id, COUNT(user_id)user_count from
user_create_review_on_business GROUP BY user_id)r where u.business_id = t.business_id and u.user_id
=r.user_id)) where user_rankk <=3 ;
```

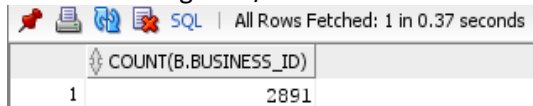
	BUSINESS_ID	USER_ID	USER_RANKK
1	2weQS-Rno0Bhb1KsHKyoSQ	bLbSNkLggFnqWNNzzq-Ijw	1
2	2weQS-Rno0Bhb1KsHKyoSQ	PKEzKWv_FktMm2mGPjwd0Q	2
3	2weQS-Rno0Bhb1KsHKyoSQ	n86B7IkbU20AkxLFX_5aew	3
4	4JNXUY8wbbaaDmk3BPzLWw	d_TB6J3twMy9GChqUEXkg	1
5	4JNXUY8wbbaaDmk3BPzLWw	cMEtAiW60I5wE_vLfTxoJQ	2
6	4JNXUY8wbbaaDmk3BPzLWw	I-4KVZ9lqHhk8469X9FvhA	3
7	5LNZ67Yw9RD6nf4_UhX0jw	QJI90SEn6ujRCtrX06vs1w	1
8	5LNZ67Yw9RD6nf4_UhX0jw	n86B7IkbU20AkxLFX_5aew	2
9	5LNZ67Yw9RD6nf4_UhX0jw	3nDUQBjKyVor5wV0reJChg	3
10	DkYS3arL0hA8si5uUEmH0w	hWDybu_KvYLSdEFzGrniTw	1
11	DkYS3arL0hA8si5uUEmH0w	n86B7IkbU20AkxLFX_5aew	2
12	DkYS3arL0hA8si5uUEmH0w	Fv0e9RIV9jw5TX3ctA1WbA	3

## 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

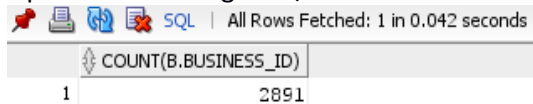
<Initial Running time/IO:



SQL | All Rows Fetched: 1 in 0.37 seconds

	COUNT(B.BUSINESS_ID)
1	2891

Optimized Running time/IO:



SQL | All Rows Fetched: 1 in 0.042 seconds

	COUNT(B.BUSINESS_ID)
1	2891

Explain the improvement: We have two conditions on review\_count and business\_stars. Using index for both of these attributes, makes a difference of order of magnitude (from 370 ms to 42 ms)

Initial 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/>



1	Plan hash value: 3449779207						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		1	58	1061 (1)	00:00:01
7	1	SORT AGGREGATE		1	58		
8	* 2	HASH JOIN		33411	1892K	1061 (1)	00:00:01
9	* 3	TABLE ACCESS FULL	LOCATE_3	33412	880K	616 (1)	00:00:01
10	* 4	TABLE ACCESS FULL	BUSINESS	36208	1096K	445 (1)	00:00:01

Improved plan>

PLAN_TABLE_OUTPUT							
1	Plan hash value: 2960384554						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		1	58	744 (1)	00:00:01
7	1	SORT AGGREGATE		1	58		
8	* 2	HASH JOIN		33411	1892K	744 (1)	00:00:01
9	* 3	TABLE ACCESS FULL	LOCATE_3	33412	880K	616 (1)	00:00:01
10	* 4	TABLE ACCESS BY INDEX ROWID BATCHED	BUSINESS	36208	1096K	128 (0)	00:00:01
11	* 5	INDEX RANGE SCAN	STAR_INDEX	1733		5 (0)	00:00:01
12	-----						

Based on the cost, it's clear that when we define an index on business\_star, we are optimizing the query.

## Query 9

<Initial Running time/IO:

SQL   All Rows Fetched: 10 in 0.489 seconds		
BUSINESS_ID	NAMEE	BUSINESS_STARS
1 UfUddig0a3jDlIrz3Jg-JQ	Fireplace Door Guy	5
2 ZbZCTHxQBv0QGLeL_BYBw	Goldeen Myofascial Release Therapy	5
3 FSeUlx0m0zfNU2-IJnxWA	Melody Events	5
4 meK9y7zRV0N4uaf6sAC8KQ	Finest-Edge Precision Sharpening Service	5
5 UE_fvET7Aj16w7Cluj7ZGg	Beachside Tans	5
6 d8QM8eV7dyjBTTPeveR-RA	Jaclyn Webb Healing	5
7 HUA_z4eTS7hNVqCh00FccQ	Respclearance	5
8 x_vRaIuDevvRK5tQpV9hoA	Pretty Girl Lingo	5
9 YMeWj0dlsvHDGdCKoiGgg	Electric Daisy Carnival	4.5
10 Lbq36N-MFTn9ozAYslZiCA	Couture Pawz	4.5

Optimized Running time/IO:

**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/>

SQL | All Rows Fetched: 10 in 0.022 seconds

BUSINESS_ID	NAMEE	BUSINESS_STARS
1 UfUddig0a3jDlIrz3Jg-JQ	Fireplace Door Guy	5
2 x_vRaIuDsvvRK5tQpV9hoA	Pretty Girl Lingo	5
3 ZbZCTHxQBv0QGLEcL_BYBw	Goldeen Myofascial Release Therapy	5
4 FSeUlxco0mOzfNU2-IJnxWA	Melody Events	5
5 meK9y7zRVon4uaf6sAC8KQ	Finest-Edge Precision Sharpening Service	5
6 UE_fvET7Aj16w7Cluj7ZGg	Beachside Tans	5
7 d8QM8eV7dyjBTTPeve-RA	Jaclyn Webb Healing	5
8 HUA_z4eTS7hNVqCh00FccQ	Respclearance	5
9 YMeWj0dlsvHDGdDCKoiGgg	Electric Daisy Carnival	4.5
10 Lbq36N-MFtn9ozAYslZiCA	Couture Pawz	4.5

Explain the improvement: Significant improvement in the cost because we indexed state attribute from location table which makes it way easier to find California state information.

**Initial plan**

1	Plan hash value: 2908860281						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		10	790	655 (1)	00:00:01
7	* 1	COUNT STOPKEY					
8	2	VIEW		19	1501	655 (1)	00:00:01
9	* 3	SORT ORDER BY STOPKEY		19	1406	655 (1)	00:00:01
10	4	NESTED LOOPS		19	1406	654 (1)	00:00:01
11	5	NESTED LOOPS		19	1406	654 (1)	00:00:01
12	* 6	TABLE ACCESS FULL	LOCATE_3	19	513	616 (1)	00:00:01
13	* 7	INDEX UNIQUE SCAN	SYS_C00118614	1		1 (0)	00:00:01
14	8	TABLE ACCESS BY INDEX ROWID	BUSINESS	1	47	2 (0)	00:00:01
15	-----						

**Improved plan>**

PLAN_TABLE_OUTPUT							
1	Plan hash value: 1236929310						
2							
3	-----						
4	Id	Operation	Name	Rows	Bytes	Cost (%CPU)	Time
5	-----						
6	0	SELECT STATEMENT		10	790	43 (3)	00:00:01
7	* 1	COUNT STOPKEY					
8	2	VIEW		19	1501	43 (3)	00:00:01
9	* 3	SORT ORDER BY STOPKEY		19	1406	43 (3)	00:00:01
10	4	NESTED LOOPS		19	1406	42 (0)	00:00:01
11	5	NESTED LOOPS		19	1406	42 (0)	00:00:01
12	6	TABLE ACCESS BY INDEX ROWID BATCHED	LOCATE_3	19	513	4 (0)	00:00:01
13	* 7	INDEX RANGE SCAN	STATE_INDEX	19		1 (0)	00:00:01
14	* 8	INDEX UNIQUE SCAN	SYS_C00118614	1		1 (0)	00:00:01
15	9	TABLE ACCESS BY INDEX ROWID	BUSINESS	1	47	2 (0)	00:00:01

## General Comments

For query optimization, we tried indexing for a lot of queries, but only two of them seem to be effective. This could be due to the fact that for most of the queries in DDL3, we should have used nested queries, group by, and count of different attributes. This fact automatically reduces the chances for query optimization. For example, if we have to order by within a group, it doesn't matter that much if we have defined index for that attribute and this practice was omnipresent in the queries. On the other hand, since for query 1 we are directly scanning the business table for review\_count and business\_stars, the use of index can be helpful to some extent.