UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

DISEÑO DE BASES DE DATOS: LABORATORIO 2

ANÁLISIS DE EVENTOS SOCIALES MASIVOS

ELÍAS GONZÁLEZ

JOSÉ LATAPIATT

IAN ORELLANA

Profesora: Carolina Bonacic

Ayudantes: Fabián Arismendi

Miguel Cárcamo

Santiago - Chile

26 de marzo de 2015

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

CAPÍTULO 1. INTRODUCCIÓN

En la experiencia anterior, se creo un modelo conceptual con el objetivo de ejemplificar y exponer las distintas entidades y relaciones involucradas en la base de datos de lo que es el sistema que analiza los eventos masivos, vistas de una forma simple e intuitiva. En esta experiencia se busca realizar un modelo que satisfaga los requisitos del sistema, esto es, llevar a cabo distintos procedimientos que den lugar a mayores beneficios o menores costos a la hora de acceder a la base de datos para realizar búsquedas o cualquier otra operación. Junto con esto, se realizan las respectivas consultas que llevara a cabo el sistema para implementar el modelo y se describe el procedimiento de almacenamiento que este lleva a cabo.

CAPÍTULO 2. OBJETIVOS

2.1 OBJETIVO GENERAL

El objetivo, como segunda etapa parcial del proyecto, consiste en realizar un modelo satisfaciendo los distintos requisitos que se plantearon en la experiencia anterior, describiendo las actividades necesarias para su funcionamiento y posterior conexión con los datos a recoger de twitter.

2.2 OBJETIVOS ESPECÍFICOS

- Diseñar un modelo físico de bases de datos, a partir del modelo anterior.
- Normalizar el modelo creado.
- Describir el funcionamiento de la API de twitter.
- Describir las consultas SQL realizadas.
- Describir el procedimiento de almacenado utilizado.

CAPÍTULO 3. ALCANCES

3.1 HERRAMIENTAS UTILIZADAS

- PowerDesigner (Modeling and Metadata Managment), utilizado para diseñar los distintos modelos.

3.2 CONOCIMIENTO NECESARIO

- Se utilizan convenciones de modelos de entidad relación para bases de datos relacionales, tanto modelo conceptual como físico.
- Se utilizan normas convencionales de normalización de tablas de bases de datos.
- Se utilizan consultas a la base de datos en SQL.

CAPÍTULO 4. MODELOS DE BASES DE DATOS

4.1 MODELO CONCEPTUAL

El siguiente modelo presenta las entidades y relaciones existentes en la base de datos, a continuación se exhibirá en detalle los atributos correspondiente a cada tabla junto con la descripción de su utilidad para el sistema.

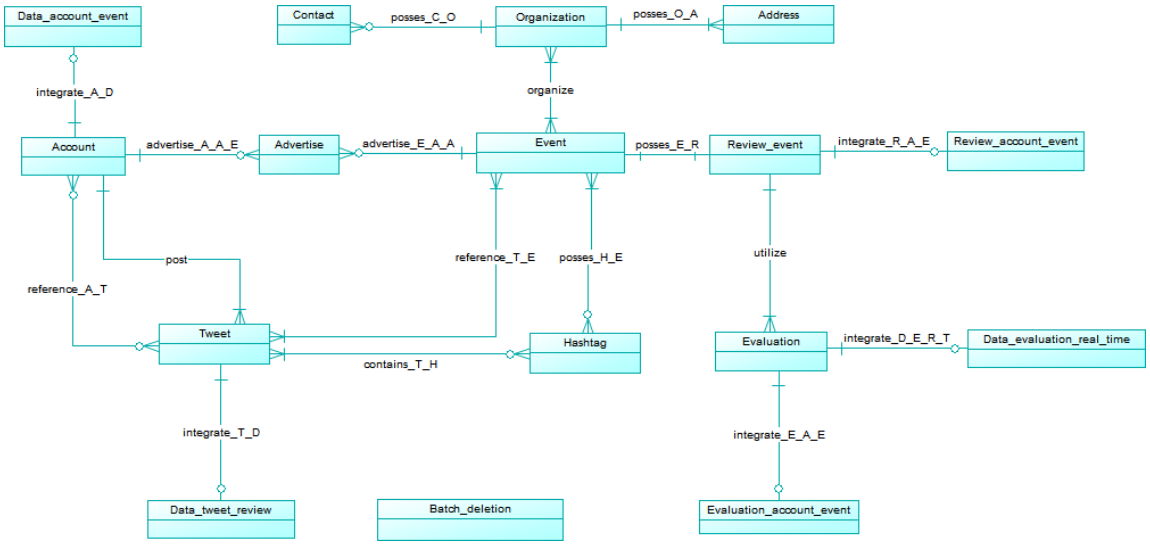

Figura 4.1: Modelo Conceptual de bases de datos.

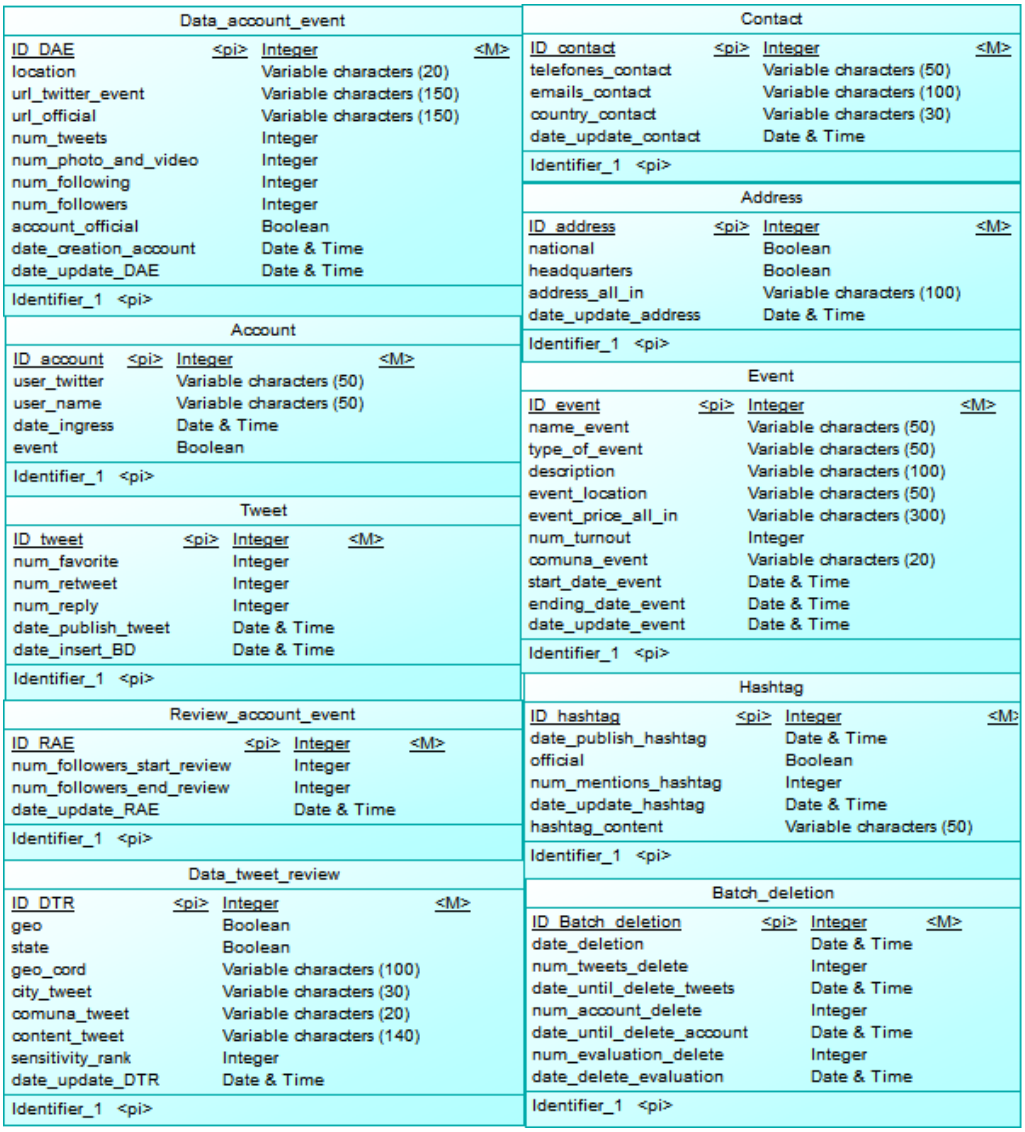

Figura 4.2: Entidades y atributos del modelo conceptual. Sección 1

Data_account_event, es una entidad dirigida a soportar datos correspondientes a las cuentas asociadas a los eventos, entre ellos localización, número de tweets, número de seguidores, fechas en que comenzó a publicar y distingue un atributo exclusivo para la página oficial del evento, diferenciándola del que posee el evento en twitter.

Account contiene los datos específicos de un elemento de estudio, como lo son su twitter y su nombre, además posee un dato tipo *boolean* para identificar si corresponde a una cuenta

oficial de un evento.

Tweet se encarga de contar las veces que un tweet fue retweeteado, compartido o añadido a favoritos, acompañado de su fecha de publicación y .a fecha en que fue añadida a la base de datos. Posee dos tipos de relación con *Account*, puesto que una cuenta puede publicar varios tweets (relación uno a muchos), pero por otro lado, una cuenta puede ser referenciada por varios tweets y a la vez referenciar a muchos otros.

Event captura datos y características del evento, en su mayor parte son los que no se obtienen de twitter, tales como comuna, tipo de evento o descripción. Además añade fechas de inicio y término si es un evento que se lleva a cabo por más de un día.

Hashtag almacena información importante de los hashtag, como su contenido, fecha, hora y número de veces que fue mencionado.

Adress y *Contact*, como su nombre lo indica, guardan información acerca de la dirección física de la organizadora del evento y el contacto por cualquier medio con esta.

Batch_deletion la cantidad de tweets que son borrados y la fechas útiles para evaluar cuando un tweet debe ser eliminado de la base de datos.

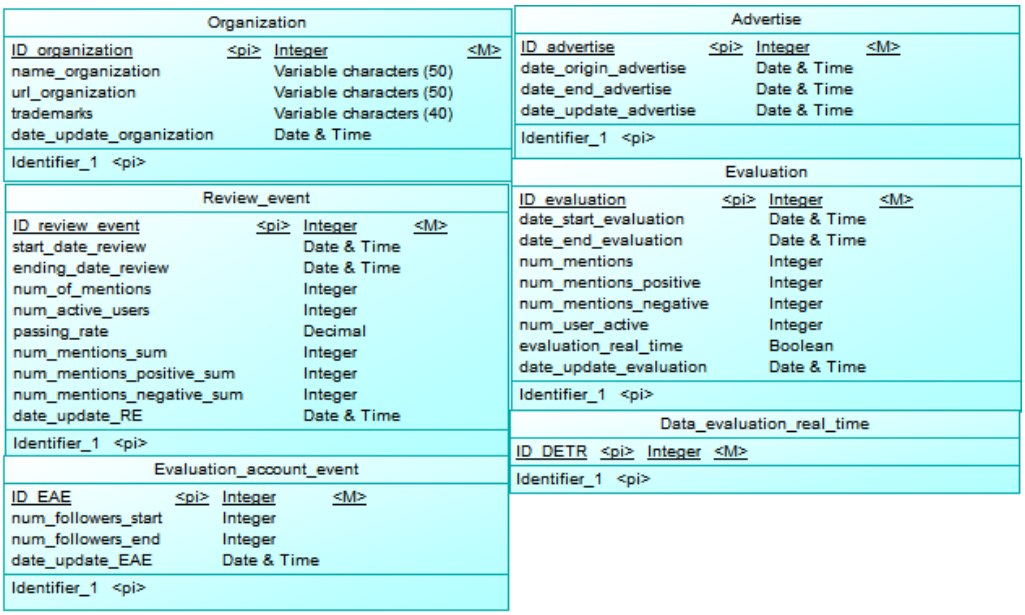

Figura 4.3: Entidades y atributos del modelo conceptual. Sección 2

Organización contiene los datos principales de la organización que dirige el evento, nombre, dirección de su página web y marcas asociadas.

Advertise, es una tabla que conecta *Account* y *Event*, y almacena las fechas de comienzo y fin de publicidad de un evento por parte de una cuenta.

Evaluation contiene información sobre el análisis realizado del evento en un plazo determinado, además, algunas evaluaciones servirán para persistir información en tiempo real en caso de falla de sistema, contiene los datos que se usarán para crear las estadísticas de cierto evento de acuerdo a sus menciones positivas o negativas.

Review_event posee la información final del análisis realizado para determinado evento, para esto maneja atributos como la cantidad total de las menciones positivas y negativas.

Evaluation_acount_event mantiene el número de seguidores a un evento en un tiempo determinado.

Data_evaluation_real_time mantiene información sobre eventuales evaluaciones en tiempo real.

4.2 MODELO FÍSICO

Con el objetivo de producir una descripción base de la implementación de la base de datos, se muestra el modelo físico de bases de datos, el cuál expone además las relaciones de mayor importancia o relaciones base, donde es necesario almacenar ciertos datos puesto que el tipo de relación es de muchos a muchos.

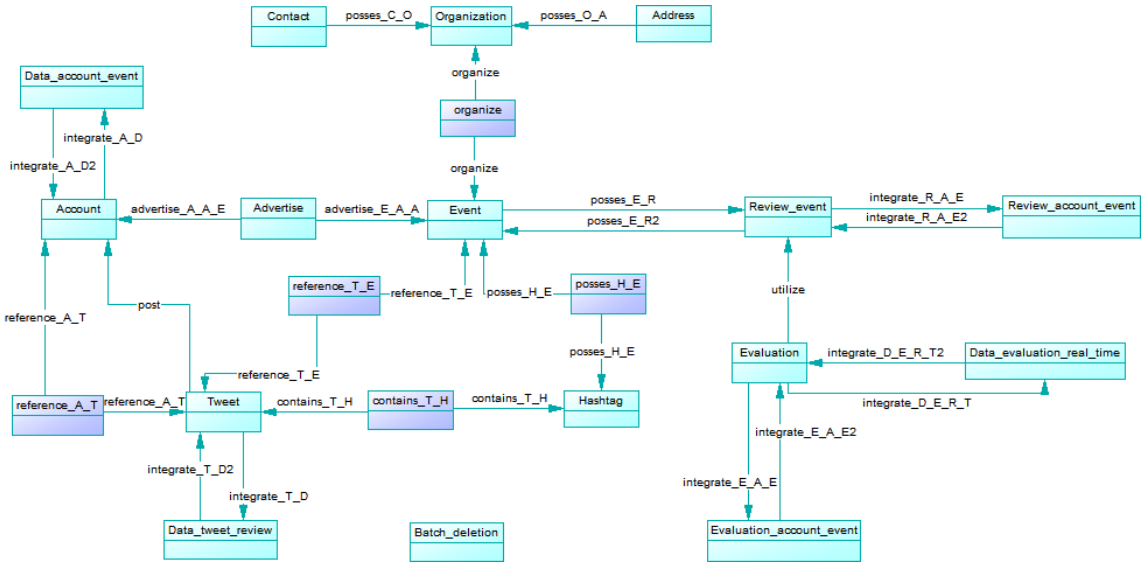

Figura 4.4: Modelo Físico de bases de datos.

Figura 4.5: Entidades y atributos del modelo físico. Sección 1

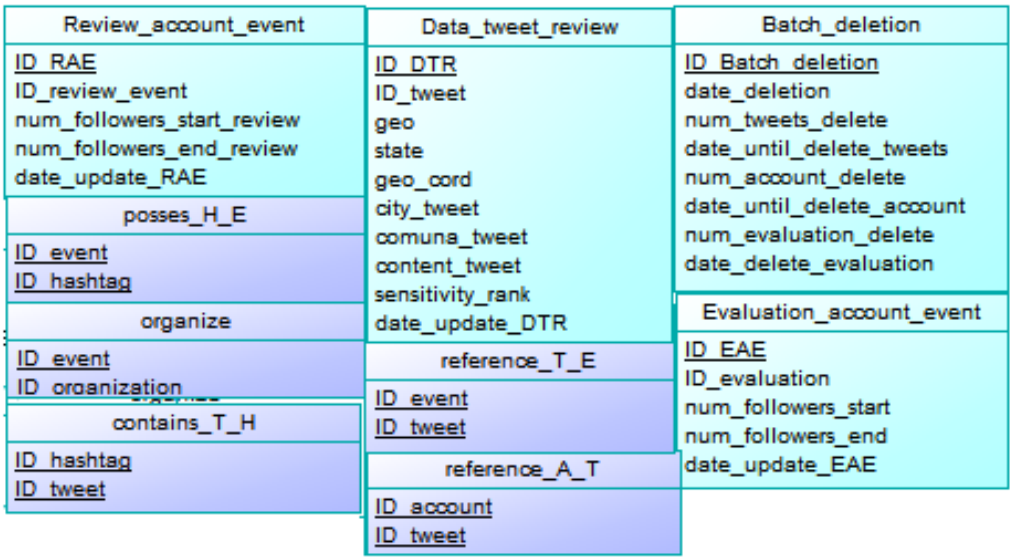

Figura 4.6: Entidades y atributos del modelo físico. Sección 2

reference_A_T es una relación que contiene los identificadores de una cuenta y un tweet que realiza. Esta relación hace referencia a que una cuenta puede referenciar a distintos tweets, y a la vez ser referenciada por varios tweets.

reference_T_E indica que un tweet puede referenciar a varios eventos y un evento puede referenciar a varios tweets. Almacena tanto la identificación del evento como la del tweet.

posses_H_E, indica que un evento puede poseer varios hashtag que lo referencien, y a la vez

contains_T_H hace referencia a la que en un tweet se pueden incluir varios hashtag, y a la vez, un hashtag puede ser referenciado en muchos tweets. Guarda la identificación del tweet y el hashtag utilizado.

organize ya que una organización puede organizar varios eventos, y a la vez un evento puede estar a cargo de varias organizaciones, esta relación se encarga de guardar las identificaciones tanto de la organización como del evento.

4.3 DESCRIPCIÓN DE LA NORMALIZACIÓN

A continuación se mostrarán cada una de las tablas del modelo de bases de datos previamente expuesto seguido de un análisis sobre sus atributos. El proceso de normalización consiste en organizar los atributos y entidades de una forma eficiente en la base de datos, permitiendo mejorar su acceso y evitando errores al agregar o eliminar elementos. El formato mediante el cual se muestran es *Entidad(atributo_1, atributo_2....,atributo_n)*

1.- **Data_account_event(ID_DAE, ID_account, location, url_twitter_event, url_official, num_tweets, num_photo_and_video, num_following, num_followers, account_official, date_creation_account, date_update_DAE)**

Se encuentra en 1FN (Primera forma normal) ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN (Segunda forma normal) ya que está en 1FN, además cada atributo no llave depende de las llaves ID_DAE, ID_account (1:1).

Se encuentra en 3FN (Tercera forma normal) ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC (Forma normal de Boyce Codd), ya que no hay dependencias por transitividad y en 4FN (Cuarta forma normal) no posee dependencias multi-evaluadas.

2.- **Account (ID_account, ID_DAE, user_twitter, user_name, date_ingress, event)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que está en 1FN, además cada atributo no llave depende de las llaves ID_account, ID_DAE, ID_advertise(1:0 |1:0 |1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

3.- **Tweet(ID_tweet, ID_account, ID_DTR, num_favorite, num_retweet, num_reply, date_publish_tweet, date_insert_BD)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuntra en 2FN ya que está en 1FN, además cada atributo no llave depende de las llaves ID_tweet, ID_account, ID_DTR (1:1: 0|1). Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

4.- **Data_tweet_review (ID_DTR, ID_tweet, geo, state, geo_cord, city_tweet, comuna_tweet, content_tweet, sensitivity_rank, date_update_DTR)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares. Se encuentra en 2FN ya que está en 1FN, además cada atributo no llave depende de las llaves ID_DTR, ID_tweet (1:1). Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

5.- **Advertise (ID_advertise, ID_account, ID_event, date_origin_advertise, date_end_advertise, date_update_advertise)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_advertise, ID_account, ID_event (1:1:1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

6.- **Event (ID_event, ID_review_event, name_event, type_of_event, description, event_location, event_price_all_in, num_turnout, comuna_event, start_date_event, ending_date_event, date_update_event)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_event, ID_review_event, ID_organization, ID_advertise (1:1:1 : 0|1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

7.- **Organization (ID_organization, name_organization, url_organization, trademarks, date_update_organization)**

Se encuentra en 1FN ya que todos sus datos son atomicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_organization.

Se encuentra en 3FN ya que esta en 2FN y cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

8.- **Address(ID_address, ID_organization, national, headquarters, address_all_in, date_update_address)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_address, ID_organization (1:*). Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

9.- **Review_event(ID_review_event, ID_event, ID_RAE, start_date_review, ending_date_review, num_of_mentions, num_active_users, passing_rate, num_mentions_positive_sum, num_mentions_negative_sum, date_upgrade_RE)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_review_event, ID_event, ID_RAE (1:1: 0|1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma

funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

10.- **Review_account_event(ID_RAE, ID_review_event, num_followers_start_review, num_followers_end_review, date_update_review)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_RAE, ID_review_event (1:1) .

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

11.- **Evaluation (ID_evaluation,ID_review_event, ID_DETR, ID_EAE, date_start_evaluation, date_end_evaluation, num_mentions, num_mentions_positive, num_mentions_negative, num_user_active, evaluation_real_time, date_update_evaluation)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_evaluation,ID_review_event, ID_DETR, ID_EAE (1:1: 0|1 : 0|1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma fucional transitiva de la llave primaria.

Se encuentra en 4FN no posee dependencias multievaluadas.

12.- **Data_evaluation_real_time (ID_DETR, ID_evaluation)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_DETR, ID_evaluation (1:1).

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

13.- **Evaluation_account_event(ID_EAE, ID_evaluation, num_followers_start, num_followers_end, date_update_EAE)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.
Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_EAE, ID_evaluation (1:1).
Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria. Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

14.- **Hashtag(ID_hashtag, date_publish_hashtag, official, num_mentions_hashtag, date_update_hashtag, hashtag_content)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.
Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las llaves ID_hashtag.
Se encuentra en 3FN ya que esta en 2FN y cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.
Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

15.- **Batch_deletion (ID_Batch_deletion, date_deletion, num_tweets_delete, date_until_delete_tweets, num_account_delete, date_until_delete_account, num_evaluation_delete, date_delete_evaluation)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.
Se encuentra en 2FN ya que esta en 1FN, ademas cada atributo no llave depende de las llaves ID_Batch_deletion.
Se encuentra en 3FN ya que esta en 2FN y cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.
Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

16.- **Reference_A_T (ID_account, ID_tweet)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_account, ID_tweet.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

17.- **Reference_T_E (ID_event, ID_tweet)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_event, ID_tweet.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

18.- **Contains_T_H (ID_hashtag, ID_tweet)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_hashtag, ID_tweet.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

19.- **Organize (ID_event, ID_organization)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_event, ID_organization.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

20.- **Posses_H_E (ID_event, ID_hashtag)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares.

Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_event, ID_hashtag.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

21.- **Contact (ID_contact, ID_organization, Phones_contact, emails_contact, country_contact)**

Se encuentra en 1FN ya que todos sus datos son atómicos o escalares. Se encuentra en 2FN ya que esta en 1FN, además cada atributo no llave depende de las ID_contact, ID_organization.

Se encuentra en 3FN ya que cada atributo no llave de la tabla no depende de forma funcional transitiva de la llave primaria.

Se encuentra en FNBC, ya que no hay dependencias por transitividad y en 4FN no posee dependencias multi-evaluadas.

CAPÍTULO 5. DESCRIPCIÓN Y FUNCIONAMIENTO DE LA API

Twitter cuenta con 3 tipos de APIs, Search API, REST API y Streaming API, para nuestro proyecto se utilizarán las dos últimas, de estas dos, cada una tiene su utilidad particular que se procederá a explicar a continuación.

La REST API es de las dos que será utilizada y la que posee más funcionalidades. Por otro lado, su mayor restricción, un uso de máximo 350 usos por hora de forma autentificada, lo que imposibilita usarla para todos los requerimientos. En particular esta API puede ser utilizada para subir tweets, agregar localización, seguir gente, crear listas, a grandes razgos lo que puede hacer un usuario de twitter desde el sitio web, incluyendo la búsqueda de contenido. Cabe destacar el hecho de que esta API al buscar, permite conseguir Tweets de una antigüedad superior a la streaming API, con un máximo actual de 3200 Tweets. Su utilidad para el proyecto involucra la obtención de los datos de las cuentas de los eventos.

La Streaming API en cambio, permite mantenerse conectados a Twitter y recibir los Tweets con keywords, configuraciones de locación e idioma que son requeridas, recibiendo los datos casi en tiempo real en comparación a lo que se esta subiendo a Twitter. Es decir, se recibe información del flujo de la misma que es al final el servicio de Twitter. En este caso la limitante esta que no se puede conseguir los tweets más antiguos con respecto a un tema, para ello se ocupará la Rest API. Esta API es particularmente efectiva en trabajos de minería de datos, de análisis de grandes cantidades de tweets a través del tiempo, que es lo que se espera lograr con el proyecto. La streaming API tiene a su vez tres grandes formas de usarse, statusesfirehose que es un gran flujo de toda la información recibida por twiter sin mayores filtros, statusessample que entrega una muestra aleatoria de lo que se busca y un filtrado (statusesfilter) a base de keywords, a estas también se puede ocupar statuses/links para obtener solo tweets que incluyan links o statusesretweet para obtener solo tweets que sean retweets, lo que ayudara a separar estos tipos de tweets para generar

las estadísticas mencionadas en los requerimientos del sistema.

Para conectarse con la API de Twiter se utiliza RUBY on Rails, en particular las gema Twiter.

CAPÍTULO 6. DESCRIPCIÓN DE LA APLICACIÓN

Las funciones que el sistema debe implementar incluyen la exposición en cada sección correspondiente a un evento, de sus atributos convencionales, ubicación, precio de entradas, empresa organizadora, fecha, hora y tipo de evento (musical (género), deportivo, artístico); Además debe obtener el número de seguidores de cada evento, número al cuál el usuario puede acceder.

El sistema debe clasificar los tweets según su intención positiva o negativa, y exponer dicha clasificación mediante una estadística simple de porcentajes apoyada por un gráfico de torta.

El sistema debe monitorear los tweets referidos a un evento, controlando la fecha y hora de su emisión para luego, con la ayuda de un gráfico de líneas, tener la posibilidad de acceder a la forma en que se altera el volumen de tweets emitidos a lo largo del tiempo, o por un periodo de tiempo a seleccionado por el usuario.

El sistema debe mostrar un ránking con los 10 eventos que posean más tweets, así como los hashtags más utilizados.

El sistema debe proveer una vista que permita al usuario seleccionar el tipo de evento que quiere observar.

El sistema debe incluir una vista donde se muestren los tweets emitidos de cierto evento geo-localizados en un mapa.

El objetivo del observatorio es principalmente el de poder realizar comparaciones entre los distintos eventos para crear un fácil análisis de los eventos más populares y sus posibles causas. El incluir la medición del volumen de tweets a través del tiempo, contribuye al estudio del impacto que causan estos eventos durante ciertos periodos, y pueden denotar eventos importantes que hicieron que en cierto momento se hablara con mayor frecuencia sobre cierto evento. El objetivo de la geo-localización es proporcionar información útil para usuarios interesados en realizar estudios sociales que involucren tendencias de sectores de la región a asistir a cierto tipo de eventos o para la empresa organizadora, ya que entrega un indicio de los sectores que comentan más el evento que organizan, dato que puede ser de utilidad para organizar publicidad.

CAPÍTULO 7. DESCRIPCIÓN DE CONSULTAS SQL

En esta sección se exhibirán consultas SQL realizadas en la aplicación utilizando Ruby on Rails, junto a una descripción de su utilidad para completar los objetivos de la aplicación.

- Obtener eventos más populares:

  ```
  a = ReviewEvent.order('num\_of\_mentions').pluck(:id)
  b = Event.where('id = ?', a).pluck(:name\_event)
  b.limit(10)
  ```

  Obtiene el número de menciones de la tabla ReviewEvent, utilizándolo como criterio de orden descendente, exhibiendo solo los 10 primeros tweets con más menciones. En otras palabras, obtiene los 10 eventos con más menciones, que serán incluidos como "los más populares" dentro de la aplicación.

- Seleccionar el tipo de evento

  ```
  c = Event.where('type\_of\_event = ?', '<TIPO>')
  ```

  Obtiene la tabla con los eventos que pertenecen al tipo de evento seleccionado por el usuario.

- Tweets por periodo de tiempo

  ```
  d = Tweet.where('date\_publish\_tweet > ? AND
  date\_publish\_tweet < ?', datetime1, datetime2)
  ```

  Obtiene de la tabla tweets, aquellos datos que cumplan con las restricciones de tiempo ingresadas. Esta consulta entrega los datos útiles para usuarios que se interesan en la cantidad de tweets emitidos durante cierto periodo, y en el caso de requerir la variación de volumen a lo largo del tiempo, se obtienen el nombre del evento y la fecha y hora en que el tweet fue emitido.

- Obtener cantidad de tweets positivos o negativos del evento

```
e = Event.where('name\_event = ?', '<NombreEvento>')
.pluck(:id)
```

Primero se obtiene el id de la tabla Event (que guarda los datos del evento), donde el atributo name_event (nombre del evento) coincida con el campo NombreEvento.

```
f = ReferenceTE.where('event\_id = ?', e).
pluck(:tweet\_id)
```

El siguiente paso es, obtener el identificador del tweet de cada tweet presente en la tabla ReferenceTE, cuyo identificador del evento coincida con el de la tabla de datos obtenida anteriormente.

```
g = DataTweetReview.where('tweet\_id = ? AND
sensitivity\_rank = ?',f,0).count
```

Finalmente se cuentan los tweets de la tabla DataTweetReview donde son contados los tweets cuyo sensivity_rank corresponda con el valor que desea ser estudiado. Este atributo solo posee dos valores posibles, 1 y 0, de esta forma se guardan con valor 1 los comentarios considerados negativos y con un 0 los positivos.

CAPÍTULO 8. DESCRIPCIÓN DE TRIGGERS UTILIZADOS

Los *triggers* utilizados para este sistema pueden dividirse en dos tipos, aquellos que son utilizados cuando la base de datos esta vacía (*INSERT ON*, ubicados en el archivo adjunto [*TriggersAllInInsert.sql*]), y aquellos que se usan una vez que se ha poblado la base de datos (*UPDATE ON*, ubicados en el archivo adjunto [*TriggersAllInUpdate.sql*]). Los *triggers* se usan para lanzar ciertas acciones en el momento que se realiza alguna operación en la base de datos, los procedimientos *INSERT ON*, como su nombre lo indica, insertan datos en tablas de la base de datos cuando estas están vacías, y *UPDATE ON*, actualiza estos campos. A continuación se expone una muestra del código perteneciente a los *triggers* utilizados para el sistema.

```
Drop trigger if exists TriggerInsert_DAE;
--Borra el trigger anterior con el mismo nombre en caso de que exista para crer uno nuevo.
DELIMITER \\
CREATE TRIGGER TriggerInsert_DAE before INSERT ON data_account_events
--Crea trigger tipo 'INSERT ON', es decir, inserta donde no hay datos previamente
FOR each row -- Para cada fila
BEGIN
      declare msg varchar(255);
      --Se declara una variable llamada 'msg' tipo varchar para cadenas de caracteres de tamaño 255.
      if LOCATE('https://twitter.com/', new.url_twitter_event) != 1 then
      --Si el url almacenado en url_twitter_event no incluye la secuencia de carcteres https://twiter.com/ :
        set msg = concat('TriggerInsert_Account: Mal formato de entrada , requiere "https://twitter.com/*****":', cast(new.id as char));
      --Se carga 'msg' con el mensaje que indica el error, en este caso, mal formato de entrada.
        signal sqlstate '45000' set message_text = msg;
        --Se entrega el mensaje de error (sqlstate 4500 es un valor generico que ilustra excepciones no manejadas o definidas)
      end if;
      if new.num_tweets<0 or new.num_photo_and_video<0 or new.num_following<0 or new.num_followers<0 then
        --Si el número de tweets, seguidores, o cualquier otra variable contable es menor a 0 entonces
        set msg = concat('TriggerInsert_Account: Mal formato de entrada , numeros deben ser positivos":', cast(new.id as char));
        --Se carga 'msg' con el mensaje que indica el error, en este caso, los números deben ser positivos.
        signal sqlstate '45000' set message_text = msg;
        --se entrega el mensaje de error
      end if;
          set new.updated_at = now(); -- La variable que guarda la fecha de actualización guarda el valor que corresponde a la fecha y hora actual
      set new.created_at = now();     -- Se modifica la variable que guarda la fecha de creación con los valores actuales de fecha y hora
          set new.id = null;
END \\
DELIMITER ;
```

Figura 8.1: Trigger *INSERT ON*

Extracto del archivo adjunto *TriggersAllInsert*.

```
Drop trigger if exists TriggerUpdate_DAE;
DELIMITER \\
CREATE TRIGGER TriggerUpdate_DAE before UPDATE ON data_account_events
FOR each row
BEGIN
      declare msg varchar(255);
      if LOCATE('https://twitter.com/', new.url_twitter_event) != 0 then
        set msg = concat('TriggerInsert_Account: Mal formato de entrada , requiere "https://twitter.com/*****":', cast(new.id as char));
        signal sqlstate '45000' set message_text = msg;
      end if;
      if new.num_tweets<0 or new.num_photo_and_video<0 or new.num_following<0 or new.num_followers<0 then
        set msg = concat('TriggerInsert_Account: Mal formato de entrada , numeros deben ser positivos":', cast(new.id as char));
        signal sqlstate '45000' set message_text = msg;
      end if;
          set new.updated_at = now();
      set new.created_at = old.created_at;
          set new.id = old.id;
END \\
DELIMITER ;
```

Figura 8.2: Trigger *UPDATE ON*

Extracto del archivo adjunto *TriggersAllUpdate*.

Como se puede observar, la única diferencia de este código con el anterior es que se crea utilizando UPDATE ON en lugar de INSERT ON, es decir, se utiliza para actualizar la tabla pero el procedimiento para agregar un elemento es el mismo. Además, la fecha de actualización en el *trigger* UPDATE ON es modificada, mientras que se mantiene la fecha de creación (*set new.created_at = old.created_at*) y la identidad (*set new.id = old.id*).

CAPÍTULO 9. CONCLUSIÓN

Se logra crear un modelo adecuado para almacenar los datos requeridos, normalizado y sin pérdida de información.

Se identifican las API a utilizar, necesarias para los tipos de datos que se determina obtener, la REST API para obtener datos de eventos y tweets antiguos con respecto a un tema, y streaming API para analizar grandes flujos de datos para generar datos estadísticos propuestos en los objetivos del sistema.

Puesto que la mayoría de los objetivos de la aplicación se relacionan con comparaciones entre los eventos, relacionando la magnitud de tweets que reciben, se crear consultas sql que permiten obtener los datos necesarios para cumplir con los requerimientos funcionales que posee el sistema.

CAPÍTULO 10. REFERENCIAS

Twitter Developers (2014). GET statuses/firehose. [ONLINE] Available at: https://dev.twitter.com/streaming/reference/get/statuses/firehose. [Last Accessed e.g. 31 August 11].

2014 Twitter, Inc. Twitter API limits. [ONLINE] Available at: https://support.twitter.com/articles/160385-twitter-api-limits# [Last Accessed e.g. 31 August 11].

Active Record Query Interface [ONLINE] Available at: http://guides.rubyonrails.org/active_record_querying.html [Last Accessed e.g. 31 August 11].