Examen del módulo Bases de Datos

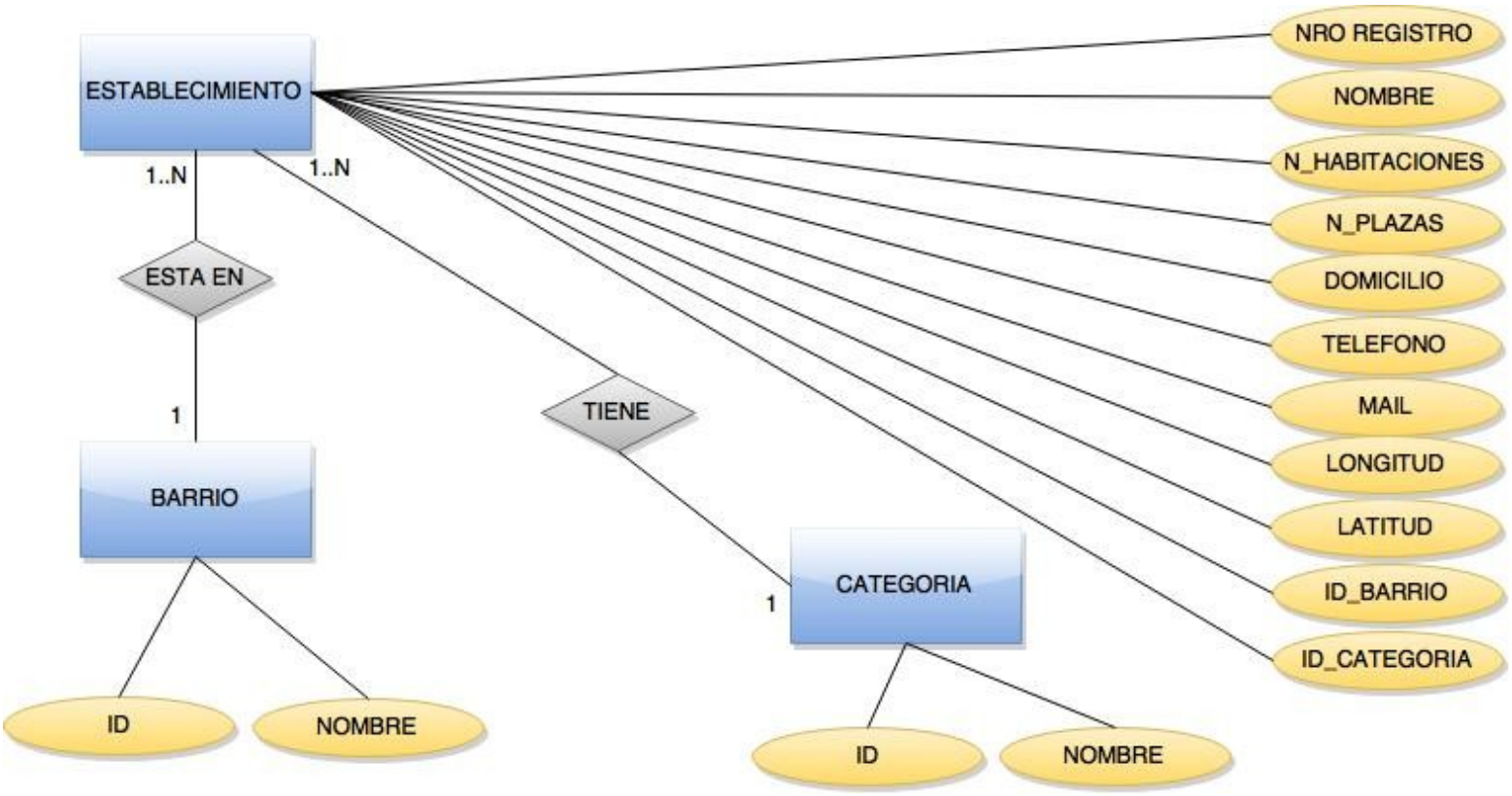
Emiliano Dalla Verde Marcozzi

Presentación del caso de estudio

Utilizo el archivo con formato csv[i] con el listado de los alojamientos turísticos categorizados por el Ente de Turismo e inscriptos en el registro de Prestadores Turísticos de la provincia de Buenos Aires a fin de volcar dichos datos en el SGBD PostgreSQL. Sería de utilidad una aplicación que me permita importar el archivo csv en un SGBD, y ademas separe en entidades de relación diferentes los atributos “barrio” y “categoria”.

El modelo relacional y el álgebra relacional

El esquema de relación planteado en el archivo csv[ii] se encuentra normalizado en 3FN. A fin de hacer más atractivo el ejercicio decido separar en esquemas de relación diferentes los atributos **`BARRIO`** y **`CATEGORIA`**. Ademas el separar **`BARRIO`** y **`CATEGORIA`** permite actualizar el nombre de las categorías o barrios, sin tener que actualizar todos los valores del esquema de relación **`ESTABLECIMIENTO`**.



Determinamos las relaciones, atributos y dominios. Los esquemas de relación son los siguientes:

ESTABLECIMIENTO(nro_registro, nombre, n_habitaciones, n_plazas, domicilio, telefono, mail, longitud, latitud, id_barrio, id_categoria)

BARRIO(id, nombre)

CATEGORIA(id, nombre)

En los esquemas de relación **BARRIO** y **CATEGORIA** he decidido agregar un atributo id, el cual a posterior voy a utilizar como clave primaria. Esto nos permite que los esquemas de relación **BARRIO** y **CATEGORIA** puedan actualizar su atributo nombre sin romper la integridad con el esquema de relación **ESTABLECIMIENTO**, puesto que la clave primaria utilizada para la relación es el atributo id.

A continuación defino los dominios de los atributos de los esquemas de relación:

ESTABLECIMIENTO:

dominio(nro_registro): números

dominio(nombre): texto

dominio(n_habitaciones): números

dominio(n_plazas): números

dominio(domicilio): texto

dominio(teléfono): texto

dominio(mail): texto

dominio(longitud): números

dominio(latitud): números

BARRIO:

dominio(id): números

dominio(nombre): texto

CATEGORIA:

dominio(id): números

dominio(nombre): texto

A continuación defino las claves candidatas y primarias:

ESTABLECIMIENTO:

Claves candidatas: {nro_registro}

Clave primaria: {nro_registro}

BARRIO:

Claves candidatas: {id, nombre}

Clave primaria: {id}

CATEGORIA:

Claves candidatas: {id, nombre}

Clave primaria: {id}

Las claves foráneas son las siguientes:

ESTABLECIMIENTO:

Tiene de clave foránea los atributos:

- `id_barrio`, que establece la relación con el atributo `id`, clave primaria del esquema de relación **BARRIO**.

- `id_categoria`, que establece la relación con el atributo `id`, clave primaria del esquema de relación **CATEGORIA**.

Respecto de las reglas de integridad, defino lo siguiente:

CATEGORIA: No se puede borrar una categoría si su clave primaria está siendo referenciada por el atributo id_categoria de un **ESTABLECIMIENTO**.

BARRIO: No se puede borrar un barrio si su clave primaria está siendo referenciada por el atributo id_barrio de un **ESTABLECIMIENTO**.

Para probar el modelo relacional realizamos las siguientes consultas con algebra relacional:

Suponiendo que el atributo `id` con valor 1 del esquema de relación **`BARRIO`** sea `SAN NICOLAS`, seleccionamos los establecimientos que se encuentren en el barrio `SAN NICOLAS`.

R := ESTABLECIMIENTOS(ID_BARRIO = 1)

Ahora suponiendo que el atributo `id` con valor 1 del esquema de relación **`CATEGORIA`** sea "`5 estrellas`", selecciono los establecimientos que se encuentren en el barrio `SAN NICOLAS` y que sean de categoría `5 estrellas`:

R := ESTABLECIMIENTOS(ID_BARRIO = 1 y ID_CATEGORIA = 1)

También puedo obtener un establecimiento turístico mediante su `nro_registro`:

R := ESTABLECIMIENTOS(NRO_REGISTRO = 119)

El lenguaje SQL

A continuación defino las sentencias a ejecutar en el SGBD PostgreSQL

Creo la base de datos:

```
postgres=# CREATE DATABASE alojamientos;
CREATE DATABASE
postgres=# \l
                                  List of databases
     Name      |  Owner   | Encoding  | Collate | Ctype |   Access privileges
---------------+----------+-----------+---------+-------+-----------------------
 alojamientos  | postgres | SQL_ASCII | C       | C     |
 postgres      | postgres | SQL_ASCII | C       | C     |
 recuento2007  | postgres | SQL_ASCII | C       | C     |
 recuento2013  | postgres | SQL_ASCII | C       | C     |
```

Creo las tablas

```
alojamientos=# CREATE TABLE categoria (
    id SERIAL PRIMARY KEY,
    nombre text UNIQUE NOT NULL
);
CREATE TABLE
alojamientos=# CREATE TABLE barrio (
    id SERIAL PRIMARY KEY,
    nombre text UNIQUE NOT NULL
);
CREATE TABLE
alojamientos=# CREATE TABLE establecimiento (
    nro_registro integer PRIMARY KEY,
    nombre text NOT NULL,
    n_habitaciones integer,
    n_plazas integer,
    domicilio text NOT NULL,
    telefono text,
    mail text,
    longitud float,
    latitud float,
    id_barrio integer REFERENCES barrio (id),
    id_categoria integer REFERENCES categoria (id)
);
CREATE TABLE
alojamientos=#
```

A continuación indico algunas sentencias de manipulación corrientes para completar la documentación:

Nuevo barrio:

```
INSERT INTO BARRIO VALUES (“SAN NICOLAS”);
```

Nueva categoría:

```
INSERT INTO CATEGORIA VALUES (“3 estrellas”);
```

Nuevo establecimiento:

```
INSERT INTO ESTABLECIMIENTO VALUES (1, “Chascomús Apart”, 10, 10, “Lavalle 133”, “4112233”, “chascomúsapart@gmail.com”, -58.3743, -34.6019, 1, 1)
```

Cambio de teléfono de un establecimiento:

```
UPDATE ESTABLECIMIENTO SET telefono = “4221133” where nro_registro = 1;
```

Análisis comparativo de bases de datos

Característica	**Firebird**	**MySQL**	**Oracle**	**PostgreSQL**
Licencia	IPL/IPDL	GPL/Comercial	Propietario	Licencia BSD
Multiplataforma	Si	Si	Si	Si
Clusterizable	No	Si	Si	Si
Comunidad	Buena	Muy buena	Buena	Excelente
Valoración	Buena	Buena	Muy buena	Excelente
Difusión	Baja	Muy buena	Buena	Buena
Soporte	No	Comercial	Comercial	No

iiiiv

Consideraciones de su implementación en PostgreSQL

Respecto de la seguridad e integridad de los datos es conveniente:

- Auditar el código SQL, esto consiste en revisar el código SQL en busca de vulnerabilidades, por ejemplo posibles inyecciones SQL. Esto sucede al momento de obtener datos del usuario, es conveniente revisar o no permitir ingresar valores que puedan ser utilizados para ejecutar sentencias SQL.
- Definir restricciones de integridad. Estas restricciones son reglas que determinan que se puede o no hacer con los datos, por ejemplo al querer borrar un registro, o al querer actualizar un valor con un tipo de dato diferente del dominio perteneciente al atributo que se está actualizando.
- En caso de ser necesario, se puede crear un registro de auditoría para monitorizar una tabla particular[v]

- Proteger y resguardar los logs (por ejemplo en un medio o servidor externo).
- Tener una configuración segura en el archivo pg_hba.conf. Este fichero determina que usuarios tienen permisos para conectarse a la base, entre otras settings.
- Realizar backups periódicos de la base de datos. En PostgreSQL se puede utilizar la herramienta de linea de comandos pg_dump cual se utiliza por ejemplo como:

  ```
  pg_dump alojamientos > alojamientos.sql
  ```

  Luego para restaurar la base se puede hacerlo de la forma:

  ```
  psql alojamientos < alojamientos.sql
  ```

Creando la base y tablas en PostgreSQL

En mi caso voy a trabajar con el usuario postgres cual tiene acceso al servidor PostgreSQL. Primeramente me logueo al shell con el comando ***psql***, y luego creo la base de datos con el comando: `CREATE DATABASE alojamientos;`

Luego para crear las tablas, disponemos de un archivo ***createTables.sql***, el cual tiene definidas las sentencias para crear las tablas. Para crear las tablas ejecutamos el siguiente comando: `psql -U postgres -h localhost -d alojamientos < createTables.sql`

Podemos chequear de que las tablas fueron creadas logueandonos en el shell de postgres, con los siguientes comandos:

- `psql`
- `\c alojamientos;`
- `\d`

Para importar los datos del archivo alojamientos.csv en la base de datos vamos a utilizar la aplicación app.py la cual creará dos archivos csv los cuales son:

- El archivo `categories.csv` es una set de nomres de categorias extraídas desde `alojamientos.csv`.
- El archivo `neighborhoods.csv` es un set de nombres de barrios extraídos desde `alojamientos.csv`

Para crear los archivos, la aplicación ***app.py*** provee un método `write_all_csv_files.`

Para importar el contenido de los tres archivos csv generados utilizamos el comando COPY, en mi caso de la siguiente forma:

```
COPY categoria(nombre) from '/var/lib/pgsql/categories.csv';
```

```
COPY barrio(nombre) from '/var/lib/pgsql/neighborhoods.csv';
```

Por último nos falta importar todos los alojamientos, a los cuales hemos representado en la base de datos como el esquema de relación **`ESTABLECIMIENTO`**. Para importar todos los establecimientos, hacemos uso del método `insert_places` definido en ***app.py***.

Análisis FODA - Fortalezas

PostgreSQL	MySQL
Integridad: PITR – WAL Concurrencia: Múltiples usuarios – escritura MVCC Funcionalidad: tipos (OO), reglas, lenguajes embebidos	Escalabilidad: replicación, clustering Flexibilidad: Múltiples Motores de Almacenamiento Velocidad: Sin integridad / concurrencia.

Análisis FODA – Oportunidades

PostgreSQL	MySQL
MySql -> Sun -> Oracle: forks: MariaDB, Drizzle, ... Estable y Sólido: (MySQL 5.1 != bug free)	Nuevos Motores de Almacenamiento (Maria/Falcon?) Gran aceptación por parte del mercado (11 millones de instalaciones)

Análisis FODA – Debilidades

PostgreSQL	MySQL
Soporte Comercial Ausencia de replicacion nativa (roadmap 8.5) Ausencia de cluster nativo	(Modelo de Arquitectura) SQL rudimentario Planner Basico MyISAM no soporta FKs (roadmap 5.2)

Análisis FODA – Amenazas

PostgreSQL	MySQL
	Oracle compra InnoDB Oy Sun Compra MySQL Oracle compra SUN

i http://data.buenosaires.gob.ar/dataset/alojamientos-turisticos
ii https://recursos-data.buenosaires.gob.ar/ckan2/alojamientos-turisticos/alojamientos.csv
iii https://es.wikipedia.org/wiki/Oracle_Database
iv https://es.wikipedia.org/wiki/Anexo:Comparaci%C3%B3n_de_sistemas_administradores_de_bases_de_datos_relacionales

v http://www.alberton.info/postgresql_table_audit.html#.VbutVq0Viko
vi http://www.postgresql.org.ar/trac/raw-attachment/wiki/PgDayUnnoba/PDF_pgsql_mysql_FODA.pdf