Bases de Datos

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Trabajo Práctico 2: *Histórico de Accesos y Facturación*

7/07/2018

Integrante	LU	Correo electrónico
Enzo Samuel Cioppettini	405/15	tenstrings5050@gmail.com
Tomás Ariel Pastore	266/15	pastoretomas96@gmail.com
Patricio López Valiente	457/15	patriciolopezvaliente@gmail.com
Fernando Balboa	246/15	fbalboa95@gmail.com

Reservado para la cátedra

Instancia	Docente	Nota
Primera entrega		
Segunda entrega		

Índice

1. Introducción **3**

2. Modelo Conceptual **4**

3. Modelo de Interacción de Documentos **5**

3.1. Asunciones 6

3.2. Diseño lógico (JSONSchema) 7

4. Funcionalidades a implementar **12**

5. Experimentación con Sharding **14**

6. Generación automática de datos **15**

7. Conclusiones **15**

1. Introducción

En el trabajo práctico 1, diseñamos e implementaremos una Base de Datos relacional para "Entretenimiento Completo S.A.", cuyo principal interés es proveer tarjetas de acceso personalizadas tanto a parques de diversiones como a eventos, basándose en un programa de fidelidad.

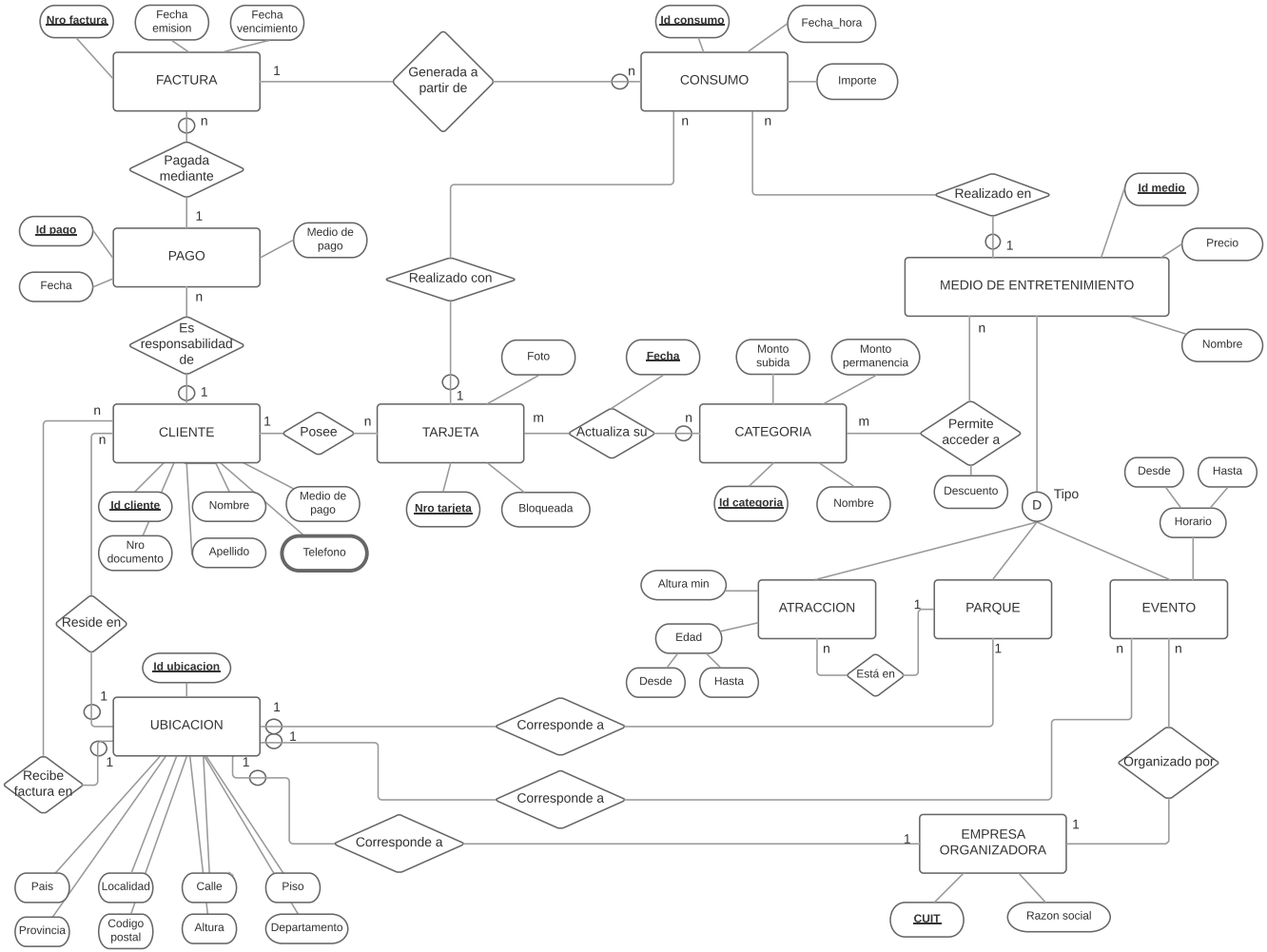
Tomando en cuenta el crecimiento de la base de datos por la enorme cantidad de usuarios de las tarjetas, el nuevo requerimiento de E.C.S.A. es guardar el historial de visitas a parques, atracciones y eventos evitando afectar la base de datos relacional de transacciones antes mencionada. Se desea, además, guardar además el histórico de facturas. Para implementar estas funcionalidades vamos a utilizar RethinkDb como base de datos NoSQL basada en documentos.

Las bases de datos NoSQL tienen ciertas ventajas en comparación con las relacionales (SQL), entre ellas podemos mencionar:

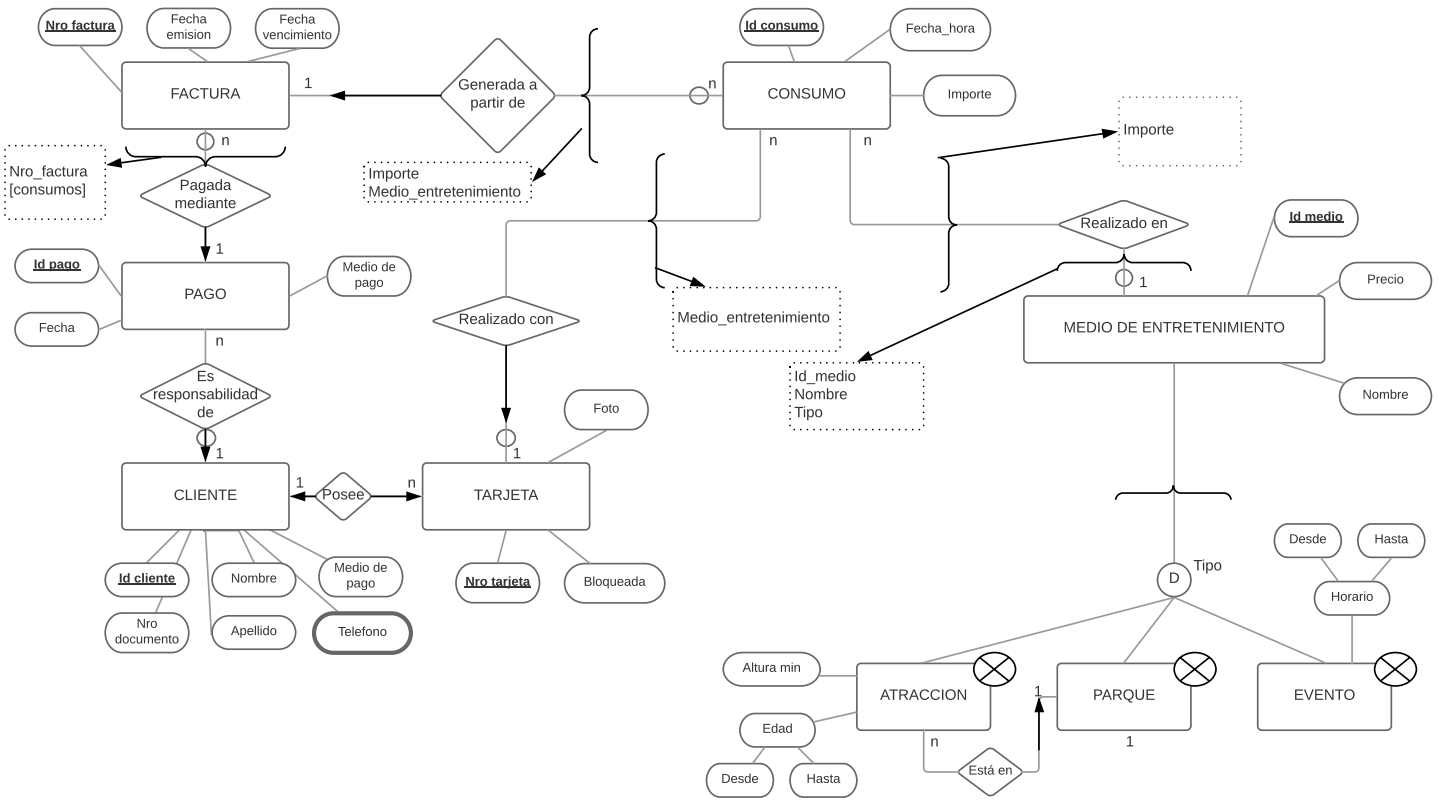
- Representación de datos libre de esquemas. No hay que pensar demasiado para definir una estructura y se puede seguir evolucionando en el tiempo (agregando nuevos campos o anidando datos) a medida que se agrega funcionalidad.
- Tiempo de desarrollo. Al ser la representación de datos mas adecuada para el problema, se pueden evitar ciertos JOINs extremadamente complejos.
- Velocidad. En muchos casos es posible dar respuesta en el orden de los milisegundos en vez de cientos de milisegundos. Esto es beneficioso en celulares y otros dispositivos con conexion intermitente.
- Planificar escalabilidad. La aplicacion puede ser bastante elastica y manejar picos repentinos de carga.

2. Modelo Conceptual

El modelo conceptual es el mismo que el del trabajo práctico 1. Recordamos el diagrama de entidad relación correspondiente.



3. Modelo de Interacción de Documentos



3.1. Asunciones

1. Para la consulta **Lista de atracciones visitadas por un cliente dado** asumimos que sólo se desea el id y el nombre de la atracción.

2. Para la consulta **Top 5 de Eventos con mayor facturación** asumimos que se desea considerar sólo los consumos que ya fueron pagados.

3. Para la consulta **Total de importes por atracción en un parque dado** asumimos que se desea considerar los importes de todos los consumos y no sólo los de consumos pagos.

3.2. Diseño lógico (JSONSchema)

```
{
 "title": "Cliente",
 "description": "Properties of a client",
 "type": "object",
 "properties":{
    "id": {"type":"integer"},
    "nombre": {"type": "string"},
    "apellido": {"type": "string"},
    "dni": {"type": "integer"},
    "medio_pago": {"type": "string"},
    "telefonos": {"type": "array", "items": {"type": "string"}},
    "tarjetas": {"type": "array", "items": {"type": "integer"}}
 },
 "required":["id","nombre","apellido","dni","medio_pago","telefonos","
    tarjetas"],
 "additionalProperties": false
}
```

```
{
 "title": "Consumo",
 "type": "object",
 "properties": {
  "id": {"type": "integer"},
  "fecha_hora": {"type": "string", "format": "date-time"},
  "importe": {"type": "number"},
  "nro_factura": {"type": "integer"},
  "nro_tarjeta": {"type": "integer"},
  "medio_entretenimiento": {
   "type": "object",
   "properties": {
    "id": {"type": "integer"},
    "nombre": {"type": "string"},
    "tipo": {
     "type": "object",
     "properties": {
      "nombre":{"type": "string", "enum": ["ATRACCION", "PARQUE", "
         EVENTO"]}
     },
     "required":["nombre"]
    }
   },
   "required":["id","nombre","tipo"]
  }
 },
 "required":["id","fecha_hora","importe","nro_factura","nro_tarjeta","
    medio_entretenimiento"],
 "additionalProperties": false
```

```
}

{
 "title": "Tarjeta",
 "type": "object",
 "properties":{
  "numero": {"type":"integer"},
  "bloqueada": {"type": "boolean"},
  "foto": {"type": "string"},
  "id_cliente": {"type": "integer"},
  "consumos": {
   "type": "array",
   "items": {
    "type": "object",
    "properties": {
     "medio_entretenimiento": {
      "type":"object",
      "properties":{
       "id": {"type": "integer"},
       "nombre": {"type": "string"},
       "tipo": {
        "type": "object",
        "properties": {
         "nombre":{"type": "string", "enum": ["ATRACCION", "PARQUE", "
            EVENTO"]}
        },
        "required":["nombre"]
       }
      },
      "required":["id","nombre","tipo"]
     }
    },
    "required": ["medio_entretenimiento"]
   }
  }
 },
 "required": ["numero","bloqueada","foto","id_cliente","consumos"],
 "additionalProperties": false
}
```

```
{
 "title": "Medio de entretenimiento",
 "type": "object",
 "properties": {
  "id": {"type": "integer"},
  "nombre": {"type": "string"},
  "precio": {"type": "number"},
  "importes_consumos": {
   "type": "array",
   "items": {"type": "number"}
  },
  "tipo": {
   "type": "object",
   "properties": {
    "nombre":{"type": "string", "enum": ["ATRACCION", "PARQUE", "EVENTO
       "]},
    "altura_min": {"type": "integer"},
    "edad_desde": {"type": "integer"},
    "edad_hasta": {"type": "integer"},
    "id_parque": {"type": "integer"},
    "horario_desde": {"type": "string", "format": "date-time"},
    "horario_hasta": {"type": "string", "format": "date-time"}
   },
   "required":["nombre"],
   "oneOf": [
    {   "properties": {
         "nombre": {"enum": ["ATRACCION"]}
        },
        "required": ["altura_min", "edad_desde", "edad_hasta", "
          id_parque"],
        "additionalProperties": false
    },
    {   "properties": {
         "nombre": {"enum": ["PARQUE"]}
        },
        "additionalProperties": false
    },
    {   "properties": {
         "nombre": {"enum": ["EVENTO"]}
        },
        "required": ["horario_desde","horario_hasta"],
        "additionalProperties": false
    }
   ]
  }
 },
 "required": ["id", "nombre", "precio", "importes_consumos","tipo"],
 "additionalProperties": false
```

```
}
```

```
{
 "title": "Pago",
 "type": "object",
 "properties": {
  "id": {"type": "integer"},
  "fecha": {"type": "string", "format": "date-time"},
  "medio_pago": {"type": "string"},
  "facturas": {
   "type": "array",
   "items": {
    "type": "object",
    "properties": {
     "nro_factura": {"type": "integer"},
     "consumos": {
      "type": "array",
      "items": {
       "type": "object",
       "properties": {
        "importe": {"type": "number"},
        "medio_entretenimiento": {
         "type": "object",
         "properties": {
          "id": {"type": "integer"},
          "nombre": {"type": "string"},
          "tipo": {
           "type": "object",
           "properties": {
            "nombre":{"type": "string", "enum": ["ATRACCION", "PARQUE",
               "EVENTO"]}
           },
           "required":["nombre"]
          }
         },
         "required":["id","nombre","tipo"]
        }
       },
       "required": ["importe","medio_entretenimiento"]
      }
     }
    },
    "required": ["nro_factura", "consumos"]
   }
  }
 },
 "required": ["id","fecha","medio_pago","facturas"],
 "additionalProperties": false
}
```

```
{
 "title": "Factura",
 "type": "object",
 "properties": {
  "numero": {"type": "integer"},
  "fecha_emision": {"type": "string", "format": "date-time"},
  "fecha_vencimiento": {"type": "string", "format": "date-time"},
  "id_pago": {"type": "integer"},
  "consumos": {
   "type": "array",
   "items": {
    "type": "object",
    "properties": {
     "importe": {"type": "number"},
     "medio_entretenimiento": {
      "type": "object",
      "properties": {
       "id": {"type": "integer"},
       "nombre": {"type": "string"},
       "tipo": {
        "type": "object",
        "properties": {
         "nombre":{"type": "string", "enum": ["ATRACCION", "PARQUE", "
            EVENTO"]}
        },
        "required":["nombre"]
       }
      },
      "required":["id","nombre","tipo"]
     }
    },
    "required": ["importe","medio_entretenimiento"]
   }
  }
 },
 "required":["numero","fecha_emision","fecha_vencimiento","id_pago","
    consumos"],
 "additionalProperties": false
}
```

4. Funcionalidades a implementar

Algorithm 1 Lista de atracciones visitadas por un cliente dado

INPUT: idCliente

```
docCliente ← table('cliente').get(idCliente)
for tarjeta in docCliente.tarjetas do
    for consumo in tarjeta.consumos do
        if consumo.medio_entretenimiento.tipo.nombre = 'ATRACCION' then
            result ∪ (consumo.medio_entretenimiento.id, consumo.medio_entretenimiento.nombre)
        end if
    end for
end for
return result
```

Algorithm 2 Ranking de atracciones por cantidad de visitas para un parque dado

INPUT: idParque

```
visitasXAtraccion ← emptyDictionary
for m in table('Medio_entretenimiento') do
    if m.tipo.nombre = 'ATRACCION' AND m.tipo.id_parque = idParque then
        if not visitasXAtraccion.defined(m.id) then
            visitasXAtraccion.define(m.id, (m.nombre, 1))
        else
            visitasXAtraccion.incrementarContador(m.id)
        end if
    end if
end for
return visitasXAtraccion.toList.orderedBy('contador', 'DESC')
```

Algorithm 3 Top 5 de Eventos con mayor facturación

```
facturacionXEvento ← emptyDictionary
for pago in table('pago') do
    for factura in pago.facturas do
        for c in factura.consumos do
            if c.medio_entretenimiento.tipo.nombre = 'EVENTO' then
                if not facturacionXEvento.defined(c.medio_entretenimiento.id) then
                    facturacionXEvento.define(c.medio_entretenimiento.id,
(c.medio_entretenimiento.nombre), c.importe)
                else
                    facturacionXEvento.sumarImporte(c.medio_entretenimiento.id, c.importe)
                end if
            end if
        end for
    end for
end for
return facturacionXEvento.toList.orderedBy('importe', 'DESC').takeFirst(5)
```

Algorithm 4 Total de importes por atracción en un parque dado

INPUT: idParque

```
importesXAtraccion ← emptyList
for m in table('Medio_entretenimiento') do
    if m.tipo.nombre = 'ATRACCION' AND m.tipo.id_parque = idParque then
        importesXAtraccion.append( (m.id,m.nombre,Sum(m.importes)) )
    end if
end for
return importesXAtraccion.orderedBy( 'total', 'ASC' )
```

5. Experimentación con Sharding

Descripción: Crearemos para la tabla **experimento.consumos** 3 shards y graficaremos la evolución de los shards a medida que se agregan registros. Tendremos un conjunto de 10.000 consumos y tomaremos mediciones cada 500 registros agregados. El experimento puede reproducirse utilizando el script **experimento.py**, habiendo ejecutado anteriormente **generar_datos.py**. Este encarga de cargar los datos a la base, crear los shards y graficar los resultados.

Nota: *RethinkDB* automáticamente rebalancea las tablas a medida que se incrementa el número de shards, y mientras los documentos tengan sus primary keys uniformemente distribuidas es raramente necesario un rebalanceo manual. Esto no ocurre en nuestro caso, pues los IDs generados son enteros secuenciales crecientes. Esto provoca que sea necesario ejecutar un rebalanceo manual de las tablas luego de cada inserción.

Resultados: A continuación presentamos el gráfico obtenido del experimento.

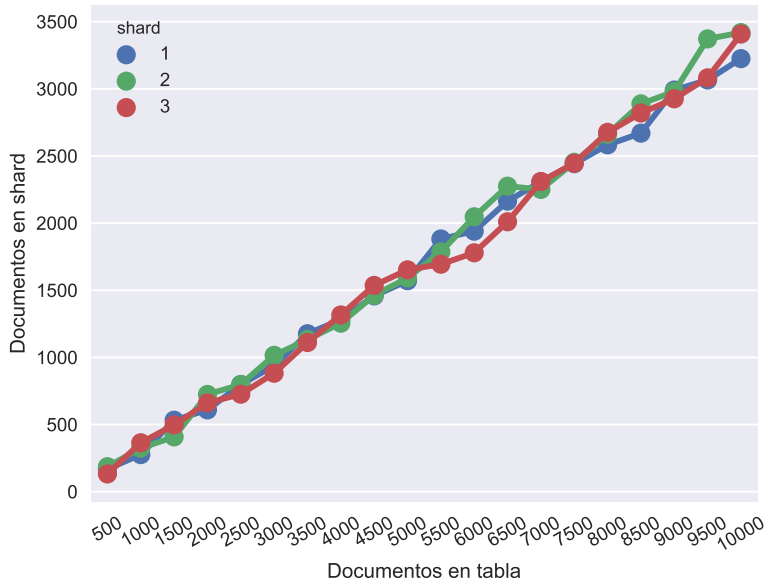


Figura 1: Experimento: documentos por shard en función de documentos totales.

Conclusiones del experimento: La idea de los shards es que particionen los datos de la base de forma balanceada entre distintos servidores. Es por esto que el gráfico se corresponde con lo que esperábamos ver. A medida que se agregan documentos a la base, estos se distribuyen de forma relativamente uniforme entre los shards disponibles, y por lo tanto la evolución de los documentos por shard es proporcionalmente lineal.

6. Generación automática de datos

Los datos utilizados para probar las queries y realizar la experimentación fueron generados automáticamente por el script de python en el archivo **generar_datos.py**. Básicamente lo que hace el script es construir diccionarios que representan a los documentos de cada entidad y luego convertirlos a formato json. Los campos obtienen valores aleatorios dentro del dominio correspondiente a cada campo. Para simular cantidades verosímiles entre las diversas entidades abordamos la siguiente elección:

- Clientes: 20
- Tarjetas: 25 (1 activa para cada cliente + 5 bloqueadas)
- Consumos: 10000 para poder experimentar cómodamente.
- Facturas: 1000, la variedad de cantidad por factura no afecta el experimento por lo cual para simplificarlo asignamos 10 consumos a cada una.
- Pagos: 1 pago por factura, es decir, 1000 pagos.
- Medios de entretenimiento: 400, de los cuales 50 son parques, 240 atracciones y 110 eventos.

7. Conclusiones

Logramos el pasaje a partir de un DER inicial al DID, tomando las decisiones sobre embeber (total o parcialmente) o referenciar de acuerdo a las consultas pedidas. Finalmente realizamos la implementación física de la base de datos NoSQL con *RethinkDB*. A su vez, utilizamos el lenguaje de consultas del motor, *ReQL*, para implementar queries que podrían ser de utilidad en una situación real. También fue necesario utilizar los drivers provistos por la base, en nuestro caso *Python*, para interactuar con los datos. Esta interacción resulta considerablemente distinta a la realizada para el primer trabajo práctico, dado que la sintaxis se asemeja más a lo que haría una aplicación que utiliza a la base que a SQL.

Por otro lado, definimos y trabajamos con documentos *JSON*, muy diferentes a a la forma de almacenar datos de las tradicionales bases SQL. Las bases de datos NoSQL, al ser mucho más abiertas y flexibles, nos permitieron adaptarnos a nuestras necesidades mucho más fácilmente que los modelos relacionales. De esta forma las queries que debíamos realizar se volvieron considerablemente más sencillas de implementar y con una muy buena performance. Por otro lado esta ductilidad que las bases NoSQL presentan, hace que mantener su consistencia pueda ser más complejo que en las SQL, y que a veces no sea claro el formato exacto de un documento.