Trabajo Práctico 1: Mercado Virtual

Base de Datos

Primer Cuatrimestre de 2016

Integrantes:

Amil, Diego - 68/09 - amildie@gmail.com
Garrone, Javier - 151/10 - javier3653@gmail.com
Langberg, Martín - 86/10 - martinlangberg@gmail.com
Truffat, Ignacio - 837/10 - itruffat@dc.uba.ar

Fecha de entrega: 06/05/2016

Tabla de Contenidos

Introducción

El objetivo de este trabajo fue implementar una solución a un problema del “mundo real” (en este caso, un modelo de e-commerce similar a Mercadolibre) en el que pudiéramos usar algún motor de base de datos relacional, así como también describir la solución previamente usando los modelos vistos en clase.

Para implementar la base de datos usamos la herramienta SQLite. SQLite fue diseñado para ser simple y compacto, por lo cual carece de ciertos features avanzados que motores más complejos tienen, como los *Stored Procedures*.

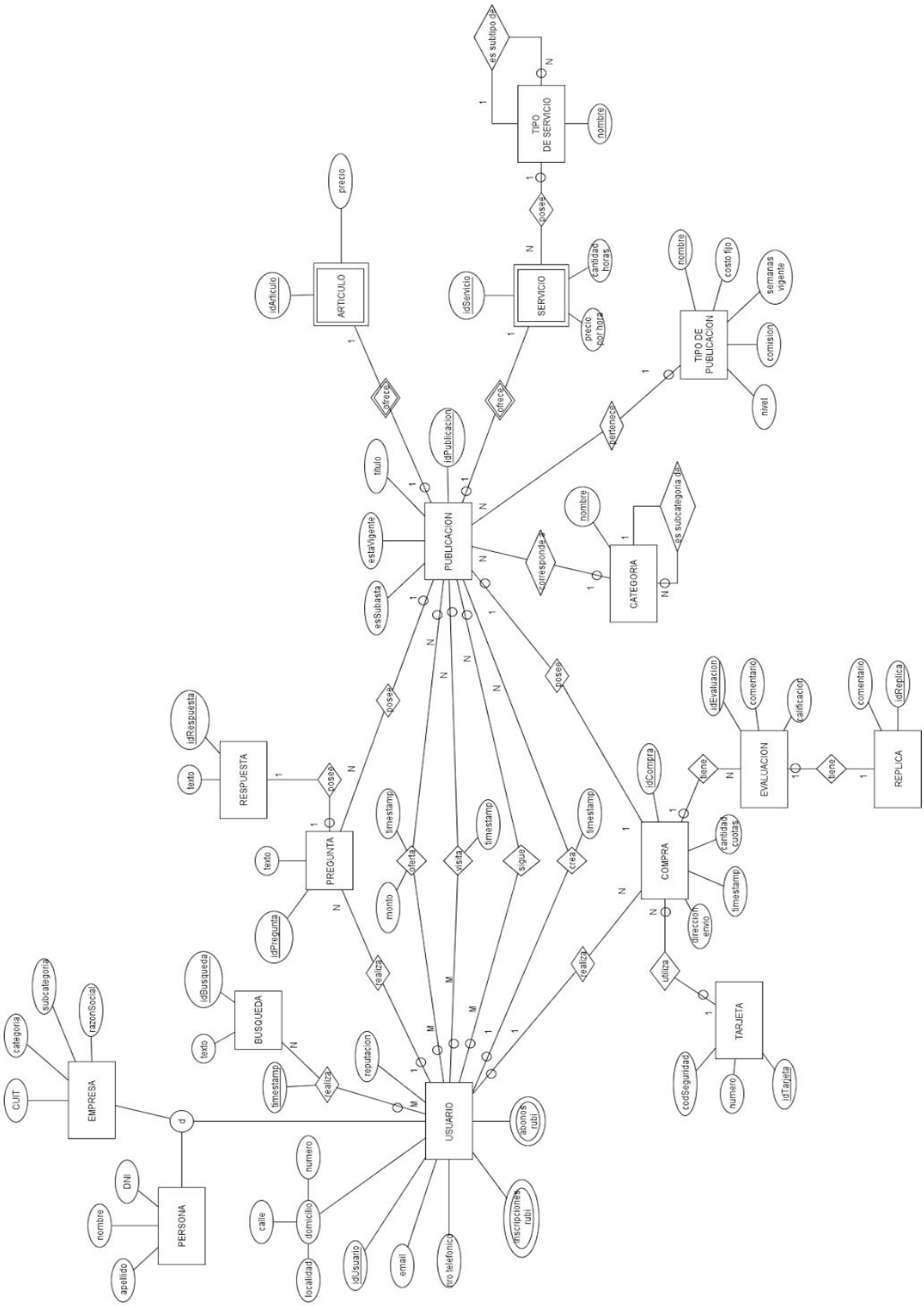
Un *Stored Procedure* (SP) es una rutina almacenada en la base de datos (DB) que provee alguna funcionalidad que se desea repetir, ya sea para ser llamada durante un query o para otras funciones del motor (como por ejemplo, durante triggers). En cierto sentido, puede pensarse como una funciones de un lenguaje imperativo a ser llamadas.

La ventaja principal de este feature es que "pega" parte de la lógica del proyecto a la DB, estando siempre disponible para todo programador que tenga acceso a ella. Además, si llega a haber algún cambio en las necesidades del proyecto (por ejemplo, cambia una norma interna y las personas que gastaron menos de 100 U$S pasan a ser "no prioritarias"), ajustar un StoredProcedure asegura que ese cambio se ve reflejado en cualquier llamado al subproceso sin tener que hacer ninguna clase de mantenimiento. Por último (y esto es más dependiente de cada motor), existen motores de DB que optimizan los SP almacenados para mejorar su eficiencia.

Suposiciones

- Una publicación “termina” al ser comprada. No se puede comprar una publicación varias veces. Además posee un único comprador.
- Una compra posee como máximo 2 calificaciones: 1 hecha por el comprador y 1 hecha por el vendedor.
- Las publicaciones de servicio poseen una cantidad fija de horas ofrecida.
- Ningún artículo puede repetirse en publicaciones.
- La cantidad de publicaciones RubiDeOriente que se puede realizar por mes es fija (3) y no es necesario almacenarla en la db.
- Hay 1 solo tipo de membresía (RubiDeOriente).
- La vigencia de una subasta es la misma que la de una venta.

Modelo Entidad-Relación



Modelo Relacional derivado

```
USUARIO(idUsuario, nroTelefónico, email, calle, número, localidad,
inscripcionesRubi, abonosRubi)
PK = {idUsuario}, FK = {}

PERSONA(idUsuario, apellido, nombre, DNI)
PK = {idUsuario}
FK = {idUsuario -> USUARIO.idUsuario}

EMPRESA(idUsuario, CUIT, categoría, subcategoría, razónSocial)
PK = {idUsuario}
FK = {idUsuario -> USUARIO.idUsuario}

BÚSQUEDA(idBúsqueda, texto)
PK = {idBúsqueda}, FK = {}

REALIZA_BÚSQUEDA(idBúsqueda, idUsuario, timestamp)
PK = {(idBúsqueda, idUsuario, timestamp)}
FK = {idBúsqueda -> BÚSQUEDA.idBúsqueda, idUsuario -> USUARIO.idUsuario}

PREGUNTA(idPregunta, idUsuario, idPublicacion, texto)
PK = {idPregunta}
FK = {idUsuario -> USUARIO.idUsuario, idPublicacion ->
PUBLICACION.idPublicacion}

RESPUESTA(idRespuesta,  texto, idPregunta)
PK = {idRespuesta}
FK = {idPregunta -> PREGUNTA.idPregunta}

OFERTA(idUsuario, idPublicacion, timestamp, monto)
PK = {(idUsuario, idPublicacion, timestamp)}
FK = {idUsuario -> USUARIO.idUsuario, idPublicacion ->
PUBLICACION.idPublicacion}

VISITA(idUsuario, idPublicacion, timestamp)
PK = {(idUsuario, idPublicacion, timestamp)}
FK = {idUsuario -> USUARIO.idUsuario, idPublicacion ->
PUBLICACION.idPublicacion}

SIGUE(idUsuario, idPublicacion)
PK = {(idUsuario, idPublicacion)}
FK = {idUsuario -> USUARIO.idUsuario, idPublicacion ->
PUBLICACION.idPublicacion}
```

PUBLICACIÓN(idPublicación, título, esSubasta, estaVigente, nombreCategoria, nombreTipo, idUsuarioCreador, **timestamp**)
PK = {idPublicación}
FK = {nombreCategoria -> CATEGORIA.nombre, nombreTipo -> TIPO_DE_PUBLICACION.nombre, idUsuarioCreador -> USUARIO.idUsuario}

CATEGORÍA(nombre, superCategoria)
PK = {nombre}
FK = {superCategoria -> CATEGORIA.nombre}

ARTICULO(idArticulo, idPublicacion, precio, nombre)
PK = {(idArticulo, idPublicacion)}
FK = {idPublicacion -> PUBLICACION.idPublicacion}

SERVICIO(idServicio, idPublicacion, precioPorHora, cantidadDeHoras, tipo)
PK = {(idServicio, idPublicacion)}
FK = {idPublicacion -> PUBLICACION.idPublicacion, tipo -> TIPO_DE_SERVICIO.nombre}

TIPO_DE_SERVICIO(nombre, nombreSupertipo)
PK = {nombre}
FK = {nombreSupertipo -> TIPO_DE_SERVICIO.nombre}

TIPO_DE_PUBLICACIÓN(nombre, costoFijo, semanasVigente, comisión, nivel)
PK = {nombre}, FK = {}

COMPRA(idCompra, timestamp, cantidadCuotas, direcciónEnvío, idTarjeta, idUsuario, idPublicacion)
PK = {idCompra}
FK = {idTarjeta -> TARJETA.idTarjeta o NULL, idUsuario -> USUARIO.idUsuario, idPublicacion -> PUBLICACION.idPublicacion}

TARJETA(idTarjeta, numero, **idUsuario**)
PK = {idTarjeta}
FK = {idUsuario -> USUARIO.idUsuario}

EVALUACIÓN(idEvaluacion, comentario, calificacion, idCompra)
PK = {idEvaluacion}
FK = {idCompra -> COMPRA.idCompra}

RÉPLICA(idReplica, comentario, idEvaluacion)
PK = {idReplica}
FK = {idEvaluacion -> EVALUACION.idEvaluacion}

Restricciones en lenguaje natural

- Un usuario no puede realizar acciones (seguir, preguntar, etc) sobre una publicación no vigente.
- Un usuario no puede inscribirse nuevamente a “RubíDeOriente” si la fecha de su último abono sigue vigente, o viceversa.
- Un usuario no puede crear una publicación nivel “RubíDeOriente” si no posee una inscripciónRubí con abono vigente (30 días desde su pago).
- Un usuario no puede crear más de 3 publicaciones nivel “RubíDeOriente” durante la vigencia de su último abono.
- Un usuario no puede poseer más de un abono vigente.
- Si el campo “direcciónEnvío” en la tabla COMPRA posee valor NULL, significa que el retiro es de forma personal.
- Una publicación debe tener necesariamente un artículo o un servicio (o combinación de ambos), no puede haber publicaciones “vacías”.
- Un usuario no puede ofertar por una publicación que NO es subasta, y no puede comprar una publicación que lo sea.
- Una publicación está vigente la cantidad de semanas que indica su nivel a partir del momento de su creación
- Con respecto a las categorías y a los tipos de servicio: ningún subtipo/subcategoría puede tener como tipo/categoría principal uno de mayor “profundidad” al que posea. Considerando al primer tipo/categoría como profundidad 0, y sumando uno a la profundidad de sus subtipos/subcategorías correspondientes.

Código Fuente

Consulta_info_usuario_categorias_mas_frecuentes.sql

```
--Consulta por usuario: obtener, para un usuario específico, las primeras 3 categorías
--de artículos que visitó con mayor frecuencia en el último año.

SELECT P.nombreCategoria AS Categoria, COUNT(P.nombreCategoria) AS Visitas
FROM VISITA AS V
     INNER JOIN PUBLICACION AS P ON V.[idPublicacion] = P.[idPublicacion]
WHERE V.[idUsuario] = 'pepito' AND strftime('%Y', timestamp) = strftime('%Y', CURRENT_DATE)
GROUP BY P.nombreCategoria
ORDER BY Visitas DESC LIMIT 3
```

Consulta_por_categoria_de_producto.sql

```
--Consulta por categoría de producto: obtener, dada una categoría de producto
("Computación","Hogar, muebles y jardín", etc), un listado de los vendedores que han
publicado artículos de dicha categoría y la cantidad de ventas que efectuó cada uno de
dichos vendedores.

SELECT P.[idUsuarioCreador] AS Usuario, COUNT(C.[idCompra]) AS 'Cantidad de ventas'
FROM PUBLICACION AS P
     INNER JOIN COMPRA AS C ON P.[idPublicacion] = C.[idPublicacion]
WHERE C.[idUsuario] IN (
  SELECT DISTINCT P2.idUsuarioCreador
  FROM PUBLICACION AS P2
  WHERE P2.[nombreCategoria] = 'Hogar'
)
GROUP BY P.[idUsuarioCreador]
```

Consulta_info_usuario.sql

```
--Consulta por usuario: obtener, para un usuario específico, información sobre los artículos que ha comprado y vendido, los artículos que ha visitado con su fecha de visita, los artículos que tiene en su lista de favoritos.

SELECT * FROM (

SELECT 'Compra' AS 'Accion', A.[nombre] as Nombre, A.[precio] as Precio, C.[timestamp] AS Fecha
FROM COMPRA AS C
     INNER JOIN PUBLICACION AS P ON C.[idPublicacion] = P.[idPublicacion]
     INNER JOIN ARTICULO AS A ON P.[idPublicacion] = A.[idPublicacion]
WHERE C.[idUsuario] = 'pepito'

UNION ALL

SELECT 'Venta' AS 'Accion', A.[nombre] as Nombre, A.[precio] as Precio, C.[timestamp] AS Fecha
FROM COMPRA AS C
     INNER JOIN PUBLICACION AS P ON C.[idPublicacion] = P.[idPublicacion]
     INNER JOIN ARTICULO AS A ON P.[idPublicacion] = A.[idPublicacion]
WHERE P.[idUsuarioCreador] = 'pepito'

UNION ALL

SELECT 'Visita' AS 'Accion', A.[nombre] as Nombre, A.[precio] as Precio, V.[timestamp] AS Fecha
FROM VISITA AS V
     INNER JOIN PUBLICACION AS P ON V.[idPublicacion] = P.[idPublicacion]
     INNER JOIN ARTICULO AS A ON P.[idPublicacion] = A.[idPublicacion]
WHERE V.[idUsuario] = 'pepito'

UNION ALL

SELECT 'Favorito' AS 'Accion', A.[nombre] as Nombre, A.[precio] as Precio, NULL AS Fecha
FROM SIGUE AS S
     INNER JOIN PUBLICACION AS P ON S.[idPublicacion] = P.[idPublicacion]
     INNER JOIN ARTICULO AS A ON P.[idPublicacion] = A.[idPublicacion]
WHERE S.[idUsuario] = 'pepito'


)

ORDER BY Fecha
```

Consulta_por_ganadores.sql

```
--Consulta por ganador/es anual de viaje a Khan El-Khalili: obtener, para un año
específico, el ganador/es.

SELECT idUsuario, SUM(Monto) AS Total FROM (

  SELECT C.[idUsuario], COALESCE(SUM(A.precio), 0) + COALESCE(SUM(S.[precioPorHora] *
S.[cantidadDeHoras]), 0) AS Monto
  FROM COMPRA AS C
       INNER JOIN PUBLICACION AS P ON P.[idPublicacion] = C.[idPublicacion]
       LEFT JOIN ARTICULO AS A ON A.[idPublicacion] = P.[idPublicacion]
       LEFT JOIN SERVICIO AS S ON S.[idPublicacion] = P.[idPublicacion]
  WHERE strftime('%Y', C.[timestamp]) = '2016'
  GROUP BY C.[idUsuario]

  UNION ALL

  SELECT O.[idUsuario], COALESCE(SUM(O.monto), 0) AS Monto
  FROM OFERTA AS O
       INNER JOIN PUBLICACION AS P ON P.[idPublicacion] = O.[idPublicacion]
  WHERE P.[estaVigente] = 0 AND strftime('%Y', O.[timestamp]) = 2016 AND
        O.[timestamp] = (
        SELECT MAX(O2.timestamp)
        FROM OFERTA AS O2
        WHERE O2.[idPublicacion] = P.[idPublicacion]
              AND strftime('%Y', O2.[timestamp]) = '2016'
  )
  GROUP BY O.[idUsuario]
)
GROUP BY idUsuario
ORDER BY Total DESC
LIMIT 1
```

Consulta_por_keyword.sql

```
--Consulta por keyword: obtener, para un cierto keyword (por ejemplo "mesa"), la lista de
publicaciones
--vigentes que tengan en el título, dicha keyword. El usuario debe poder restringir su
--búsqueda sólo a cierta categoría de artículos o servicios.

SELECT idPublicacion, titulo
FROM PUBLICACION
WHERE titulo LIKE '%escoba%' AND nombreCategoria = ? -- ej: Hogar
```

Consulta_por_usuario_y_preguntas.sql

```
--Consulta por usuario y preguntas: obtener para un usuario específico, la lista de
preguntas que ha realizado, con las respectivas respuestas que haya recibido (sólo la
pregunta, si aún no recibió respuesta).

SELECT P.texto AS Pregunta, R.texto AS Respuesta
FROM PREGUNTA AS P LEFT JOIN RESPUESTA AS R ON P.idPregunta = R.idPregunta
WHERE idUsuario = ?
```

Ofertar.py

```
import sqlite3
import time
import datetime

con = sqlite3.connect('TP1.db')
c = con.cursor()
numero_subasta = int(raw_input("Numero de subasta: "))
query_hay_subasta = 'SELECT * FROM PUBLICACION p WHERE p.idPublicacion = ' +
str(numero_subasta) + " AND p.esSubasta = 1"
if(len(c.execute(query_hay_subasta).fetchall()) == 0):
    print("ERROR : NO EXISTE ESA SUBASTA")
else:
    usuario = str(raw_input("Nombre de Usuario: "))
    query_hay_usuario = "SELECT * FROM PERSONA p WHERE p.idUsuario = '" + usuario + "'"
    if(len(c.execute(query_hay_usuario).fetchall()) == 0):
        print("ERROR : NO EXISTE ESE USUARIO")
    else:
        valor_a_escribir = float(raw_input("Monto a escribir: "))
        query_elegir_valor_actual = 'SELECT r.monto, r.idUsuario FROM OFERTA r WHERE
r.idPublicacion = ' + str(numero_subasta) + " ORDER BY r.monto DESC"
        entrar = False
        if(len(c.execute(query_elegir_valor_actual).fetchall()) == 0):
            usuario_actual = ""
            entrar = (valor_a_escribir > 0)
        else:
            valor_actual = c.execute(query_elegir_valor_actual).fetchone()[0]
            usuario_actual = c.execute(query_elegir_valor_actual).fetchone()[1]
            entrar = (valor_a_escribir >= valor_actual) & (valor_a_escribir < (valor_actual
* 2))
        if(entrar):
            if(usuario_actual == usuario):
                print("ERROR : ESTA OFERTANDO EL MISMO USUARIO")
            else:
                timestampstr =
datetime.datetime.fromtimestamp(time.time()).strftime('%Y-%m-%d %H:%M:%S')
                query_insertar_valor_a_escribir = "INSERT INTO OFERTA (idUsuario,
idPublicacion, timestamp, monto) VALUES ( '" + usuario + "', " + str(numero_subasta) + ",
'" + timestampstr + "' , " + str(valor_a_escribir) + ")"
                c.execute(query_insertar_valor_a_escribir)
                con.commit()
        else:
            print("ERROR : VALOR DE INSERCION INVALIDO")
```

Conclusiones

Este trabajo práctico nos permitió ver al Diagrama de Entidad- Relación como una herramienta, ya que nos facilitó la construcción del diseño de la base de datos a través del Modelo Relacional como intermediario. A diferencia de los ejercicios de la práctica pudimos desarrollar el Modelo Relacional y la implementación de la base de datos viendo el proceso completo de diseño.
Por otro lado, las iteraciones nos permitieron ir puliendo el DER a través de la interacción
con el tutor, una experiencia más similar al de un diseñador de una base de datos.

Una de las limitaciones que encontramos al usar SQLite como manager de la base fue la implementación de funciones, característica no soportada por el sistema, por lo que fueron escritas en lenguaje Python.
De todas formas, consideramos que usar una función definida de forma externa puede proveer resultados satisfactorios a pesar de ser más limitado. Mientras se tengan múltiples cuidados (estas funciones están siempre disponibles y up-to-date para todos los usuarios, los programadores evitan crear código si existe una tarea externa que cumpla ese rol a la cual puedan llamar, etc.) consideramos que la consistencia de la DB no peligra por tener ese código menos "pegado".
Esto nos permitió a su vez entender las principales limitaciones de este manager con respecto a este tipo de usos, de mayor utilidad en aplicaciones que corren de manera local sin muchas escrituras simultáneas, y con una cantidad de datos relativamente reducida[1].

Correcciones en el MER

- La relación “crea” (1:N) entre las entidades USUARIO y PUBLICACIÓN poseía un atributo timestamp. Dado que en la materia solo se admiten atributos para relaciones (N:M), se movió el atributo a la entidad PUBLICACIÓN.
- Se le agregó el atributo “nombre” a la entidad ARTÍCULO en el diagrama (se había tenido en cuenta para el MR y para la DB, pero faltaba en el DER).
- Se vinculó a las entidades USUARIO y TARJETA por medio de la relación “registra”. Por consiguiente, se agregó una FK “idUsuario” a la tabla TARJETA.
- Se quitó el atributo “codSeguridad” a la entidad TARJETA. Se almacena únicamente el número de la tarjeta.

Bibliografía

[1] http://www.sqlite.org/whentouse.html
[2] García Molina/Ullman/Widom - Database Systems: The Complete Book, Prentice Hall, 2nd Edition, 2009
[3] https://www.dc.uba.ar/materias/bd/2016/c1/descargas/apuntes/ApunteSQL
[4] https://www.dc.uba.ar/materias/bd/2016/c1/descargas/apuntes/modelizacion