Pontificia Universidad Católica de Chile
Facultad de Ingeniería
Departamento de Ciencias de la Computación
Primer Semestre 2020
Docente: Adrián Soto - Juan Reutter

Entrega Proyecto N°2

Nicolás Ignacio Sumonte Fuenzalida - Jorge Ignacio Toro Castro

Grupo N° 85

1. El proyecto

Durante el año 2020 se celebran 500 años de la muerte de Raffaello Sanzio, conocido simplemente como Rafael, quien es uno de los más célebres pintores y arquitectos del renacimiento. Gente alrededor de todo el mundo está planeando hacer viajes para conocer personalmente toda la obra de este artista. Por lo mismo, la empresa de viajes Splinter S.A. ha decidido comenzar a vender paquetes de turismo para conocer toda la obra de este artista. Ha sido tal el éxito, que están convencidos de que deben abrir un sitio en línea que permita a sus clientes armar viajes que permitan visitar la obra de varios artistas de distintos periodos. El objetivo fínal es tener una página que permita buscar cierto artista y comprar inmediatamente un pack que permita ver toda (o lo más importante de) la obra del artista. Para eso han contratado a nosotros, encargados de desarrollar el modelo de datos y una primera versión de la página.

2. Base de datos Splinter S.A

Los usuarios deberán poder comprar tickets de los posíbles viajes, reservar hoteles, escoger la forma de transportarse, etc. Para ello Springer S.A tiene las siguientes entidades tentativas listadas a continuacion:

- **usuarios**(uid, nombreusuario , username , correo, dirección)
- **tickets**(tid, uid , asiento , fechaviaje, fechacompra)
- **reservas**(rid, uid , hid , fecha_llegada, fecha_salida)
- **paises**(pid, nombre_pais,telefono)
- **Medio**(mid, medio, capacidad)
- **hoteles**(hid, cid , nombrehotel, dirección , teléfono, precio)
- **destinos**(did, cid_ciudad_origen , cid_ciudad_destino , mid, precio, horadesalida)
- **ciudades**(cid, ciudad , pid)

nosotros le hemos propuesto a Springer S.A trabajar con el siguiente esquema Entidad Relación, que incluye la entidades pedidas:

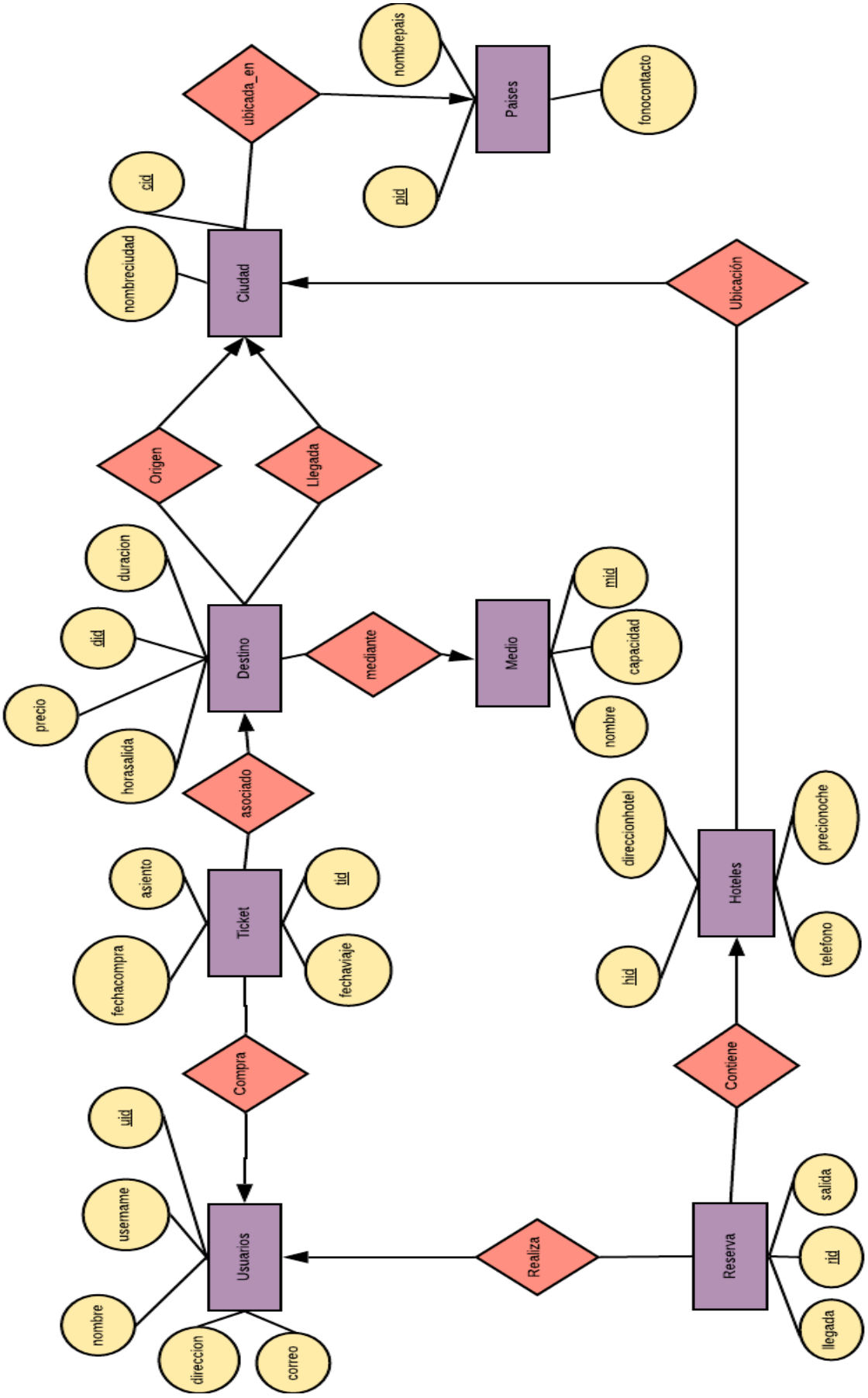

Figura 1: Diagrama Entidad/Relación, Springer S.A

De esta forma es posible construir un esquema relacional que incluya el dato de cada atributo, asi como también las `Primary Keys Y las Foreign Keys.`

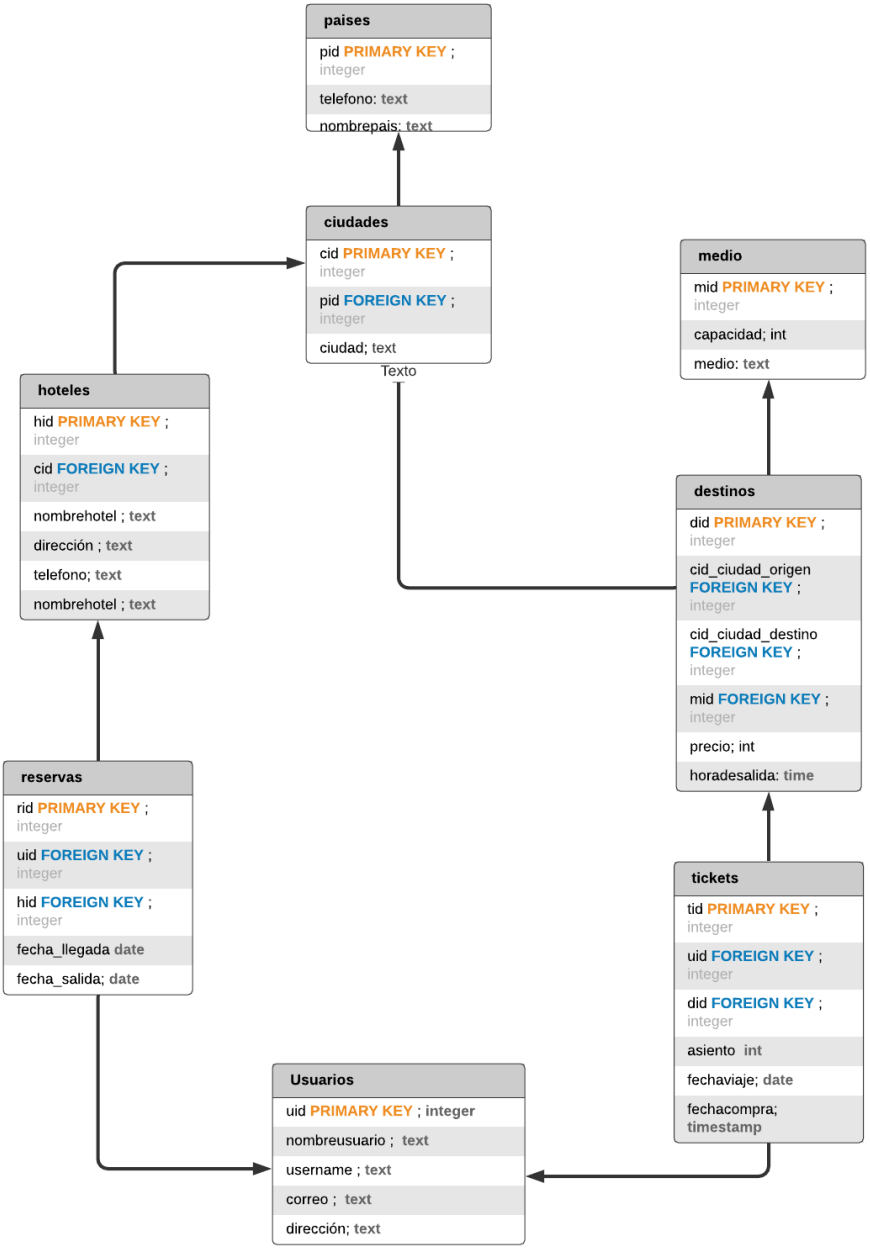

Figura 2: Diagrama de Entidades Sprinter S.A.

3. justificación Base De Datos

A continuación, justificaremos porqué este modelo se ajusta al BCNF o 3NF para que así, Springer S.A no tenga problemas durante la ejecución de su Base de Datos.

- **usuarios**(uid:int(PK), nombreusuario:text , username:text , correo:text, dirección:text): En este caso (uid, username) son una superllave, por lo que pueden determinar todos los demás atributos, y como define BCNF esto puede ocurrir y está justificado como normalizado. Por otro lado, decidimos elegir uid como PK ya que la eficiencia en la búsqueda es mayor, si bien en nuestra base de datos la diferencia es minima, si se trabajase con bases de datos mayores si harían la diferencia.

  uid, username → nombre, direccion, correo

- **tickets**(tid:int(PK), uid:text(FK) , asiento:int , fechaviaje:date, fechacompra:timestamp): En este caso, está normalizado ya que presenta una única dependencia, debido a que las fechas de compras pueden variar y ningún atributo depende funcionalmente de otro. Además se justifica la creación de una PK debido a que pueden existir multiples destinos que tienen asignados distintos asientos, y de esa manera no se pierden datos. Por ultimo, se relaciona con dos FK ya que como la relación que tiene con usuarios es 1-N y la que tiene con destinos es 1-N no era necesario crear tablas que los relacionen.

  tid → uid, did, asiento, fechaviaje, fechacompra

- **reservas**(rid:int(PK), uid:int(FK) , hid;int(FK) , fecha_llegada:Date, fecha_salida: Date): La dependencia en esta tabla es única, ya que un usuario puede tener multiples reservas en un hotel con diferentes llegadas y salidas, por lo tanto no se presenta dependencia. Por otro lado, dado las relaciones 1-N que tiene con usuario y hoteles, no fue necesario crear una tablas que los relacionen.

  rid → uid, hid, llegada , salida

- **paises**(pid:int (PK), nombre_pais:text, telefono:text): En este caso hay una dependencia transitiva, debido a que no hay dos paises en el mundo con el mismo nombre, pero debido a que no es el objetivo del proyecto, se puede justificar que una PK con valor int es más eficiente que una con valor texto.

  pid → nombrepais , fonocontacto.

  nombrepais → fonocontacto.

- **Medio**(mid:int (PK), medio:text, capacidad:int): En este caso, un medio solo puede ser definido por un mid, ya que para distintos tipos de transporte pueden haber distintas capacidadesy estos atributos no dependen transitivamente entre ellos, solo del mid

  mid → nombre, capacidad

- **hoteles**(hid:int(PK), cid:int(FK) , nombrehotel:text, dirección:text , teléfono:text, precio:int): En este caso, no podemos decir que la dirección de un hotel sería una llave ya que no se puede asumir que las direcciones son únicas en el mundo, por otro lado un numero de telefono no es suficiente, por lo tanto se creó la PK con tal de no perder tuplas en caso de que los demás atributos se repitan y sean restrictivos.

  hid → cid, direccionhotel, telefono, precionoche

- **destinos**(did:int (PK), cid_ciudad_origen:int(FK) , cid_ciudad_destino:int(FK) ,

`mid:int (FK), precio:int, horadesalida: time)`:En este caso, podemos decir que pueden existir entre dos destinos, diferentes viajes que varíen en hora de salida, duración o el precio, por lo tanto estos atributos no dependen transitivamente entre ellos, si no que solo del did. Además, pueden existir diferentes viajes entre ciudad de origen y ciudad de destino que no dependen entre sí, si no que dependen de la PK.

$$\text{did} \rightarrow \text{cid\_ciudadorigen, cid\_ciudadllegada, horasalida, duración, precio}$$

- **`ciudades`**(`cid:int (PK), ciudad:text , pid:int (FK))`: Los nombres de las ciudades se puede repetir, por lo tanto creamos solo una dependencia, debido a que el nombre no depende del país dado que existen múltiples ciudades por paises.

$$\text{cid} \rightarrow \text{pid, nombreciudad}$$

4. Consultas

A continuacion mostramos todas las consultas pedidas:

Consulta 1

Muestre todos los username junto a su correo.

Código

```
SELECT username, correo
    FROM usuarios;
```

Consulta 2

Ingrese el nombre de un país. Muestre todos los nombres de las ciudades del país con ese nombre en su base de datos.

Código

```
SELECT ciudades.ciudad
  FROM ciudades,paises
 WHERE ciudades.pid = paises.pid
AND nombre_pais LIKE LOWER('%pais_elegido%');
```

Consulta 3

Ingrese un username. Muestre todos los nombres distintos de países en los que ha hospedado el usuario con ese username mediante hoteles de la agencia.

Código

```
SELECT distinct paises.nombre_pais
FROM usuarios,reservas,hoteles,ciudades,paises
WHERE usuarios.uid = reservas.uid
AND hoteles.hid = reservas.hid
AND ciudades.cid = hoteles.cid
AND ciudades.pid = paises.pid
AND reservas.fecha_salida <current_date
AND username LIKE LOWER('%username%');
```

Consulta 4

Ingrese el identificador de un usuario. Muestre la cantidad de dinero que ha gastado el usuario con ese identificador en todos los tickets que ha comprado.

Código

```
SELECT sum(destinos.precio)
FROM usuarios,tickets,destinos
WHERE usuarios.uid = tickets.uid
AND tickets.did = destinos.did
AND usuarios.uid = id_usuario;
```

Consulta 5

Entregue el identificador y nombre de usuario junto a la fecha de inicio en formato YYYY-MM-DD, la fecha de término y el nombre del hotel de las reservas que parten desde el 01 de enero del 2020 y terminan antes del 31 de marzo del 2020, ambas fechas inclusive.

Código

```
SELECT usuarios.uid, usuarios.username, hoteles.nombrehotel,
reservas.fecha_llegada, reservas.fecha_salida
FROM usuarios,hoteles,reservas
WHERE usuarios.uid = reservas.uid
AND reservas.hid = hoteles.hid
AND fecha_llegada >= '2020-01-01' AND fecha_salida <'2020-04-01';
```

Pregunta 6

Ingrese dos fechas en formato YYYY-MM-DD: una de inicio y una de fin. Muestre el id, nombre de usuario y el total de dinero gastado en tickets entre esas dos fechas, ambas inclusive.

Código

```
SELECT usuarios.uid,usuarios.nombreusuario, sum(destinos.precio)
                FROM usuarios,tickets,destinos
               WHERE usuarios.uid = tickets.uid
                 AND tickets.did = destinos.did
               AND fechacompra >= '$fecha_inicio'
  AND fechacompra <= '$fecha_termino' GROUP BY usuarios.uid ;
```

Link Biblioteca de Consultas: http://codd.ing.puc.cl/~grupo85/index.php?