Modelamiento y Normalización de datos (Parte II)

El modelo físico y el caso N a N

Competencias

- Crear un modelo físico basado en un modelo lógico para la construcción de tablas en la base de datos.
- Crear tablas en base a un modelo físico para la construcción de una base de datos consistente.

Introducción

En este capítulo aprenderás el proceso que se recomienda cumplir para traspasar un modelo físico a SQL, es decir pasar el diagrama de un caso planteado a un motor de bases de datos y así poder posteriormente trabajarlo dentro de un software.

Además te presentaré el famoso caso de relaciones N a N, el cual genera comúnmente duplicidad y tendencia de almacenar datos de forma redundante en nuestras bases de datos. Aprenderás cómo evitar este caso por medio de las 3 formas normales y la selección de los campos únicos e identificadores que correspondan a las claves primarias en las entidades mayormente fuertes.

Modelo físico

El modelo físico es el responsable de representar cómo se construirá finalmente el modelo en nuestra base de datos. Incluye la estructuras de las tablas, definiendo cada uno de sus atributos y tipo de dato para cada atributo, las restricciones de las columnas, las claves primarias, claves foráneas y las relaciones entre las tablas.

En síntesis, el modelo físico es la versión final del modelo, lo puedes interpretar como el modelo lógico con los metadatos declarativos de cada atributo.

Traspasando el modelo físico a SQL

Veamos un ejemplo: Una compañía telefónica designa un departamento para recibir llamadas de los usuarios que presenten problemas con la señal en sus teléfonos o la plataforma online, y necesita almacenar todos los reportes como registros en la base de datos. Tenemos el siguiente modelo:

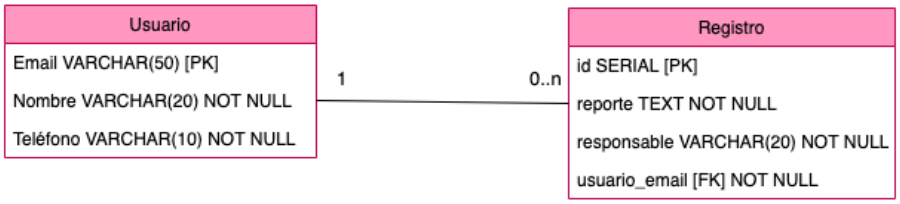


Imagen 1. Modelo Físico.

Como puedes ver en la imagen 1, con el modelo físico podemos pasar a construir nuestras tablas y agregar los tipos de datos. En el siguiente código verás el ejemplo de cómo se construirían estas tablas con consultas SQL.

```
CREATE TABLE usuario
  (
     email    VARCHAR2(50),
     nombre   VARCHAR2(20) NOT NULL,
     telefono VARCHAR2(15) NOT NULL,
     PRIMARY KEY (email)
  );

CREATE TABLE registro
  (
     id            NUMBER,
     reporte       VARCHAR2(500) NOT NULL,
     responsable   VARCHAR2(50) NOT NULL,
     usuario_email VARCHAR2(50) REFERENCES usuario (email),
     PRIMARY KEY (id)
  );
```

Podemos ver la estructura con DBeaver:

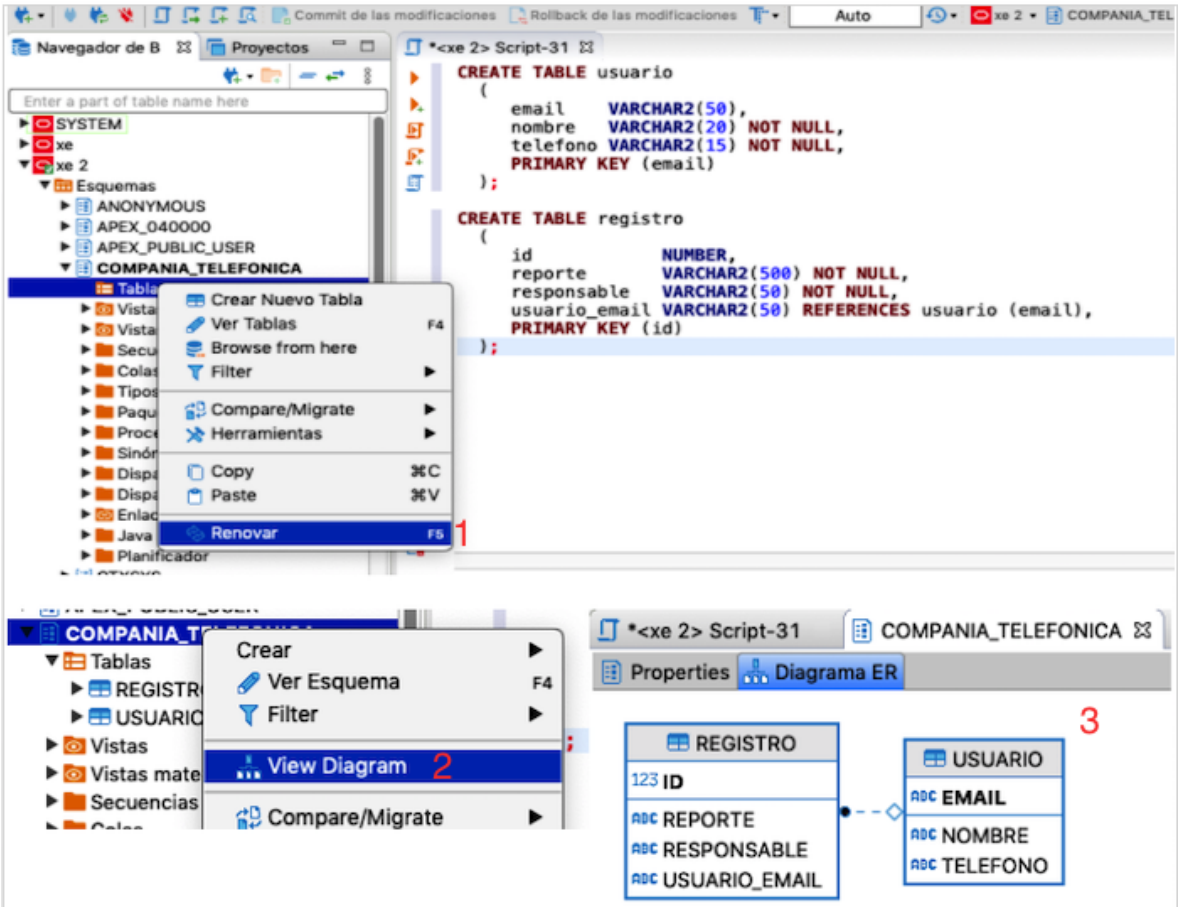


Imagen 2. Generar diagrama con DBeaver.

Al traspasar los datos a SQL nos damos cuenta que nuestro modelo físico tiene toda la información necesaria. Existen programas que nos permiten generar SQL a partir de modelos físicos y otros que nos permiten generar los diagramas a partir de SQL, hoy día es normal encontrarlos como aplicaciones web.

Si ejecutamos el código de ejemplo anterior dentro de una base de datos finalmente podremos agregar los datos para probar nuestro modelo.

```
INSERT INTO usuario (email, nombre, telefono)
VALUES      ('usuario1@gmail.com',
             'Juan',
             '12345678');

INSERT INTO usuario (email, nombre, telefono)
VALUES      ('usuario2@gmail.com',
             'Francisca',
             '12345679');

INSERT INTO registro (id, reporte, responsable, usuario_email)
VALUES      ( 1,
      'El usuario presenta problemas para realizar el pago online',
              'Javiera',
              'usuario1@gmail.com' );

INSERT INTO registro (id, reporte, responsable, usuario_email)
VALUES      (2,
             'El usuario  presenta problemas para ingresar a la plataforma',
             'Javiera',
             'usuario2@gmail.com');
```

Luego si queremos seleccionar todos los registros con sus usuarios realizaremos un join entre ambas tablas con el siguiente código.

```
SELECT *
FROM   registro
       JOIN usuario
         ON usuario.email = registro.usuario_email;
```

Obteniendo la siguiente tabla de la siguiente imagen.

id	reporte	responsable	usuario_email	email	nombre	telefono
1	El usuario presenta problemas para realizar el pago online	Javiera	usuario1@gmail.com	usuario1@gmail.com	Juan	12345678
2	El usuario presenta problemas para ingresar a la plataforma	Javiera	usuario1@gmail.com	usuario1@gmail.com	Francisca	12345679

Imagen 2. Tabla resultado de la creación de tablas en la base de datos a partir de un modelo físico.

Ahora te toca a ti (1)

Crear las tablas correspondientes al modelo físico de la siguiente imagen, corresponde a una plataforma que ofrece reseña de películas. Este modelo tiende a tener registros duplicados y redundancia de datos, es decir, no está normalizado pero lo estaremos normalizando en próximos ejercicios propuestos.

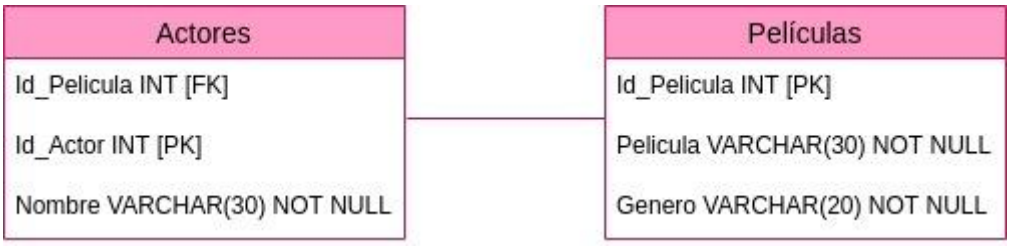


Imagen 3. Modelo físico ejercicio propuesto.

Caso con relaciones N a N

Un caso muy común al momento de modelar consiste en las relaciones N a N. Estudiemos el siguiente caso para entenderlo:

Un empleado puede trabajar en diversos proyectos de una empresa, los empleados para entrar a la plataforma necesitan identificarse con un correo corporativo, password y su nombre. De cada proyecto se tiene el nombre y una descripción.

Se pueden dar situaciones donde en un proyecto no trabaje ningún empleado y un empleado no trabaje en ningún proyecto. Procedamos con la modelación de este caso:

Modelo conceptual

Recordemos los pasos importantes:

1. **Identificar entidades:** En este caso empleado y proyecto.

2. **Agrupar entidades con sus atributos:**

    a. `Empleado (email, password, nombre).`
    b. `Proyecto (nombre, descripción).`

3. **Identificar relaciones y sus cardinalidades:**

    Un empleado puede trabajar en 0 o N proyectos por lo que la relación puede ser "participación".

4. **Identificar candidatos únicos:**

   En el caso de la entidad usuario el email es único, en el caso de proyecto el nombre podría serlo pero eventualmente dos proyectos podrían quedar con el mismo nombre por lo que le agregaremos el atributo id como se muestro en la siguiente imagen.

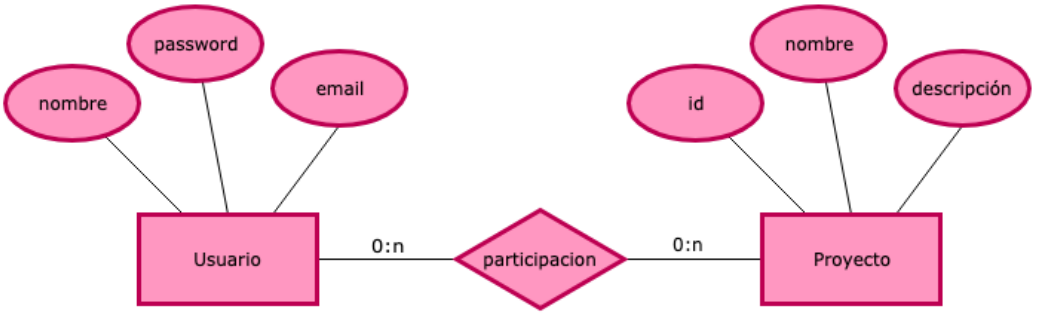


Imagen 4. Modelo Conceptual.

Ahora que tenemos el modelo conceptual terminado, procedamos con el modelo lógico.

Ahora te toca a ti (2)

Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe realizar el diagrama del diseño conceptual, considerando la relación entre ambas entidades.

Modelo lógico

Para traspasar el modelo conceptual a lógico recordemos nuestros pasos:

1. Transformar todas las entidades en tablas y agregar los atributos como columnas de la tabla.

2. Transformar todas las relaciones del tipo N:N en tablas.

3. Propagar la clave primaria de las tablas en las relaciones 1:N desde el lado de las 1 al lado de las N.

En este caso tenemos dos entidades más la entidad de la relación N a N. Al propagar las claves quedaremos con el diagrama de la siguiente imagen:

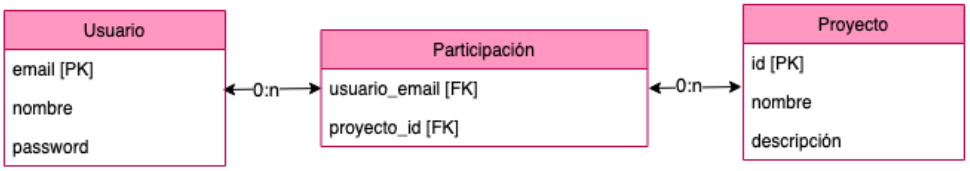


Imagen 5. Modelo Lógico.

Ahora te toca a ti (3)

Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe diseñar el modelo lógico basado en el resultado del ejercicio propuesto 2.

Pero todavía nos queda hacernos unas preguntas importantes:

- ¿Puede un usuario trabajar dos veces en el mismo proyecto?.
- ¿Cómo evitamos que esto suceda?.

Recordemos que estos casos son delicados pues producen redundancia y no queremos eso, por lo que procederemos con la normalización y aprenderás cómo hacerlo en el siguiente capítulo.

Mientras tanto, para evitar los grupos repetitivos o sea que el usuario con email 123@123 trabaje en el proyecto 1 varias veces, podemos agregar una clave primaria compuesta. De esta forma nos aseguramos que los trabajos siempre tengan ambos campos y además que estos existan asegurando la integridad referencial.

Modelo físico

Una vez teniendo el modelo lógico, el único cambio que debemos hacer es incluir en los atributos el tipo de dato y la longitud de este para pasarlo a un modelo físico pues este será el último dato necesario para pasar a la construcción de las tablas. El modelo físico entonces nos quedaría entonces como lo que te muestro en la siguiente imagen.

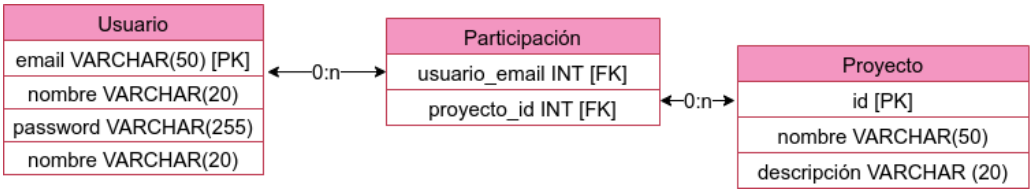


Imagen 6. Modelo físico.

Usa el siguiente código para la creación de las tablas basadas en el modelo físico de la imagen 6.

```
CREATE TABLE usuario
  (
     email    VARCHAR2(50),
     nombre   VARCHAR2(20),
     password VARCHAR2(255),
     PRIMARY KEY (email)
  );

CREATE TABLE proyecto
  (
     id          NUMBER,
     nombre      VARCHAR2(50),
     descripcion VARCHAR2(500),
     PRIMARY KEY (id)
  );

-- La tabla intermedia se crea al último para poder agregar las referencias
CREATE TABLE participacion
  (
     usuario_email VARCHAR2(45) REFERENCES usuario (email),
     proyecto_id   NUMBER REFERENCES proyecto (id),
     PRIMARY KEY(usuario_email, proyecto_id)
  );
```

Como puedes notar en la creación de la tabla participación estamos definiendo dos claves primarias “usuario_email” y “proyecto_id”, esto entonces lo entendemos como una clave primaria compuesta y como son primarias no podrán repetirse entre los registros de la tabla. Con esto estamos logrando avanzar en nuestro objetivo de evitar redundancia de información.

Insertemos datos para probar nuestra consulta.

```
INSERT
	INTO
	usuario (email, nombre, password)
VALUES ('usuario1@gmail.com', 'Juan', '12345678');

INSERT
	INTO
	usuario (email, nombre, password)
VALUES ('usuario2@gmail.com', 'Francisca', 'asdfghi');

INSERT
	INTO
	proyecto (id, nombre, descripcion)
VALUES (1, 'Proyecto1', 'Proyecto  secreto');

INSERT
	INTO
	proyecto (id, nombre, descripcion)
VALUES (2, 'Proyecto2', 'Proyecto  público');

INSERT
	INTO
	participacion (usuario_email, proyecto_id)
VALUES ('usuario1@gmail.com', 1);

INSERT
	INTO
	participacion (usuario_email, proyecto_id)
VALUES ('usuario1@gmail.com', 2);

INSERT
	INTO
	participacion (usuario_email, proyecto_id)
VALUES ('usuario2@gmail.com', 1);
```

Para seleccionar todos los usuarios con sus proyectos respectivos necesitamos hacer dos JOIN puesto que se ven involucradas las 3 tablas, usuarios con participación y participación con proyecto como te muestro en el siguiente código.

```
SELECT *
FROM   usuario
       INNER JOIN participacion
               ON email = usuario_email
       INNER JOIN proyecto
               ON proyecto.id = participacion.proyecto_id;
```

Obteniendo como respuesta la siguiente tabla.

email	nombre	password	usuario_email	proyecto_id	id	nombre	descripcion
usuario1@gmail.com	Juan	12345678	usuario1@gmail.com	1	1	Proyecto1	Proyecto secreto
usuario1@gmail.com	Juan	12345678	usuario1@gmail.com	2	2	Proyecto2	Proyecto público
usuario2@gmail.com	Francisca	asdfghi	usuario2@gmail.com	1	1	Proyecto1	Proyecto secreto

Tabla 1.Tabla de usuarios y sus proyectos.

Aún no hemos logrado a plenitud nuestra misión de evitar redundancia pero estamos cerca, y el proceso que falta lo aprenderás en el siguiente capítulo.

Ahora te toca a ti (4)

Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe diseñar el modelo físico basado en el resultado del ejercicio propuesto 3.

Normalización

Competencias

- Identificar y manejar grupos repetitivos para iniciar un proceso de normalización.
- Explicar cuando es necesario implementar la normalización entre tablas.
- Identificar las primeras tres formas normales para realizar una normalización a la base de datos.
- Reconocer la importancia de la desnormalización para facilitar consultas a la base de datos.
- Implementar técnicas de normalización para reducir redundancia entre tablas.

Introducción

En la industria podemos encontrar bases de datos enormes, que han sido creadas por un equipo completo de especialistas dedicados a la optimización del proceso de almacenamiento de información.

La mayoría de los problemas lógicos en las bases de datos se deben a las inconsistencias, a las malas relaciones o a la redundancia de información en los registros. En este capítulo aprenderás el concepto de normalización y su gran importancia para evitar esta redundancia de información innecesaria, además aprenderás cuándo se debe realizar un proceso inverso, para casos peculiares donde sea considerable tener datos almacenados repetidos en tablas diferentes con el objetivo de ofrecer la información en consultas menos procesadas.

La normalización es un proceso indispensable en bases de datos, en especial cuando empiezan a escalar por manipular grandes masas de información. La aplicación de las 3 formas normales será el camino en este capítulo para cumplir con el objetivo de cualquier base de datos.

Concepto de normalización

La normalización es un proceso que elimina la redundancia de una base de datos. El proceso consta de una serie de pasos en los cuales se van eliminando distintos tipos de redundancias con el propósito de prevenir inconsistencias. Estas etapas son muy importantes y reciben el nombre de formas normales. En este capítulo aprenderemos a llevar una base de datos a tercera forma normal.

¿Para qué sirve la normalización?

La normalización es una fórmula que podemos aplicar a una base de datos, a un modelo lógico o incluso a una tabla de excel o archivo con formato .csv que nos entreguen para crear finalmente una nueva base de datos sin redundancia.

Comúnmente se entiende por "normalizar una base de datos" pasarla a “tercera forma normal”, lo cual aprenderás más adelante en esta lectura. Sin embargo en cada etapa lo que estamos haciendo es precisamente normalizar.

En resumen, la normalización busca:

- Evitar la redundancia de datos.
- Simplificar la actualización de datos.
- Garantizar la integridad referencial (prevenir inconsistencia en las relaciones).

Para pasar una base de datos de una etapa a la otra tenemos que aplicar un conjunto de reglas, por lo cual definiremos algunos términos para luego entrar a normalizar.

Notación

Utilizaremos la siguiente notación para detallar la normalización de una tabla:

```
nombre_tabla(atributo_1, atributo_2, ..., atributo_n)
```

- Fuera de los paréntesis se tiene el nombre de la tabla con la que se trabajará.
- Dentro de los paréntesis se encuentran los atributos existentes.
- Hay casos en que dentro de una tabla pueden existir atributos repetidos. Para definirlos de forma explícita, se deben encapsular entre llaves { } de la siguiente forma:

```
nombre_tabla (
  atributo_1,
  atributo_2,
  ...,
  atributo_n,
  {atributo_repetido}
);
```

Te podrías estar preguntando ¿Atributos repetidos? ¿Cómo es posible? Veámoslo un caso a continuación.

¿Qué es un grupo repetitivo?

Lo más fácil es verlo con un ejemplo, tenemos la siguiente tabla Cliente:

N Cliente	Nombre	Teléfonos
1-9	Fernanda	123-456 123-457
2-7	Felipe	234-412
3-5	Francisco	123-411

Tabla 3. Grupo repetitivo tabla cliente.

En este caso, puede haber más de un teléfono asociado a cada cliente y no solo uno o dos, pueden haber cientos, no sabemos. Por eso es un atributo (o grupo de atributos) repetitivo, lo anterior es muy similar a esta nueva tabla:

N Cliente	Nombre	Teléfono1	Teléfono2
1	Fernanda	123-456	123-457
2	Felipe	234-412	
3	Francisco	123-411	

Tabla 4. Grupos repetitivos.

En ambos casos los atributos se repiten. Ahora si supiéramos que solo existe la posibilidad que el Cliente tenga 2 teléfonos, no sería un grupo repetitivo. El problema es cuando no conocemos realmente la cantidad de columnas como para poder almacenar los datos.

Utilizando la notación aprendida podríamos representar nuestra tabla como:

```
Cliente(N_Cliente, Nombre, {Teléfono})
```

Para estos casos debemos pensar en dividir esta entidad en 2 o más tablas y relacionarlas por llaves primarias y foráneas.

Identificando las claves primarias y foráneas

Para identificar la clave primaria dentro de una tabla, utilizaremos el símbolo # y letras en negrita, para representar claves foráneas dentro de una tabla las identificamos con letras cursivas. En caso de que sea necesario crear primarias compuestas, todos los atributos correspondientes a claves compuestas deberán estar en negritas y empezar con el signo numeral(#).

Ejemplo de clave primaria en una tabla:

```
Tabla1 (
  #clavePrimaria,
  atributo_1,
  ...,
  atributo_n
);
```

Ejemplo de clave primaria y foránea en una tabla:

```
Tabla2(
  #clavePrimaria,
  claveForanea,
  ...,
  atributo_n
);
```

Ejemplo de clave primaria compuesta:

```
Tabla3(
  #atributoPrimario,
  claveForanea,
  #otroAtributoPrimarioCompuesto,
  ...,
  atributo_n
);
```

Ahora que entendemos como incluir las claves a nuestras tablas en esta notación, apliquemoslo al ejemplo de la tabla cliente de la siguiente manera:

```
Cliente (
 #N_Cliente,
 Nombre,
 {telefono}
);
```

Como probablemente te das cuenta, por cada teléfono que tenga el cliente se tendrá que guardar otra fila en la tabla repitiendo los datos de la persona, esta redundancia la aprenderemos a tratar con la implementación de las formas normales a continuación.

Ahora te toca a ti (5)

Continuando con el caso propuesto de la plataforma de reseñas de películas, escribir las tablas en la notación aprendida.

Implementación de formas normales

Para ejemplificar el uso de las formas normales, utilizaremos el siguiente ejemplo, que corresponde a la factura de un paciente en un hospital, donde se muestran los datos del paciente y los ítems que consumió como verás en la imagen 7.

Factura		# 23464 20/04/2019
Nombre paciente: Marta Fuentes	#Paciente: 4724	
Dirección: Monjitas 245	Comuna: Santiago	
COD-SISTEMA-SALUD: 10	Ciudad: Santiago	

ITEM	NOMBRE	VALOR
200	Pieza semiprivada	150
204	Televisión	10
245	Rayos X	25
414	Exámenes	35
	SUBTOTAL	220
	%IMPUESTO	22
	TOTAL	242

Imagen 7. Ejemplo factura.

Realicemos un proceso de ingeniería inversa partiendo de la factura hasta el modelado de datos aplicando la normalización, pero antes debemos analizar la información que se nos presenta e identificar cuales son los atributos y grupos repetitivos en caso de existir, y lo haremos con la notación aprendida de la siguiente manera.

```
Factura(
 #numero_factura,
 fecha_factura,
 nombre_paciente,
 #numero_paciente,
 direccion,
 comuna,
 ciudad,
 #codigo_sistema_salud,
 {
  numero_item,
  nombre_item,
  valor_item
 },
 subtotal,
 impuesto,
 total
)
```

Los atributos subtotal, impuesto y total, se pueden eliminar, ya que estos atributos son derivables, es decir, **los podemos calcular** directamente en la aplicación o software, por lo que no sería necesario almacenarlo a menos que queramos persistir esta información para futuros filtros o estadísticas, no obstante también pudiéramos usar las funciones de las sentencias SQL como el "SUM()" para calcularlos, la situación entonces se convierte en un tema subjetivo que dependerá de la perspectiva del diseñador de bases de datos y en todo caso estos diseños pasan por un proceso de pruebas e incluso están enlazadas a la experiencia de usuario, considerando que más información solicitada significa más tiempo de espera en una consulta.

La tabla nos quedaría entonces como en la siguiente imagen.

#Factura	Fecha factura	Nombre paciente	Numero paciente	Dirección	Comuna	Ciudad	CSS	Numero Item	Nombre Item	Valor Item

Imagen 8. Tabla de atributos iniciales en factura.

Primera Forma Normal (1FN)

Ahora que tenemos la tabla Factura en la notación definida en puntos anteriores empecemos con la normalización.

Para que una tabla se encuentre normalizada acorde a la Primera Forma Normal (1FN), la tabla debe cumplir las siguientes condiciones:

- Cada campo o atributo deben ser atómicos, es decir debe contener un único valor.
- No pueden haber grupos repetitivos.

Como observamos en nuestro ejemplo, tenemos un grupo repetitivo de ***{numero_item, nombre_item,valor_item}***. Por lo que crearemos una nueva tabla llamada ***FacturaItem*** y su notación será la siguiente:

```
FacturaItem (
  #numero_factura,
  #numero_item,
  nombre_item,
  valor_item
)
```

Donde esta nueva tabla contiene todos los atributos del grupo repetitivo, y se crea una clave primaria compuesta con el ***numero_factura*** y ***numero_item***.

A la tabla factura se le elimina entonces el grupo repetitivo, quedando:

```
Factura(
  #numero_factura,
  fecha_factura,
  nombre_paciente,
  #numero_paciente,
  direccion,
  comuna,
  ciudad,
  #codigo_sistema_salud
)
```

El resultado de esta división la podemos ver gráficamente en la siguiente imagen.

Factura	# 23464 20/04/2019
Nombre paciente: Marta Fuentes	#Paciente: 4724
Dirección: Monjitas 245	Comuna: Santiago
COD-SISTEMA-SALUD: 10	Ciudad: Santiago

#Factura	ITEM	NOMBRE	VALOR
23464	200	Pieza semiprivada	150
23464	204	Televisión	10
23464	245	Rayos X	25
23464	414	Exámenes	35

Imagen 9. Primera división de la tabla factura.

Pasamos ahora la tabla factura de la izquierda a la misma presentación de la tabla de la derecha quedando como la imagen 10.

Factura

#Factura	Fecha factura	Nombre	#Paciente	Dirección	Comuna	Ciudad	CSS
23464	20/04/19	Marta Fuentes	4724	Monjtas 245	Santiago	Santiago	10

FacturaItem

#Factura	#Item	Nombre	Valor
23464	200	Pieza semiprivada	150
23464	204	Televisión	10
23464	245	Rayos X	25
23464	414	Exámenes	35

Imagen 10. Primera forma normal en el ejemplo del Hospital.

Cada grupo repetitivo se deja como una nueva tabla, manteniendo la clave de la cual provienen. Normalmente esta tabla tendrá una clave primaria compuesta por la clave primaria de la tabla original y el atributo del cual dependen los demás atributos del grupo repetitivo.

Ahora te toca a ti (6)

Realizar la primera forma normal con la siguiente tabla.

Peliculas				
Id_Pelicula	Pelicula	Genero	ID_Actor	Actor
1	Interestelar	Ficción	1	Matthew McConaughey
1	Interestelar	Ficción	2	Anne Hathaway
2	En busca de la felicidad	Drama	3	Will Smith
2	En busca de la felicidad	Drama	4	Jaden Smith

Imagen 11. Primera forma normal plataforma de reseñas de películas

Segunda Forma Normal (2FN)

Esta forma debe cumplir las siguientes condiciones:

- Debe satisfacer la 1FN.
- Cada atributo debe depender de la clave primaria, y no solo una parte de ella.
- Los atributos que dependen de manera parcial de la clave primaria deben ser eliminados o almacenados en una nueva entidad.

Para pasar una tabla a esta forma, nos debemos fijar en las tablas que tengan claves primarias compuestas, ya que en ellas se pueden dar dependencias parciales. En otras palabras, **si una entidad depende de sólo una parte de la clave primaria compuesta**, se deberá eliminar de esa tabla y llevarla a una nueva con la clave primaria que le corresponde.

Aplicando la Segunda Forma Normal en nuestro ejemplo

En la tabla ***FacturaItem*** tenemos dos claves, el número de factura y el número de ítem.

```
FacturaItem (
  #numero_factura,
  #numero_item,
  nombre_item,
  valor_item
)
```

Cuando ocurre algo de esta naturaleza, hay que determinar si los otros atributos dependen de una clave, o bien de ambas claves.

- nombre_item depende del numero_item.
- valor_item: Pueden ocurrir dos casos.
    - Depende de la relación numero_factura y numero_item, ya que es el valor obtenido para la factura, no unitario.
    - Es el valor unitario y depende de numero_item.

Para solucionar este conflicto, deberemos hablar con nuestro cliente para definir cuál de las dos opciones debemos seguir. Para efectos de este ejercicio, optamos por la segunda opción.

Como tenemos atributos que dependen de ambas claves y otros de una sola clave, tenemos que separarlas en 2 tablas nuevamente.

```
FacturaItem (
  #numero_factura,
  numero_item
)
```

```
Item(
  #numero_item,
  nombre_item,
  valor_item
)
```

Resultando las tablas que verás en la siguiente imagen:

#Factura	ITEM
23464	200
23464	204
23464	245
23464	414

ITEM	NOMBRE	VALOR
200	Pieza semiprivada	150
204	Televisión	10
245	Rayos X	25
414	Exámenes	35

Imagen 12. Segunda Forma Normal para las tablas Item y FacturaItem.

Tenemos la tabla factura que cumple la primera forma normal, por lo que tenemos la siguiente anotación:

```
Factura(
  #numero_factura,
  fecha_factura,
  nombre_paciente,
  #numero_paciente,
  direccion,
  comuna,
  ciudad,
  codigo_sistema_salud
```

```
)
```

Acá nuevamente tenemos una clave primaria compuesta del número de la factura y número del paciente. Por lo que debemos determinar cada uno de los atributos si dependen de una clave, o bien de ambas claves.

- `fecha_factura` depende de la relación entre el `numero_factura` y el numero_paciente.
- `nombre_paciente` depende del paciente, en este caso `numero_paciente`.
- `dirección` depende del paciente, en este caso `numero_paciente`.
- `comuna` depende del paciente, en este caso `numero_paciente`.
- `ciudad` depende del paciente, en este caso `numero_paciente`.
- `codigo_sistema_salud` depende del paciente, en este caso `numero_paciente`.

Esto entonces resulta en que tendrémos 2 tablas:

```
Factura(
  #numero_factura,
  numero_paciente,
  fecha_factura
)
```

```
Paciente(
  #numero_paciente,
  nombre_paciente,
  dirección,
  comuna,
  ciudad,
  código_sistema_salud
)
```

Las cuales podemos ver a continuación en la imagen 13.

Paciente

#Paciente	Nombre	#Dirección	Comuna	Ciudad	CSS
4724	Marta Fuentes	Monjitas 245	Santiago	Santiago	10

Factura

#Factura	Fecha factura	#Paciente
23464	20/04/19	4724

Imagen 13. Resultado de la 2FN para las tablas Paciente y Factura.

Ahora te toca a ti (7)

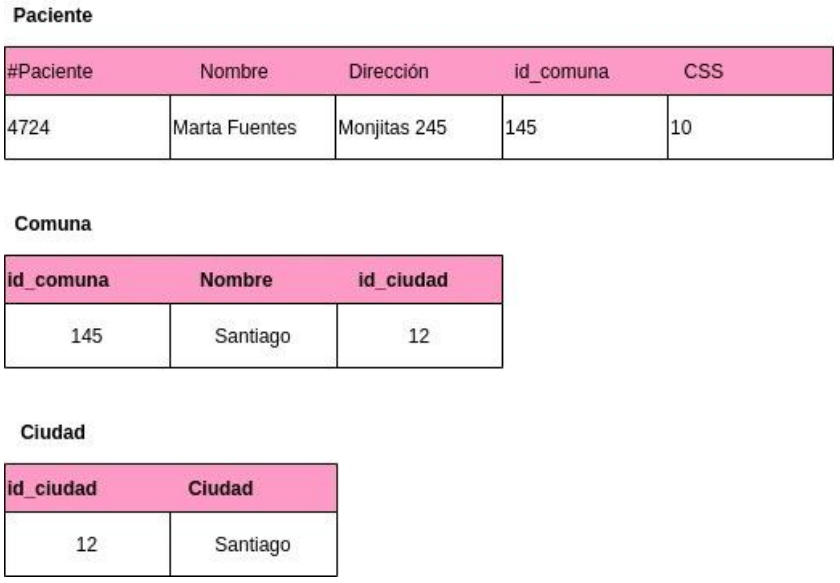
Realizar la segunda forma normal con el resultado del ejercicio de las películas expuesto en el ejercicio propuesto 6.

Tercera Forma Normal (3FN)

Esta forma debe cumplir las siguientes condiciones:

- Debe satisfacer 2FN.
- Toda entidad debe depender directamente de la clave primaria.

Si observamos en la imagen 14, las tablas que tenemos hasta el momento nos preguntamos ¿El paciente tiene alguna relación directa con la comuna y la ciudad en sí?.



Paciente

#Paciente	Nombre	Dirección	id_comuna	CSS
4724	Marta Fuentes	Monjitas 245	145	10

Comuna

id_comuna	Nombre	id_ciudad
145	Santiago	12

Ciudad

id_ciudad	Ciudad
12	Santiago

Imagen 14. Resultado de la 2FN.

La respuesta es no. A esta dependencia se le denomina dependencia funcional transitiva, y se define como la dependencia que tiene un atributo con otro de la misma entidad de forma indirecta y transitiva, es decir, en una tabla de atributos A, B y C, el atributo A depende directamente de B, y a su vez B depende directamente de C.

¿Qué sucede? Que A depende indirectamente y transitivamente de C pues su dependencia directa (B) está dependiendo directamente de C. La dependencia funcional transitiva solo puede ocurrir en entidades con 3 o más atributos en esta fase, si aún quedan atributos que no dependen directamente de la clave primaria, queda llevarlas a una nueva tabla y se puede relacionar a través de una clave foránea con la tabla proveniente. Para pasar a esta forma, debemos encargarnos de eliminar toda dependencia funcional transitiva.

Aplicando la Tercera Forma Normal en nuestro ejemplo

Tenemos la tabla Paciente, en la cual nombre_paciente, numero_paciente, direccion, `código_sistema_salud` dependenden directamente del paciente, pero comuna y ciudad no dependen directamente del paciente, por lo que tenemos que separar estos atributos de esta tabla.

Nuestras nuevas tablas quedarían de la siguiente manera:

```
Paciente(
  #numero_paciente,
  nombre_paciente,
  direccion,
  codigo_sistema_salud,
  id_comuna
)
```

```
Comuna(
  #id_comuna,
  nombre_comuna,
  ciudad
)
```

En este caso, dejamos una `id_comuna` como clave foránea en la tabla Paciente, para hacer referencia a ella.

Ahora, si observamos la tabla resultante Comuna, la ciudad no depende directamente de la `id_comuna`, por lo que nuevamente debemos separar dichos atributos, resultando las siguientes tablas:

```
Comuna (
  #id_comuna,
  nombre comuna,
  id_ciudad
)
```

```
Ciudad(
  #id_ciudad,
  nombre_ciudad
)
```

Nuevamente, `id_ciudad` en la tabla comuna queda como clave foránea a la Tabla ciudad con el id_ciudad.

Para el resultado de este ejercicio, nos quedan las siguientes tablas en 3FN expuestas en la imagen 15.

Paciente

#Paciente	Nombre	Dirección	id_comuna	CSS
4724	Marta Fuentes	Monjitas 245	145	10

Comuna

id_comuna	Nombre	id_ciudad
145	Santiago	12

Ciudad

id_ciudad	Ciudad
12	Santiago

Imagen 15. Tercera Forma Normal.

En un modelo de bases de datos quedaría representado de la forma expuesta en la siguiente imagen.

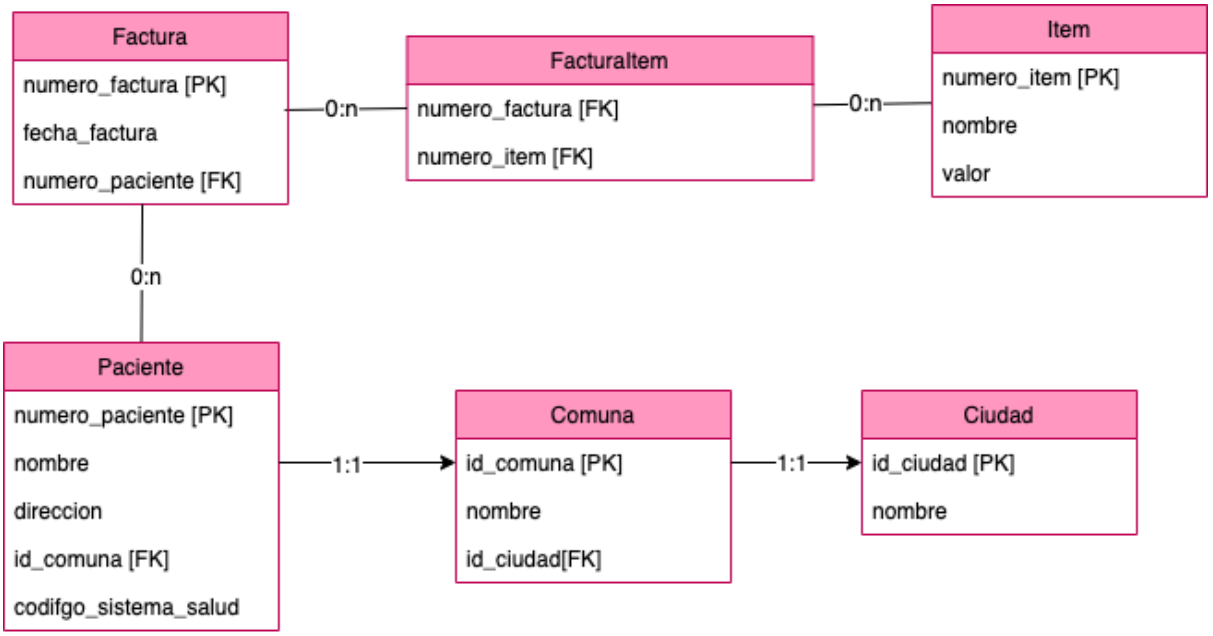


Imagen 16. Modelo de base de datos.

Ahora te toca a ti (8)

Continuando con el ejercicio de las películas, realiza la tercera forma normal con el resultado del ejercicio propuesto 7.

Resumen de la normalización

Como podemos ver, el proceso de normalización conlleva una serie de pasos ordenados que nos permiten reducir la cantidad de datos que debemos almacenar en una base de datos, evitando así la redundancia de datos, simplifica la actualización de los datos y garantiza integridad referencial, previniendo la inconsistencia en las relaciones.

Como podemos ver en la siguiente imagen, en cada paso de la normalización se va reduciendo la cantidad de datos que tenemos en nuestro universo de la base de datos.

Imagen 17. Resumen de Normalización.

¿Qué pasaría si no hubiésemos normalizado nuestra base de datos?

Si pasamos nuestra vista inicial de factura a una tabla, sin normalizar, tenemos la tabla de la siguiente imagen donde solo estamos mostrando los registros correspondientes a Marta Fuentes.

numero factura	fecha factura	nombre paciente	numero paciente	direccion paciente	comuna paciente	ciudad paciente	Cod-sis-salud	numero item	nombre item	valor item
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	200	Pieza semiprivada	150
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	204	Televisión	10
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	245	Rayos X	25
23464	20/04/2019	Marta Fuentes	4724	Monjitas 245	Santiago	Santiago	10	414	Exámenes	35

Imagen 18. Tabla sin normalizar.

Imaginemos que queremos cambiar la dirección de nuestro paciente. Al tener todo desnormalizado, antes hubiésemos tenido que ir a buscar en cada uno de los registros de las facturas y cambiar la dirección del paciente correspondiente. En cambio, al tener normalizado, basta que consultemos la tabla de Pacientes y realizar la actualización en dicha tabla.

Lo bueno de tener la base de datos completamente normalizada es que minimizamos el espacio necesario para almacenar los datos y reducimos las anomalías de mantención.

Desnormalización y sus usos

Por desnormalización, entendemos el proceso de añadir redundancia en las tablas de manera deliberada. Una de las motivaciones para desnormalizar tablas en una base de datos es el reducir el tiempo de consulta al implementar joins en múltiples tablas.

Un elemento importante a considerar es el hecho que desnormalizar no significa ignorar el proceso de normalización. Más bien, es un paso posterior a la normalización donde buscamos maximizar la eficiencia y representación de los datos, a expensas de hacer más compleja la mantención de una tabla específica.

En una base de datos tradicional buscamos modularizar las entidades asociadas a un fenómeno en distintas tablas para reducir la redundancia y problemas lógicos en éstas. En una base de datos desnormalizada, preferimos aumentar la redundancia de nuestra base para reducir la cantidad de consultas mediante joins y ganar eficiencia en las consultas.

Ejemplo de desnormalización

Tenemos una base de datos sobre pacientes en un hospital, la cual está compuesta por tres tablas.

- Una tabla Paciente, que registra el nombre del paciente, el tipo de tratamiento y el doctor con el que se atendió, éstas últimas dos columnas están registradas como claves foráneas así como ves en la siguiente tabla.

Paciente	Tratamiento_id	Doctor_id
María Zenteno	1	1
Fernando Sánchez	2	2
Jose Artigas	1	2
Emilia Tapia	1	1
Felipe Zamorano	2	1
César Oyarce	2	1
Catalina Maillard	1	2

Imagen 19 Ejemplo 1 de desnormalización.

- Una tabla Doctores que vincula la clave primaria con el nombre del doctor tratante como en la siguiente tabla.

Id	Nombre
1	Astorquiza
2	Feris

Imagen 20. Ejemplo 2 de desnormalización.

- Una tabla Tratamiento que vincula la clave primaria con el nombre del tratamiento asignado al paciente.

Id	Nombre
1	Astorquiza
2	Feris

Imagen 21. Ejemplo 3 de desnormalización.

Resulta que por sí sola, la tabla Paciente no nos informa con quién se trataron ni qué tratamiento tuvieron. Para generar sentido sobre estos números, es necesario complementar la información existente en la tabla Paciente con las tablas Doctores y Tratamiento.

Se observa que la tabla Doctores presenta una cantidad menor de datos registrados, lo cual nos permite hacer uso eficiente de la memoria en nuestra base de datos. Lo mismo se aplica para el caso de la tabla Tratamientos.

Para este ejemplo práctico, observamos que la aplicación de consultas mediante el comando JOIN no es lo suficientemente compleja, dada la baja cantidad de casos y la simpleza de las consultas utilizadas.

Una tabla desnormalizada implica duplicar la información de manera deliberada en una nueva representación de todas las tablas solicitadas.

Verás ahora la tabla desnormalizada que permitirá asociar cada paciente con un doctor y un tratamiento específico.

Paciente	Tratamiento_id	Tratamiento_non	Doctor_id	Doctor_nombre
María Zenteno	1	Control Médico	1	Astorquiza
Fernando Sánchez	2	Biopsia	2	Feris
Jose Artigas	1	Control Médico	2	Feris
Emilia Tapia	1	Control Médico	1	Astorquiza
Felipe Zamorano	2	Biopsia	1	Astorquiza
César Oyarce	2	Biopsia	1	Astorquiza
Catalina Maillard	1	Control Médico	2	Feris

Imagen 22. Tabla desnormalizada.

¿Y por qué debemos desnormalizar?

La desnormalización busca como objetivo optimizar el funcionamiento de una base de datos con el costo de agregar datos redundantes. El precio de agregar información adicional en una tabla puede generar que a la larga cueste menos con respecto a las consultas que se deba estar haciendo de manera reiterada, en pocas palabras es un dilema entre almacenar menos registros pero acomplejar las consultas a la base de datos versus ofrecer un tiempo de respuesta optimo a cambio de tener redundancia en las tablas. Esta no es una decisión genérica pues siempre dependerá del caso de uso y la problemática en evaluación.

Ahora te toca a ti (9)

Una tienda de remates tiene vendedores que manejan diferentes almacenes en diferentes direcciones y ciudades. Tienen una base de datos normalizada, sin embargo, consideran que las consultas están tardando mucho porque son consultas muy procesadas con diferentes comandos y subqueries, por lo que se ven en la necesidad de desnormalizar las siguientes 3 tablas mostradas en la siguiente imagen.

Direción

id	Nombre	ciudad
1	Maipu 727	1
2	Santo Domingo 1450	2
3	Santa Isabel 517	1
4	San Pablo 1361	2

Almacen

id	Responsable	Ciudad	Dirección
1	Valentina	1	1
2	Valentina	2	2
2	Omar	1	3
2	Omar	2	4

Ciudad

id	Nombre
1	Concepción
2	Santiago

Imagen 23. Tablas normalizadas para ejercicio de desnormalización.

Diccionario de datos

Competencias

- Reconocer la utilidad del diccionario de datos.
- Elaborar un diccionario de datos detallando un modelo relacional.

Introducción

Al pensar en un diccionario, seguramente se nos viene a la mente la idea de un libro con palabras y definiciones, donde consultamos aquella información que nos causa duda o curiosidad.

El diccionario de datos no es más que una guía descriptiva de cómo está construida la base de datos, sus entidades, relaciones y restricciones para un análisis detallado del modelo implementado, su uso persiste por casos en donde alguien que no está íntimamente familiarizado con nuestro modelo de datos necesita saber algo de las tablas, sus relaciones, los tipos de datos o cualquier detalle que le sea de utilidad, por ende recurrirá al diccionario de datos. Aquí radica la importancia de comprender este concepto.

Concepto diccionario de datos

El diccionario de datos es un repositorio de metadatos que contiene las definiciones de los objetos de datos, descripciones y relaciones entre sí. Este diccionario, incluye las características lógicas y específicas de los datos, como el nombre, descripción, alias, contenido y dominio.

¿Para qué sirve el diccionario de datos?

La información documentada es crucial en las bases de datos relacionales, pues permite que los analistas conozcan los detalles y descripciones de los elementos de la base de datos, sus relaciones, los permisos y usuarios asociados, obteniendo una lista de objetos que forman parte del flujo de datos de todo el sistema.

El diccionario de datos es un conjunto de tablas y vistas que dan información sobre el contenido de una base de datos:

- Las estructuras de almacenamiento;
- Las usuarios y sus derechos;
- Los objetos (tablas, vistas, índices, procedimientos, funciones, etc.).
- etc.

Pertenece a `SYS` y se guarda en el tablespace `SYSTEM`. Se construye al momento de la creación de la base de datos y es actualizado automáticamente cuando se ejecutan sentencias SQL DDL (Data Definition Language) (CREATE, ALTER, DROP).

Este documento debe ser transparente o agnóstico al motor y lenguaje que estemos utilizando, sin embargo, Oracle contiene instrucciones que nos permiten obtener fácilmente la información que requerimos.

Elaboración de un diccionario de datos

Oracle pone a disposición una serie de vistas, que nos proporcionan valiosa información del diccionario de datos. Estas vistas, son grandes consultas que son son tratadas como tablas, lo que permite llamar a estas vistas con el from como si fuera una tabla real.

Para una información detallada respecto al diccionario de datos y sus vistas, verificar el siguiente enlace.

```
-- Seleccionar las tablas y sus comentarios, si es que los tiene
SELECT table_name TABLA,
       comments   COMENTARIO
FROM   user_tab_comments
WHERE  table_name NOT LIKE 'BIN$%'
ORDER  BY tabla;
-- resultado
TABLA        |COMENTARIO|
-------------|----------|
PARTICIPACION|          |
PROYECTO     |          |
USUARIO      |          |
-- Seleccionar información de las columnas de las tablas
SELECT table_name,
       column_id,
       column_name,
       data_type,
       data_length
FROM   user_tab_columns;
-- resultado
TABLE_NAME   |COLUMN_ID|COLUMN_NAME  |DATA_TYPE|DATA_LENGTH|
-------------|---------|-------------|---------|-----------|
PARTICIPACION|        1|USUARIO_EMAIL|VARCHAR2 |         45|
PARTICIPACION|        2|PROYECTO_ID  |NUMBER   |         22|
PROYECTO     |        1|ID           |NUMBER   |         22|
PROYECTO     |        2|NOMBRE       |VARCHAR2 |         50|
PROYECTO     |        3|DESCRIPCION  |VARCHAR2 |        500|
USUARIO      |        1|EMAIL        |VARCHAR2 |         50|
USUARIO      |        2|NOMBRE       |VARCHAR2 |         20|
USUARIO      |        3|PASSWORD     |VARCHAR2 |        255|
```

Ahora te toca a ti (10)

Genera una sentencia SQL que extraiga la información de tus tablas para elaborar un diccionario de datos. Usa cualquiera de tus bases de datos y nombra tus atributos solo con letras.

Solución a los ejercicios planteados - Ahora te toca a ti

1. Crear las tablas correspondientes al modelo físico de la siguiente imagen, la cual corresponde a una plataforma que ofrece reseña de películas. Este modelo tiende a tener registros duplicados y redundancia de datos, es decir, no está normalizado pero lo estaremos normalizando en próximos ejercicios propuestos.

Solución:

```
CREATE TABLE peliculas
  (
     id_pelicula NUMBER,
     pelicula    VARCHAR2(30) NOT NULL,
     genero      VARCHAR2(30) NOT NULL,
     PRIMARY KEY (id_pelicula)
  );

CREATE TABLE actores
  (
     id_pelicula NUMBER REFERENCES peliculas (id_pelicula),
     id_actor    NUMBER,
     nombre      VARCHAR2(30) NOT NULL,
     PRIMARY KEY (id_actor)
  );
```

2. Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe realizar el diagrama del diseño conceptual, considerando la relación entre ambas entidades.

Solución:

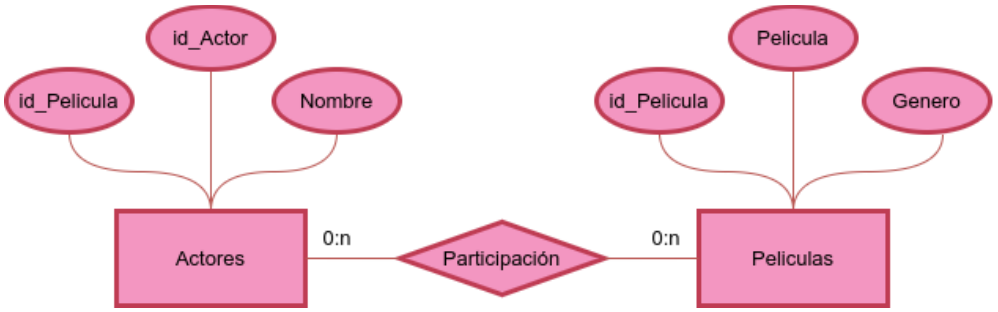


Imagen 24. Solución pregunta 2.

3. Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe diseñar el modelo lógico basado en el resultado del ejercicio propuesto 2.

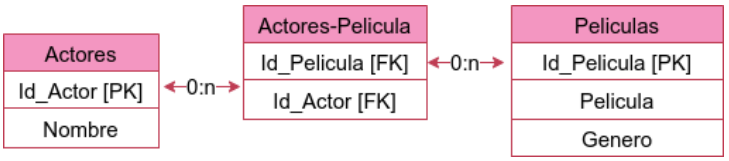


Imagen 25. Solución pregunta 3.

4. Continuando con el caso propuesto de la plataforma de reseñas de películas, se debe diseñar el modelo físico basado en el resultado del ejercicio propuesto 3.

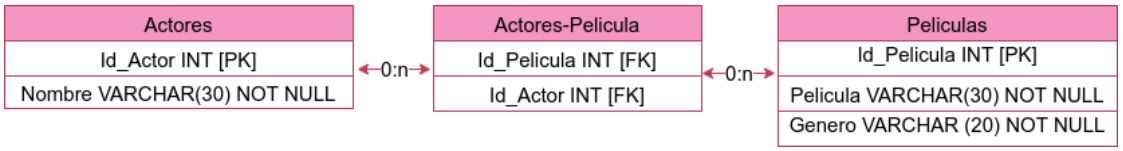


Imagen 26. Solución pregunta 24

5. Continuando con el caso propuesto de la plataforma de reseñas de películas, escribir las tablas en la notación aprendida.

```
Actores(#id_Actor, Nombre)
Peliculas(#id_pelicula, pelicula, genero)
actores_peliculas(#id_pelicula, #id_Actor)
```

6. Realizar la primera forma normal con la siguiente tabla:

Peliculas				
Id_Pelicula	Pelicula	Genero	ID_Actor	Actor
1	Interestelar	Ficción	1	Matthew McConaughey
1	Interestelar	Ficción	2	Anne Hathaway
2	En busca de la felicidad	Drama	3	Will Smith
2	En busca de la felicidad	Drama	4	Jaden Smith

Imagen 27. Tabla pregunta 6

Solución:

Pelicula		
Id_Pelicula	**Pelicula**	**Genero**
1	Interestelar	Ficción
2	En busca de la felicidad	Drama

Actores		
ID_Actor	**Actor**	**id_Pelicula**
1	Matthew McConaughey	1
2	Anne Hathaway	1
3	Will Smith	2
4	Jaden Smith	2

Imagen 28. Solución pregunta 6.

7. Realizar la segunda forma normal con el resultado del ejercicio de las películas expuesto en el ejercicio propuesto 6.

Ejemplo de solución:

Pelicula		
Id_Pelicula	**Pelicula**	**Genero**
1	Interestelar	Ficcion
2	En busca de la felicidad	Drama

Actores_Peliculas	
Id_Pelicula	ID_Actor
1	1
1	2
2	3
2	4

Actores	
ID_Actor	**Actor**
1	Matthew McConaughey
2	Anne Hathaway
3	Will Smith
4	Jaden Smith

Imagen 29. Solución pregunta 7.

8. Continuando con el ejercicio de las películas, realiza la tercera forma normal con el resultado del ejercicio propuesto 7.

Pelicula

Id_Pelicula	Pelicula	Genero
1	Interestelar	1
2	En busca de la felicidad	2

Actores

Id_Actor	Actor
1	Matthew McConaughey
1	Anne Hathaway
3	Will Smith
4	Jaden Smith

Actores_Peliculas

Id_Pelicula	Id_Actor
1	1
1	2
2	3
2	4

Generos

Id_Pelicula	Nombre
1	Ficción
1	Drama

Imagen 30. Tablas pregunta 8.

9. Una tienda de remates tiene vendedores que manejan diferentes almacenes en diferentes direcciones y ciudades. Tienen una base de datos normalizada, sin embargo, consideran que las consultas están tardando mucho porque son consultas muy procesadas con diferentes comandos y subquerys, por lo que se ven en la necesidad de desnormalizar las siguientes 3 tablas mostradas en la siguiente imagen.

Direción

id	Nombre	ciudad
1	Maipu 727	1
2	Santo Domingo 1450	2
3	Santa Isabel 517	1
4	San Pablo 1361	2

Almacen

id	Responsable	Ciudad	Dirección
1	Valentina	1	1
2	Valentina	2	2
2	Omar	1	3
2	Omar	2	4

Ciudad

id	Nombre
1	Concepción
2	Santiago

Imagen 31. Solución pregunta 9.

10. Genera una sentencia SQL **que extraiga la información de tus tablas** para elaborar un diccionario de datos. **Usa cualquiera de tus bases de datos** y nombra tus atributos solo con letras.

Ejemplo de solución:

```
SELECT a.table_name as nombre_tabla,
       a.column_id as identificador,
       a.column_name as nombre_columna,
       a.data_type as tipo_dato,
       a.data_length as largo
FROM   user_tab_columns a;
```