El modelo físico y el caso N a N

¡Comencemos!

¿Qué aprenderás?

- 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:

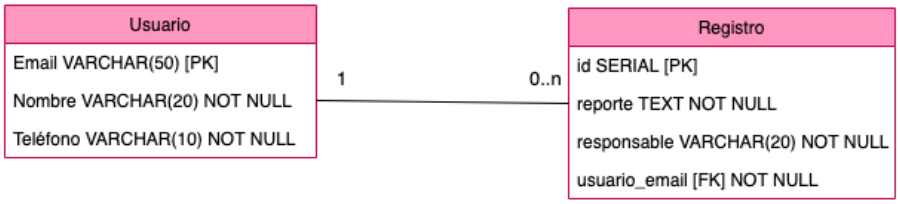

Imagen 1. Modelo Físico.

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:

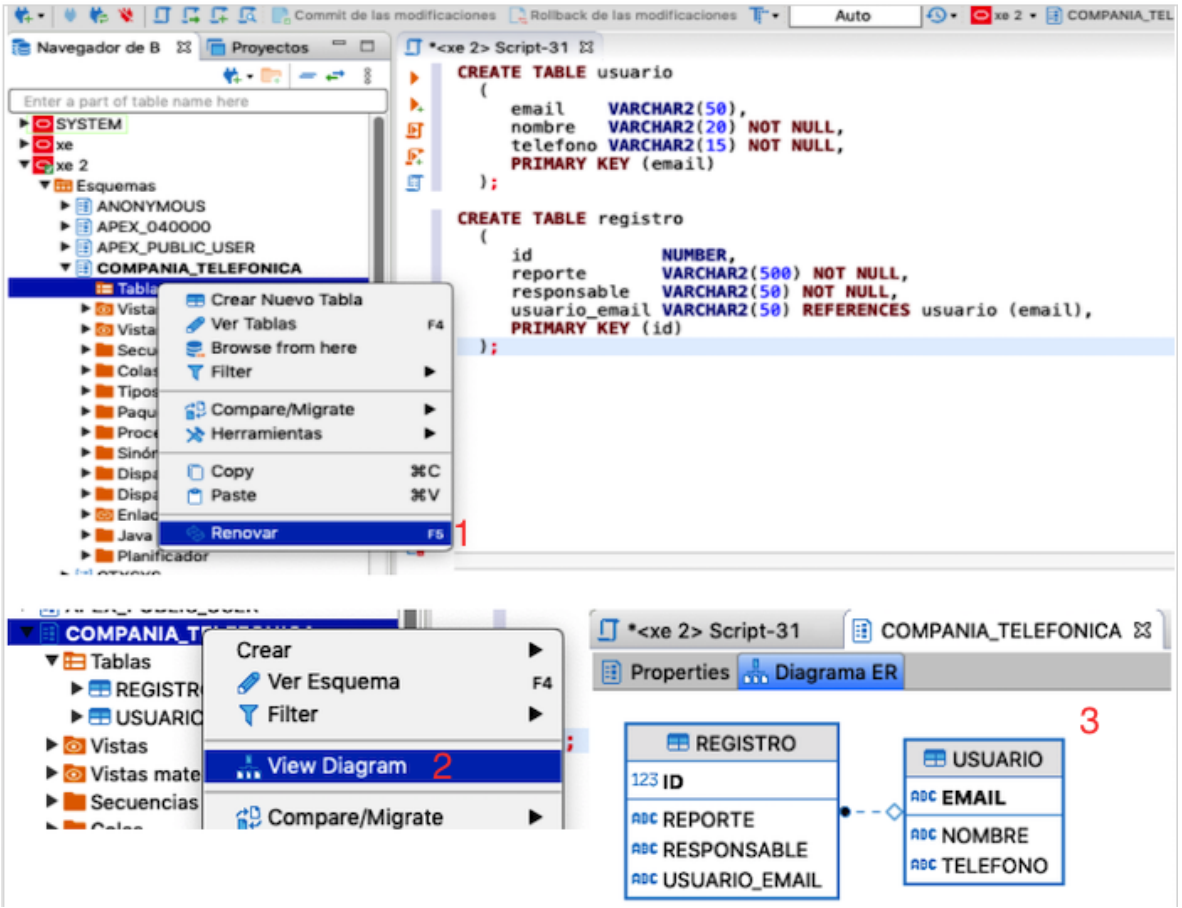

Imagen 2. Generar diagrama con DBeaver.

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.

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.

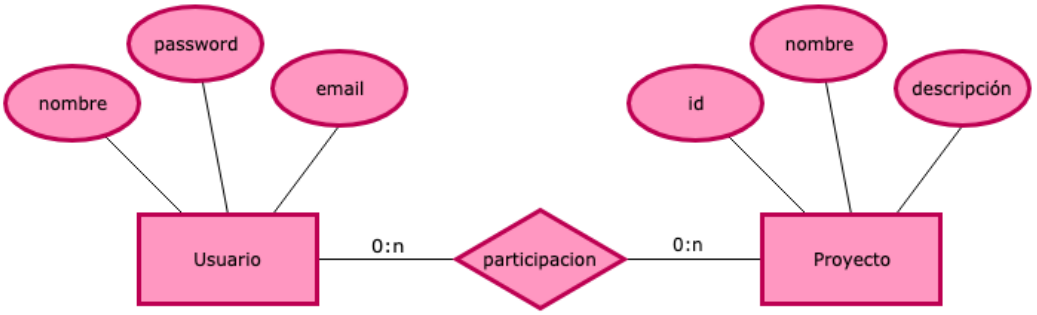

Imagen 3. Modelo Conceptual.

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

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:

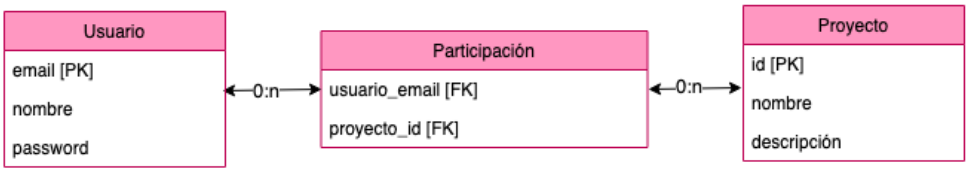

Imagen 4. Modelo Lógico.

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.

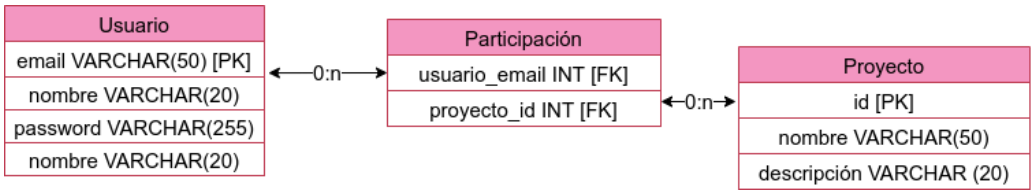

Imagen 5. Modelo físico.

Usa el siguiente código para la creación de las tablas basadas en el modelo físico de la imagen 5.

```
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.

Insertaremos 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.