Talento Digital para Chile:

Módulo 4
Lenguaje de Consultas a una Base de Datos

nodovirtual.awakelab.cl

JELOU futurejob by Adalid Chile

4.1.- Contenido 1: Elaborar un modelo de datos de acuerdo a los estándares de modelamiento para resolver un problema de baja complejidad

Objetivo de la jornada

1. Elabora un modelo de Entidad-Relación identificando entidades, identificadores y relaciones entre entidades para representar un problema dado
2. Elabora un modelo relacional utilizando reglas de transformación y de normalización (3F) de acuerdo a un modelo entidad relación dado para representar un problema
3. Elabora un diccionario de datos detallando un modelo relacional que resuelve un problema dado

4.1.1.- El modelo Entidad-Relación

4.1.1.1. El proceso de abstracción

Algunas personas que trabajan con programas de aplicación de sistemas de administración de bases de datos relacionales (RDBMS) pueden preguntarse por qué deberían preocuparse por el diseño de bases de datos. Después de todo, la mayoría de los programas vienen con bases de datos de muestra que puede copiar y modificar para adaptarlas a sus propias necesidades, e incluso puede tomar prestadas tablas de las bases de datos de muestra y usarlas en otras bases de datos que haya creado. Algunos programas también proporcionan herramientas que lo guiarán a través del proceso de definición y creación de tablas. Sin embargo, estas herramientas en realidad no lo ayudan a diseñar una base de datos, simplemente lo ayudan a crear las tablas físicas que incluirá en la base de datos.

Lo que hay que comprender es que es mejor para usted utilizar estas herramientas después de haber creado la estructura lógica de la base de datos. Los programas

RDBMS proporcionan las herramientas de diseño y las bases de datos de muestra para ayudar a minimizar el tiempo que lleva implementar la estructura de la base de datos físicamente. En teoría, reducir el tiempo de implementación le da más tiempo para concentrarse en crear y desarrollar aplicaciones para el usuario final.

Sin embargo, la razón principal por la que deberíamos preocuparnos por el diseño de la base de datos es que es crucial para la coherencia, integridad y precisión de los datos en una base de datos. Si diseña una base de datos de forma incorrecta, le resultará difícil recuperar ciertos tipos de información y correrá el riesgo de que sus búsquedas produzcan información inexacta. La información inexacta es probablemente el resultado más perjudicial del diseño inadecuado de la base de datos; puede afectar negativamente los resultados de su organización. De hecho, si su base de datos afecta la forma en que su empresa realiza sus operaciones diarias o si va a influir en la dirección futura de su negocio, debe preocuparse por el diseño de la base de datos.

Veamos esto desde una perspectiva diferente por un momento: Piense en cómo haría para construir una casa personalizada. ¿Qué es lo primero que vas a hacer? Ciertamente, no va a contratar a un contratista inmediatamente y dejar que construya su casa como quiera. Seguro que lo hará primero será contratar a un arquitecto para que diseñe su nueva casa y luego contrata a un contratista para que la construya. El arquitecto explorará sus necesidades y las expresará como un conjunto de planos, registrando decisiones sobre tamaño y forma y requisitos para varios sistemas (estructurales, mecánicos, eléctricos). A continuación, el contratista adquirirá la mano de obra y los materiales, incluidos los sistemas enumerados, y luego los ensamblará de acuerdo con los dibujos y las especificaciones.
Hay distintos objetivos que se deben alcanzar para diseñar una estructura de base de datos sólida y buena. Puede evitar muchos problemas si tiene estos objetivos en mente y se concentra constantemente en ellos mientras diseña su base de datos.

- La base de datos admite la recuperación de información requerida y ad hoc. La base de datos debe almacenar los datos necesarios para respaldar los requisitos de información definidos durante el proceso de diseño y las posibles consultas ad hoc que pueda plantear un usuario.

- Las tablas están construidas de manera adecuada y eficiente. Cada tabla en la base de datos representa un solo tema, se compone de campos relativamente distintos, mantiene los datos redundantes en un mínimo, y se identifica en toda la base de datos mediante un campo con valores únicos.

- La integridad de los datos se impone a nivel de campo, tabla y relación. Estos niveles de integridad ayudan a garantizar que las estructuras de datos y sus valores serán válidos y precisos en todo momento.

- La base de datos respalda las reglas comerciales relevantes para la organización. Los datos deben proporcionar información válida y precisa que siempre sea significativa para la empresa.

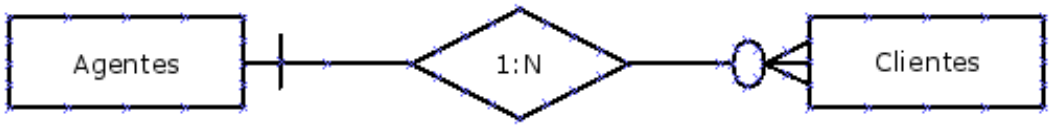
- La base de datos se presta a un crecimiento futuro. La estructura de la base de datos debe ser fácil de modificar o expandir a medida que los requisitos de información de la empresa cambian y crecen.

A veces puede resultarle difícil cumplir con estos objetivos, pero sin duda estará satisfecho con la estructura final de su base de datos una vez que los haya cumplido.

4.1.1.2. El modelo conceptual de Entidad-Relación

En general, los métodos tradicionales de diseño de bases de datos incorporan tres fases: análisis de requisitos, modelado de datos y normalización.
La fase de análisis de requisitos implica un examen del negocio que se está modelando, entrevistas con los usuarios y la gerencia para evaluar el sistema actual y analizar las necesidades futuras, y una evaluación de los requisitos de información para el negocio en su conjunto. Este proceso es relativamente sencillo y, de hecho, el proceso de diseño que presentamos sigue la misma línea de pensamiento.

La fase de modelado de datos implica modelar la estructura de la base de datos utilizando un método de modelado de datos, como diagramación "entidad-relación" (ER), modelado de objetos semánticos, modelado de roles de objetos o modelado UML. Cada uno de estos métodos de modelado proporciona un medio para representar visualmente varios aspectos de la estructura de la base de datos, como las tablas, las relaciones de las tablas y las características de las relaciones. Por ejemplo, en la figura se muestra una versión básica de los diagramas ER.



Cada método de modelado de datos incorpora un conjunto de símbolos de diagramación que se utilizan para representar la estructura y las características de una base de datos. Por ejemplo, el diagrama de la anterior proporciona información sobre varios aspectos de la base de datos.

- Los rectángulos representan dos tablas llamadas AGENTES y CLIENTES.
- El diamante representa una relación entre estas dos tablas y el "1: N" dentro del diamante indica que es una relación de uno a varios.

- La línea vertical al lado de la tabla AGENTES indica que un cliente debe estar asociado con un agente, y el círculo al lado de la tabla CLIENTES indica que un agente no necesariamente tiene que estar asociado con un cliente.

Los campos también se definen y asocian con las tablas apropiadas durante la fase de modelado de datos. A cada tabla se le asigna una clave principal, se identifican e implementan varios niveles de integridad de los datos y se establecen relaciones mediante claves externas. Una vez que las estructuras de la tabla inicial están completas y las relaciones se han establecido de acuerdo con el modelo de datos, la base de datos está lista para pasar por la fase de normalización.

La normalización es el proceso de descomponer tablas grandes en más pequeñas para eliminar datos redundantes y datos duplicados y evitar problemas al insertar, actualizar o eliminar datos. Durante el proceso de normalización, las estructuras de la tabla se prueban con las formas normales y luego se modifican si se encuentra alguno de los problemas mencionados anteriormente. Una forma normal es un conjunto específico de reglas que se pueden utilizar para probar la estructura de una tabla y asegurarse de que sea sólida y no presente problemas. Hay varias formas normales y cada una se usa para probar un conjunto particular de problemas. Las formas normales actualmente en uso son Primera Forma Normal, Segunda Forma Normal, Tercera Forma Normal, Cuarta Forma Normal, Quinta Forma Normal, Sexta Forma Normal, Forma Normal Boyce-Codd y Forma Normal de Dominio / Clave.
Para nuestros ejemplos prácticos, más adelante en las clases, hemos adaptado una metodología visual que permite ver el proceso relacional con mayor claridad en términos de implementación.

4.1.1.3. Identificación de entidades

La razón para tener una base de datos es almacenar valores de datos sobre entidades y luego recuperar los valores de datos sobre esas entidades según sea necesario. Para lograr esto, debe haber alguna forma de distinguir una entidad de otra. Los identificadores de entidad realizan esta función. Los identificadores de entidad son atributos, específicamente, atributos clave que identifican de forma única a cada entidad. Un identificador de entidad no es un atributo opcional; cada entidad debe tener un atributo clave para identificarla de forma única. Los identificadores de entidad (atributos clave) se convierten en claves primarias en una tabla.

Entidad compuesta: Una entidad compuesta también se conoce como entidad puente. Este puente se usa para manejar las relaciones de muchos a muchos que la entidad tradicional no pudo manejar. Esta entidad se encuentra entre las dos entidades que son de interés y esta entidad compuesta comparte las claves primarias de ambas tablas de conexión. Esta entidad compuesta también se conoce como "de unión" porque tiene las características de una entidad y una relación. Aprenderá sobre una entidad compuesta más adelante en el curso. La entidad compuesta existe solo para

vincular otras dos entidades juntas. Una entidad compuesta no tiene un identificador de entidad propio. En su lugar, recibe los identificadores de entidad de cada una de las dos entidades que sirve para vincular y los combina para formar un identificador de entidad compuesto. Por ejemplo, en un caso donde se tienen muchas asignaturas y muchos profesores capaces de impartir distintas de las asignaturas.

4.1.1.4. Definición de atributos e identificadores únicos

Una entidad es algo sobre lo que almacenamos datos. Por ejemplo, un cliente es una entidad, al igual que un artículo de mercadería almacenado por algún negocio. Las entidades no son necesariamente tangibles. Por ejemplo, un concierto es una entidad. Las entidades tienen datos que las describen (sus atributos). Por ejemplo, una entidad **cliente** suele ser descrita por un

- número de cliente,
- nombre de pila,
- apellido,
- calle,
- ciudad,
- estado,
- código postal, y
- número de teléfono.

Una entidad de concierto puede describirse utilizando los siguientes atributos: 1) título, 2) fecha, 3) ubicación y 4) nombre del intérprete.

Por ejemplo, si queremos definir la entidad **profesor**, como lo haremos más adelante, podríamos describirla mediante:

- nombre
- apellido
- escuela
- Fecha de contratación
- sueldo

Cuando representamos entidades en una base de datos, en realidad almacenamos solo los atributos. Cada grupo de atributos que describe una sola ocurrencia del mundo real de una entidad actúa para representar una instancia de una entidad.

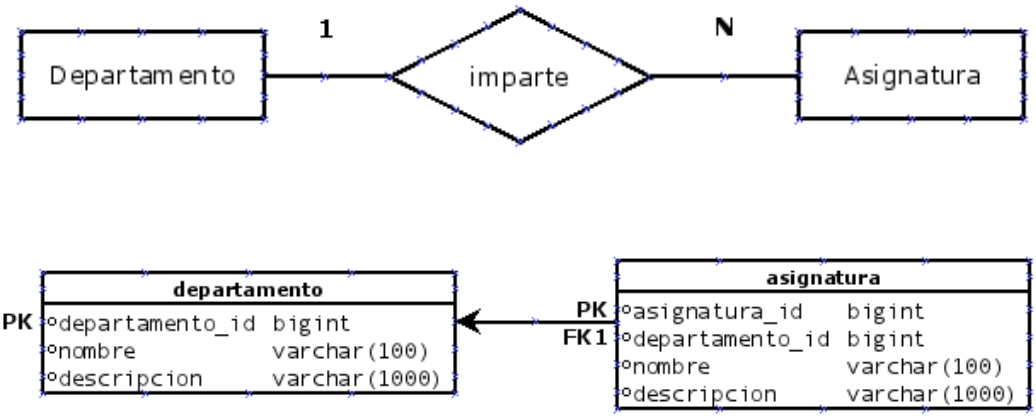
4.1.1.5. Tipos de relación entre entidades

Estamos relacionados con las cosas que nos rodean de una manera u otra. Estamos relacionados con nuestra familia, amigos, etc. y estas relaciones son de diferentes tipos. Por ejemplo, usted y su padre están relacionados. Existe una relación entre padres e hijos. Tienes un solo padre, pero él puede tener muchos hijos. Usted y sus hermanos están relacionados. Usted y su profesor están relacionados. El profesor le enseña a usted y a muchos otros estudiantes. También estudia con diferentes profesores.

Las relaciones de la base de datos también son muy similares a tales relaciones. Hemos visto el modelo ER en su forma más básica y también sabemos de las Entidades. Ahora, veremos las relaciones de la base de datos.

Por ejemplo, un profesor enseña a los estudiantes. Aquí, "enseña" es una relación y esta es la relación entre una entidad Profesor y una entidad Estudiante.

Supongamos que estamos analizando/construyendo un modelo para una escuela y entre otras entidades tenemos la entidad 'Departamento' (departamento_id, nombre, descripcion) y 'Asignatura' (asignatura_id, departamento_id, nombre, descripcion). Almacenamos los datos de 'Departamento' (por ejemplo, un registro para Ciencias Sociales) en una tabla y los detalles de sus asignaturas en la tabla 'Asignatura'. Ahora, para vincular estas dos tablas, debemos insertar la clave principal 'departamento_id' de la tabla 'Departamento' en la tabla 'Asignatura'. Esta clave actúa como una clave externa para la tabla 'Asignatura' y se refiere a una columna con el mismo nombre en la tabla 'Departamento'. Así es como se establece una relación entre dos tablas.

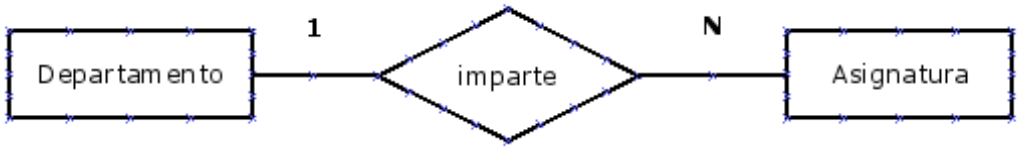


Hay tres tipos de relaciones que pueden existir entre dos entidades:

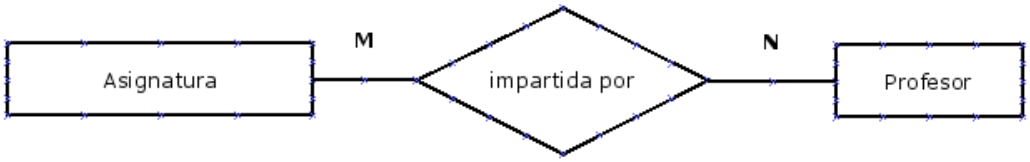
- Relación uno a uno: Esta relación existe cuando cada registro de una tabla está relacionado con un solo registro de la otra tabla. No es muy común ver este tipo de casos, ya que, se podría organizar la información en la misma tabla. Sin embargo, esta relación podría utilizarse por motivos de seguridad. Por

ejemplo, podemos almacenar fácilmente la identificación del pasaporte solo en una tabla "Persona". Sin embargo, creamos otra tabla para el "Pasaporte" porque el número de pasaporte puede ser información confidencial y debe ocultarse a ciertos usuarios. Entonces, al hacer una tabla separada, brindamos seguridad adicional que solo ciertos usuarios de la base de datos pueden verla.

- Relación de uno a varios o de varios a uno (One-to-Many or Many-to-One): Esta relación existe cuando cada registro de una tabla puede estar relacionado con uno o más de un registro de la otra tabla. Esta relación es la relación más común encontrada. Una relación de uno a muchos también se puede decir como una relación de muchos a uno, dependiendo de la forma en que la veamos. (Ver figuras más arriba).

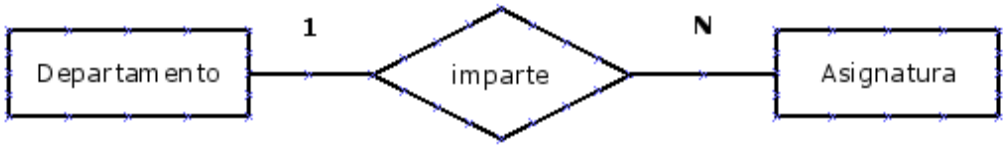


- Relación de muchos a muchos (Many-to-Many): Existe tal relación cuando cada registro de la primera tabla puede estar relacionado con uno o más de un registro de la segunda tabla y un solo registro de la segunda tabla puede estar relacionado con uno o más de un registro de la primera tabla. Una relación de varios a varios puede verse como una relación de dos a varios que está vinculada por una 'tabla de unión', 'tabla de enlace' o 'tabla asociada'. La tabla de unión vincula dos tablas al tener campos que son la clave principal de las otras dos tablas. Podemos entender esto con el siguiente ejemplo gráfico.

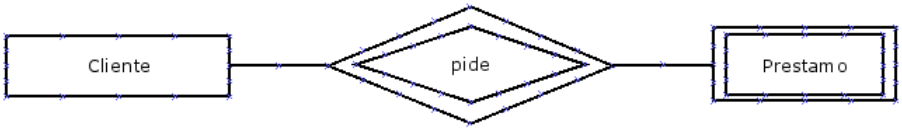


4.1.1.6. Entidades débiles y fuertes

Una **entidad fuerte** no depende de ninguna otra entidad del esquema. Una entidad fuerte siempre tendrá una clave primaria. Las entidades fuertes están representadas por un solo rectángulo. La relación de dos entidades fuertes está representada por un solo diamante. Varias entidades fuertes, cuando se combinan juntas, crean un conjunto de entidades fuertes. En la figura **Departamento** es una entidad fuerte, ya que, no depende de otras entidades.



Una **entidad débil** depende de una entidad fuerte para asegurar su existencia. A diferencia de una entidad fuerte, una entidad débil no tiene ninguna clave primaria. En cambio, tiene una clave discriminadora parcial. Una entidad débil está representada por un doble rectángulo. La relación entre una entidad fuerte y una débil está representada por un doble diamante. Por ejemplo



Diferencia entre entidad fuerte y débil:

1. La entidad fuerte siempre tiene una clave primaria. Mientras que la entidad débil tiene una clave discriminadora parcial.
2. La entidad fuerte no depende de ninguna otra entidad. La entidad débil depende de la entidad fuerte.
3. La entidad fuerte está representada por un solo rectángulo. La entidad débil está representada por un rectángulo doble.
1. La relación de dos entidades fuertes está representada por un solo diamante. Mientras que la relación entre una entidad fuerte y una débil está representada por un doble diamante.
2. La entidad fuerte tiene participación total o no.

Tenga en consideración de que muchas veces, en la vida real, estas características de representación gráfica no son seguidas al pie de la letra, no es extraño encontrar diagramas sin doble diamantes ni doble rectángulos, sino que simplemente diamantes y rectángulos simples.

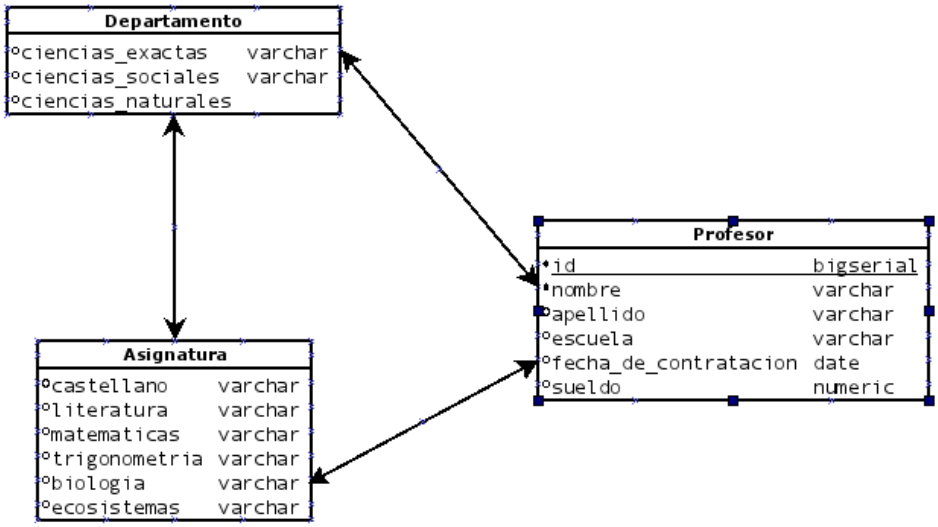
4.1.2.- El modelo Relacional

4.1.2.1. El modelo relacional y sus diferencias con el modelo conceptual

El modelo relacional (RM) representa la base de datos como una colección de relaciones. Una relación no es más que una tabla de valores. Cada fila de la tabla representa una colección de valores de datos relacionados. Estas filas de la tabla indican una entidad o relación del mundo real.

El nombre de la tabla y los nombres de las columnas son útiles para interpretar el significado de los valores en cada fila. Los datos se representan como un conjunto de relaciones. En el modelo relacional, los datos se almacenan como tablas. Sin embargo, el almacenamiento físico de los datos es independiente de la forma en que los datos están organizados lógicamente.

La figura muestra un ejemplo de un modelo relacional a rasgos generales:

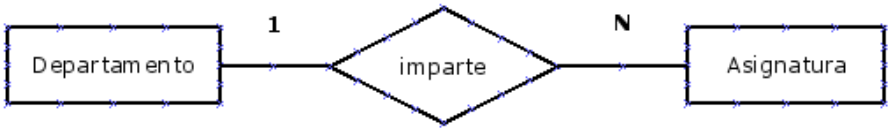


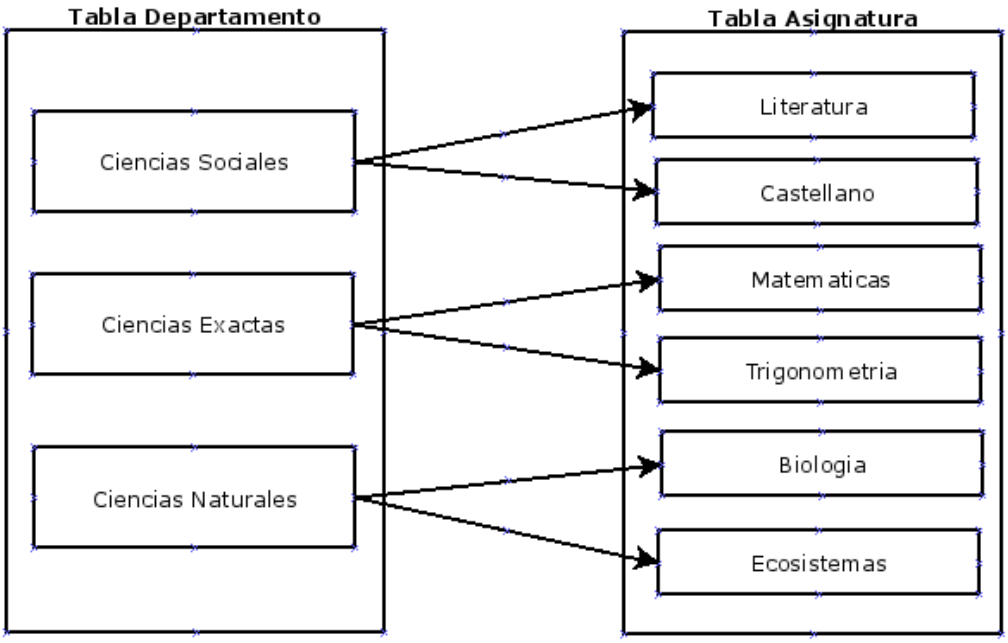
Conceptos del modelo relacional

- Atributo: cada columna de una tabla. Los atributos son las propiedades que definen una relación. por ejemplo, descripción, nombre, etc.
- Tablas: en el modelo relacional, las relaciones se guardan en formato de tabla. Se almacena junto con sus entidades. Una tabla tiene filas y columnas de propiedades. Las filas representan registros y las columnas representan atributos.
- Tupla: no es más que una sola fila de una tabla, que contiene un solo registro.
- Esquema de relación: un esquema de relación representa el nombre de la relación con sus atributos.
- Grado: El número total de atributos que en la relación se denomina grado de relación.
- Cardinalidad: número total de filas presentes en la tabla.

- Columna: la columna representa el conjunto de valores para un atributo específico.
- Instancia de relación: la instancia de relación es un conjunto finito de tuplas en el sistema RDBMS. Las instancias de relación nunca tienen tuplas duplicadas.
- Clave de relación: cada fila tiene uno, dos o varios atributos, lo que se denomina clave de relación.
- Dominio de atributo: cada atributo tiene un valor y alcance predefinidos que se conoce como dominio de atributo

Por otro lado, el modelo conceptual tiene como principal objetivo a grandes rasgos establecer las entidades y sus relaciones. En este nivel de modelado de datos, apenas hay detalles disponibles de la estructura real de la base de datos. No obstante, sirve como una primera aproximación a lo que será un modelo finalmente.





4.1.2.2. Reglas de transformación

La transformación se realiza empleando las siguientes reglas:

- Toda entidad se transforma en una tabla.
- Todo atributo se transforma en columna dentro de la tabla.
- El identificador único de la entidad se convierte en clave primaria.
- Como las relaciones del modelo E/R no tienen equivalente en el modelo relacional, ya que sólo existen tablas y operaciones entre ellas, es necesario aplicar lo siguiente:

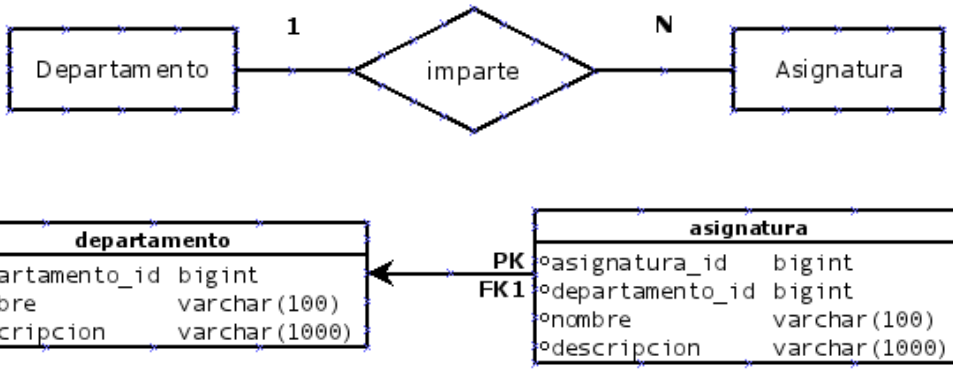
    - En las relaciones M:N se crea una nueva tabla que tendrá como clave primaria la concatenación de los atributos clave de las entidades que asocia y con los atributos propios de la relación si los hay. Esta tabla posee dos claves ajenas, una por cada entidad con la que está relacionada.

    - En las relaciones 1:N la entidad del lado N de la relación añade el conjunto de campos necesarios para incorporar a sus atributos la totalidad de la clave primaria de la entidad del lado 1, creando una clave ajena, de modo que se puedan relacionar ambas tablas mediante operadores relacionales. El nombre de la relación desaparece.

    - Las relaciones 1:1 se transforman en función de las cardinalidades:

        - Cuando ambas entidades participan con cardinalidades (1,1) propagando cualquiera de los atributos identificadores y sus atributos asociados creando una única tabla con el conjunto de los atributos de ambas entidades. La clave primaria sería cualquiera de las dos.
        - Cuando ambas tablas tienen cardinalidades (0,1) crear una nueva tabla a partir de la relación con las dos claves de ambas.
        - Propagar la clave de la entidad con cardinalidad (1,1) a la entidad que tenga (0,1).

Por si no se ha dado cuenta ya hemos mostrado un ejemplo de una transformación anteriormente:



4.1.2.3. Asignando tipos de datos y restricciones al modelo

Importante: Para nuestros propósitos prácticos utilizaremos la base de datos PostgreSQL que es un sistema de administración de bases de datos relacionales de código abierto y gratuito que enfatiza la extensibilidad y el cumplimiento de SQL.

PostgreSQL ofrece un amplio conjunto de tipos de datos nativos para los usuarios. Los usuarios pueden agregar nuevos tipos con la ayuda del comando **CREATE TYPE**.

En un nivel básico, PostgreSQL admite los siguientes tipos de datos:

- Boolean
- Character
- Number
- Temporales (basados en tiempo)
- PostgreSQL extensiones
- Objeto grande binario (BLOB - Binary Large Object

Veremos cada uno de estos tipos, excepto BLOB, que se usa con menos frecuencia.

El tipo de datos booleano

El tipo booleano es probablemente el tipo más simple posible. Puede almacenar solo dos valores posibles, verdadero y falso, y **NULL**, para cuando el valor sea desconocido. La declaración de tipo para una columna booleana es oficialmente booleana, pero casi siempre se abrevia simplemente a bool. Cuando se insertan datos en una columna booleana en una tabla, PostgreSQL es bastante flexible sobre lo que interpretará como verdadero y falso. Cualquier otra cosa será rechazada, excepto **NULL**. Al igual que las palabras clave SQL, también distinguen entre mayúsculas y minúsculas; por ejemplo, 'TRUE' también se interpretará como un booleano verdadero.

Formas de especificar valores booleanos

- Interpretado como verdadero: **'1'; 'yes'; 'y'; 'true'; 't'**
- Interpretado como falso: **'0'; 'no'; 'n'; 'false'; 'f'**

Tipos de datos de caracteres

Los tipos de datos de caracteres son probablemente los más utilizados en cualquier base de datos. Hay tres variantes de tipo de carácter, que se utilizan para representar las siguientes variaciones de cadena:

- Un solo caracter
- Cadenas de caracteres de longitud fija
- Cadenas de caracteres de longitud variable

Estos son tipos de caracteres estándar de SQL, pero PostgreSQL también admite un tipo de texto, que es similar al tipo de longitud variable, excepto que no necesitamos declarar ningún límite superior para la longitud. Sin embargo, este no es un tipo de SQL estándar, por lo que debe usarse con precaución. Los tipos estándar se definen mediante **char**, **char(n)** y **varchar(n)**. A continuación, los tipos de caracteres de PostgreSQL.

- **char**: Un solo carácter.
- **char(n)**: Un conjunto de caracteres de exactamente **n** caracteres de longitud, rellenados con espacios. Si usted intenta almacenar una cadena demasiado larga, se generará un error.
- **varchar(n)**: Un conjunto de caracteres de hasta **n** caracteres de longitud, sin relleno. PostgreSQL tiene una extensión del estándar SQL que permite especificar **varchar** sin longitud, lo que hace que la longitud sea efectivamente ilimitada.
- **text**: Efectivamente, una cadena de caracteres de longitud ilimitada, como **varchar** pero sin la necesidad de definir un máximo. Esta es una extensión de PostgreSQL para el estándar SQL.

Tipos de datos numéricos

Los tipos de números en PostgreSQL son un poco más complejos, pero no son particularmente difíciles de entender. Hay dos tipos distintos de números que podemos almacenar en la base de datos: números enteros y números de punto flotante. Estos se subdividen de nuevo, con un subtipo especial de entero, el tipo **serial** y diferentes tamaños de números enteros:

- **small integer (smallint)**: Un entero de 2 bytes con signo, capaz de almacenar números de –32768 a 32767
- **integer (int)**: Un entero de 4 bytes, capaz de almacenar números de –2147483648 a 2147483647
- **serial**: Igual que entero, excepto que su valor normalmente lo ingresa automáticamente PostgreSQL (Nota: también existe **bigserial**, lo usaremos en ejemplos)

Los números de coma flotante también se subdividen en aquellos que ofrecen valores de coma flotante de propósito general y números de precisión fija:

- **float (float(n))**: Un número de coma flotante con al menos la precisión **n**, hasta un máximo de 8 bytes de almacenamiento.
- **float8 (real)**: Un número de punto flotante de doble precisión (8 bytes).
- **numeric (numeric(p, s))**: Un número real con **p** dígitos, **s** de ellos después del punto decimal. A diferencia de **float**, este es siempre un número exacto, pero es menos eficiente trabajar con él que los números de coma flotante ordinarios.

money (numeric(9, 2)): Un tipo específico de PostgreSQL, aunque común en otras bases de datos. El tipo de dinero pasó a ser la versión 8.0 de PostgreSQL y puede eliminarse en versiones posteriores. Debería utilizar numérico en su lugar.

Precaución: Un error común es pensar que numérico **(5,2)** puede almacenar un número, como **12345.12**. Esto no es correcto. El número total de dígitos almacenados es solo cinco, por lo que una declaración numérica (5,2) puede almacenar solo hasta **999,99** antes de desbordarse.

Tipos de datos temporales

PostgreSQL tiene un rango de tipos relacionados con la fecha y la hora, como vamos a detallar, pero generalmente nos limitaremos a los tipos estándar de SQL92.

- **date**: Almacena información de fecha.
- **time**: Almacena información de tiempo.
- **timestamp**: Almacena una fecha y hora.
- **interval**: Almacena información sobre una diferencia en las marcas de tiempo.
- **timestampz**: Una extensión de PostgreSQL que almacena una marca de tiempo y la información de la zona horaria.

Tipos de datos especiales

Desde sus orígenes como un sistema de base de datos de investigación, PostgreSQL ha adquirido algunos tipos de datos inusuales para almacenar tipos de datos geométricos y de red, como se muestran a continuación. El uso de cualquiera de estas características especiales de PostgreSQL hará que la portabilidad de una base de datos de PostgreSQL sea bastante pobre, por lo que generalmente, tendemos a evitar estas extensiones. Para obtener más información sobre estos tipos, consulte la documentación de PostgreSQL, en "Tipos de datos" (Data Types).

- **box**: Una caja rectangular.
- **line**: Un conjunto de puntos.
- **point**: Un par geométrico de números.
- **lseg**: Un segmento de línea.

- **polygon**: Una línea geométrica cerrada.
- **cidr or inet**: Una dirección IP versión 4, como 196.192.12.45
- **macaddr**: Una dirección MAC (Ethernet física).

Arreglos

PostgreSQL tiene otra característica inusual: la capacidad de almacenar arreglos en tablas. Esta no era una característica estándar de SQL hasta SQL99, por lo que no es común en las implementaciones de bases de datos. Normalmente, una arreglo/matriz se implementaría utilizando una tabla adicional. Sin embargo, la función de arreglo a veces puede ser útil, especialmente cuando necesita almacenar un número fijo de elementos repetidos, y es muy fácil de usar. Hay dos sintaxis para crear matrices: la forma PostgreSQL original y la forma SQL99 más estándar.

Estilo PostgreSQL

```
CREATE TABLE tabla (refcode char(5), dias int[]);

INSERT INTO tabla VALUES('val01', '{0,1,0,1,1,1,1}');

INSERT INTO tabla VALUES('val02', '{0,1,1,1,1,0,1}');

SELECT * FROM tabla;
```

```
refcode  |      dias
---------+-----------------
Val01      | {0,1,0,1,1,1,1}
val02     | {0,1,1,1,1,0,1}
(2 rows)
```

Estilo SQL99-Arrays

```
CREATE TABLE unatabla (refcode char(5), valores int array[7]);

INSERT INTO unatabla VALUES('val01', '{0,1,0,1,1,1,1}');

INSERT INTO unatabla VALUES('val02', '{0,1,1,1,1,0,1}');

SELECT * FROM unatabla;
```

```
refcode  |     valores
---------+-----------------
val01     | {0,1,0,1,1,1,1}
Val02      | {0,1,1,1,1,0,1}
(2 rows)
```


4.1.2.4. Normalización (1FN, 2FN, 3FN)

La normalización es una técnica de diseño de base de datos que organiza las tablas de una manera que reduce la redundancia y la dependencia de los datos.

Divide las tablas más grandes en tablas más pequeñas y las vincula usando relaciones.

El inventor del modelo relacional Edgar Codd propuso la teoría de la normalización con la introducción de la Primera Forma Normal, y continuó extendiendo la teoría con la Segunda y Tercera Forma Normal. Más tarde se unió a Raymond F. Boyce para desarrollar la teoría de la forma normal de Boyce-Codd.

La teoría de la normalización de datos en SQL aún se está desarrollando más. Sin embargo, en la mayoría de las aplicaciones prácticas, la normalización alcanza su mejor nivel en la 3ª Forma Normal.

Ejemplos de normalización de base de datos

La normalización es un proceso de organización de los datos en la base de datos para evitar la redundancia de datos, anomalías de inserción, anomalías de actualización y anomalías de eliminación. Primero, analicemos las anomalías y luego las formas normales con ejemplos.

Hay tres tipos de anomalías que ocurren cuando la base de datos no está normalizada. Estos son: Anomalía de inserción, actualización y eliminación. Tomemos un ejemplo para entender esto.

Ejemplo: supongamos que la Municipalidad de Chile Chico administra sus escuelas almacenando los detalles de sus docentes en una tabla llamada profesores que tiene los siguientes atributos: **nombre**; **apellido**; **escuela**; **direccion**; **fecha_de_contratacion**; **sueldo**.

profesor_id	nombre	apellido	escuela	direccion	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	Pedro Perez 200, La Florida	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	Mao tze Tung 27, La Cisterna	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	Manquehue 12, Vitacura	2005-08-01	3400000
4	Pablo	Rojas	E-34	Caleuche S/N Panamricana	2011-10-30	300000
5	Juan Alberto	Valdivieso	Bendito Corazón de María	Manquehue 12, Vitacura	2005-08-30	8900000

La tabla anterior no está normalizada. Veremos los problemas a los que nos enfrentamos cuando una tabla no está normalizada.

- **Anomalía de actualización**: En la tabla anterior tenemos dos filas para el empleado Juan Alberto Valdivieso, ya que, trabaja en dos escuelas. Si queremos actualizar la dirección de Juan Alberto Valdivieso entonces tenemos

que actualizar la misma en dos filas o los datos se volverán inconsistentes. Si de alguna manera, la dirección correcta se actualiza en una escuela pero no en la otra, según la base de datos, Juan Alberto Valdivieso tendría dos direcciones diferentes, lo cual no es correcto y daría lugar a datos inconsistentes.

- **Anomalía de inserción**: supongamos que un nuevo profesor se unirá a una de las escuelas del municipio, el profesor está en inducción municipal y actualmente no está asignado a ninguna escuela, entonces no podríamos insertar los datos en la tabla si el campo **escuela** no permite valores nulos.

- **Anomalía de eliminación**: Supongamos que, si en un momento dado la municipalidad cierra la escuela **Gabriela Mistral**, eliminar las filas que tienen **escuela** como **Gabriela Mistral** también eliminaría la información de la profesora Juanita Pérez, ya que, ella está asignada solo a esa escuela.

Para superar estas anomalías, necesitamos normalizar los datos.

Estas son las formas normales más utilizadas:

- Primera forma normal (1NF)
- Segunda forma normal (2NF)
- Tercera forma normal (3NF)

Primera forma normal (1NF)

Según la regla de la primera forma normal, un atributo (columna) de una tabla no puede contener varios valores. Debe contener solo valores atómicos.

Supongamos que alguien tiene la idea de juntar la información de Juan Valdivieso de tal forma que deja la tabla así:

profesor_id	nombre	apellido	escuela	direccion	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	Pedro Perez 200, La Florida	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	Mao tze Tung 27, La Cisterna	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	Manquehue 12, Vitacura	2005-08-01	3400000
			Bendito Corazón de María	Manquehue 12, Vitacura	2005-08-30	8900000
4	Pablo	Rojas	E-34	Caleuche S/N Panamricana	2011-10-30	300000

Juan Alberto Valdivieso tiene una sola fila donde están almacenados sus sueldos y las escuelas en que trabaja.

Esta tabla no está en 1NF, ya que, la regla dice que "cada atributo de una tabla debe tener valores atómicos (únicos)", los valores **escuela** y **sueldo** violan esa regla para el profesor Juan.

Veamos qué pasa con la Segunda forma normal (2NF)

Segunda forma normal (2NF)

Se dice que una tabla está en 2NF si se cumplen las dos condiciones siguientes:

- La tabla está en 1NF (Primera forma normal)
- Ningún atributo no principal depende del subconjunto adecuado de cualquier llave candidata de tabla.

Un atributo que no forma parte de ninguna llave candidata se conoce como atributo no principal.
Ejemplo: supongamos que para solucionar el "problema" de conservar el hecho que Juan Alberto trabaja en dos escuelas, alguien decide modificar la tabla de la siguiente forma:

profesor_id	nombre	apellido	escuela	direccion	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	Pedro Perez 200, La Florida	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	Mao tze Tung 27, La Cisterna	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	Manquehue 12, Vitacura	2005-08-01	3400000
4	Pablo	Rojas	E-34	Caleuche S/N Panamricana	2011-10-30	300000
3	Juan Alberto	Valdivieso	Bendito Corazón de María	Manquehue 12, Vitacura	2005-08-30	8900000

La tabla está en 1NF porque cada atributo tiene valores atómicos. Sin embargo, no está en 2NF porque el atributo no principal **escuela** depende solo de **profesor_id**, que es un subconjunto adecuado de la clave candidata. Esto viola la regla para 2NF ya que la regla dice que "ningún atributo no principal depende del subconjunto adecuado de cualquier clave candidata de la tabla".

Tercera forma normal (3NF)

Se dice que un diseño de tabla está en 3NF si se cumplen las dos condiciones siguientes:

- La tabla debe estar en 2NF
- Se debe eliminar la dependencia funcional transitiva del atributo no principal en cualquier superclave.

Un atributo que no forma parte de ninguna clave candidata se conoce como atributo no principal.
En otras palabras, 3NF se puede explicar así: una tabla está en 3NF si está en 2NF y para cada dependencia funcional **X->Y** se cumple al menos una de las siguientes condiciones:

- **X** es una super llave de tabla.
- **Y** es un atributo principal de la tabla.

Tomemos la tabla con la que hemos trabajado, donde Juan está duplicado porque trabaja en dos escuelas del municipio (¿cómo lo hará...? en fin...).

profesor_id	nombre	apellido	escuela	direccion	fecha_de_contratacion	sueldo
1	Juanita	Perez	Gabriela Mistral	Pedro Perez 200, La Florida	2011-10-30	234000
2	Bruce	Lee	Republica Popular China	Mao tze Tung 27, La Cisterna	1993-05-22	780945
3	Juan Alberto	Valdivieso	Sagrada Concepcion	Manquehue 12, Vitacura	2005-08-01	3400000
4	Pablo	Rojas	E-34	Caleuche S/N Panamricana	2011-10-30	300000
5	Juan Alberto	Valdivieso	Bendito Corazón de María	Manquehue 12, Vitacura	2005-08-30	8900000

- Super llaves: **{profesor_id}**, **{profesor_id, nombre}**, **{profesor_id**, **nombre**, **apellido}**, etc.
- Llave candidata: **{profesor_id}**
- Atributos no-principales: todos los atributos excepto **{profesor_id}** no son principales, ya que, no forman parte de ninguna clave candidata.

Aquí, **direccion** depende de **nombre** y **apellido**. Por otro lado, **nombre** y **apellido** dependen de **profesor_id**, que hace al atributo no principal **direccion t**ransitivamente dependiente de la super clave **(profesor_id)**. Esto viola la regla de #NF.

Para que esta tabla cumpla con 3NF, tenemos que dividir la tabla en dos tablas para eliminar la dependencia transitiva:

Aquí organizamos dos tablas una para **profesor** y otra para **escuela**

profesor_id	nombre	apellido	direccion
1	Juanita	Perez	Pedro Perez 200, La Florida
2	Bruce	Lee	Mao tze Tung 27, La Cisterna
3	Juan Alberto	Valdivieso	Manquehue 12, Vitacura
4	Pablo	Rojas	Caleuche S/N Panamricana

escuela_id	profesor_id	nombre	fecha_de_contratacion	sueldo
1	1	Gabriela Mistral	2011-10-30	234000
2	2	Republica Popular China	1993-05-22	780945
3	3	Sagrada Concepcion	2005-08-01	3400000
4	4	E-34	2011-10-30	300000
5	3	Bendito Corazón de María	2005-08-30	8900000

4.1.2.5. El diccionario de datos

Siempre que buscamos o trabajamos en una nueva base de datos, verificamos el tipo de datos especificado para cada columna en cada tabla. Si tenemos suerte, podemos conseguir un "diccionario de datos": un documento que enumera cada columna; especifica si es un número, caracteres u otro tipo; y explica los valores de la columna valores. Desafortunadamente, muchas organizaciones no crean ni mantienen una buena documentación, por lo que no es inusual escuchar: "No tenemos un diccionario de datos". En ese caso, tenemos que aprender inspeccionando las estructuras de la tabla usando la consola o alguna herramienta visual que no acceso a esta meta-información.

Po ejemplo en SQL Server, el diccionario de datos es un conjunto de tablas de bases de datos que se utilizan para almacenar información sobre la definición de una base de datos. El diccionario contiene información sobre los objetos de la base de datos, como tablas, índices, columnas, tipos de datos y vistas. SQL Server utiliza el diccionario de datos para ejecutar consultas y se actualiza automáticamente cada vez que se agregan, eliminan o cambian objetos dentro de la base de datos.

Dado que tal documento no se encuentra y como estamos trabajando con PostgreSQL tenemos algunas alternativas para inspeccionar nuestra(s) base(s) de datos, **psql** proporciona una serie de comandos integrados potentes y útiles. Algunos de los ejecutados con más frecuencia son:

- **\dn** - Lista de esquemas

- **\dt** - Lista de tablas
- **\dv** - Vistas de lista
- **\df** - Funciones de lista

El uso de **\dt** enumerará las tablas en la base de datos activa. Para ver información adicional, use **\dt+** e incluirá el tamaño de la tabla en el disco.

Los comandos **psql** integrados son muy buenos. Lo que es aún mejor es que esos comandos brindan una gran información sobre cómo consultar los metadatos internos de PostgreSQL a través de los objetos **pg_catalog** e **information_schema**.

Para ver cómo funcionan los comandos integrados (por ejemplo, **dt+**), ejecute esto en **psql**:

```
\set ECHO_HIDDEN 1
```

Ahora, ejecutar un comando incorporado nos mostrará la consulta que está detrás.

```
mibasededatos=# \dt+
********* QUERY **********
SELECT n.nspname as "Schema",
  c.relname as "Name",
  CASE c.relkind WHEN 'r' THEN 'table' WHEN 'v' THEN 'view' WHEN 'm' THEN
'materialized view' WHEN 'i' THEN 'index' WHEN 'S' THEN 'sequence' WHEN 's'
THEN 'special' WHEN 'f' THEN 'foreign table' WHEN 'p' THEN 'partitioned table'
WHEN 'I' THEN 'partitioned index' END as "Type",
  pg_catalog.pg_get_userbyid(c.relowner) as "Owner",
  pg_catalog.pg_size_pretty(pg_catalog.pg_table_size(c.oid)) as "Size",
  pg_catalog.obj_description(c.oid, 'pg_class') as "Description"
FROM pg_catalog.pg_class c
     LEFT JOIN pg_catalog.pg_namespace n ON n.oid = c.relnamespace
WHERE c.relkind IN ('r','p','')
      AND n.nspname <> 'pg_catalog'
      AND n.nspname <> 'information_schema'
       AND n.nspname !~ '^pg_toast'
```

```
  AND pg_catalog.pg_table_is_visible(c.oid)
ORDER BY 1,2;
*************************
```

4.1.3.- Referencias

[1] Rahul Batra, SQL Primer An Accelerated Introduction to SQL Basics, 2018.

[2] PostgreSQL Tutorial
https://www.tutorialspoint.com/postgresql/

[3] Manpreet Kaur, PostgreSQL Development Essentials, 2016.

[4] PostgreSQL Foreign Key
https://www.postgresqltutorial.com/postgresql-foreign-key/

[5] PostgreSQL Documentation
https://www.postgresql.org/docs/12/index.html

[6] Allen G. Taylor, SQL, 2019.

[7] Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database
https://beginnersbook.com/2015/05/normalization-in-dbms/

[8] Normalization of Database
https://www.studytonight.com/dbms/database-normalization.php

[9] What is Normalization? 1NF, 2NF, 3NF, BCNF
https://guru99.es/database-normalization/