Guía de estudio - Integridad y Modelos de Datos

¡Hola! Te damos la bienvenida a esta nueva guía de estudio.

¿En qué consiste esta guía?

En este recurso revisaremos las siguientes temáticas:

- Integridad de datos.
- Cómo se puede producir un problema de coherencia por falta de integridad.
- Problemas por falta de identificadores únicos (clave primaria).
- Implementación de una clave primaria.
- Integridad referencial.
- Implementación de una clave foránea.
- Modelo de datos físicos.
- Relaciones.
- Cardinalidad.
- Implementación de una relación n a n.
- Agregar una FK.
- Borrado en cascada.
- Restricciones (null, unique y checks).
- Normalización de datos.

¡Vamos con todo!

Tabla de contenidos

Integridad de datos **3**
Actividad guiada N° 1 3
Reflexión de la actividad anterior 4
Clave primaria (PK) 4
Actividad guiada N° 2 5
Integridad referencial 5
Actividad guiada N° 3 6
Agregando una FK 7
Actividad guiada N° 4 7
Borrado en cascada 8
Actividad guiada N° 5 9
Agregar borrado en cascada a una clave foránea existente 10
Otras Restricciones 10
NOT NULL 11
UNIQUE 11
CHECKS 11
Modelos de datos 12
Modelo físico 12
Tipos de Relaciones 14
Cardinalidad 14
Ejemplo de relación de 1 a N 14
Relaciones N a N 15
Implementando un modelo N a N 15
Actividad guiada N° 6 17
Normalización de datos 18
Etapas de la normalización 19
Preguntas de preparación para una entrevista laboral a partir del módulo SQL 20
Temas interesantes para profundizar en bases de datos 20

¡Comencemos!

Integridad de datos

La integridad es un concepto importante en bases de datos y se refiere a cuánto podemos realmente confiar en nuestros en datos. Podemos decir que:

> Integridad de datos = datos correctos + datos completos

¿Por qué dejaríamos de confiar en nuestros datos? Porque a través de consultas con `insert`, `update` o `delete` podríamos alterar por accidente datos que no queremos alterar.

Supongamos que tenemos la siguiente tabla Usuarios:

Nombre	Edad
Consuelo	27
Consuelo	32
Francisco	27

Si queremos aumentar la edad de Consuelo, ¿cómo lo hacemos?

- Si actualizamos por edad, modificaremos 2 registros.
- Si actualizamos por nombre, modificaremos 2 registros.
- Es posible modificar por ambos, pero en algunas ocasiones se nos podría olvidar y causar un problema de correctitud de datos.

Actividad guiada N° 1

1. Crea la tabla `usuariosP1` con nombre `varchar(32)` y edad `Integer.`
2. Ingresa los datos de la tabla mostrada.
3. Intenta incrementar la edad de Consuelo en uno a través de un `update` utilizando únicamente la edad (deberás ver que se actualizan 2 registros).
4. Intenta incrementar la edad de Consuelo en uno a través de un `update` utilizando únicamente el nombre (deberás ver que actualizan 2 registros).
5. Ahora incrementa la edad utilizando el nombre y la edad actual.

Reflexión de la actividad anterior

Si bien podemos identificar un registro a través de un conjunto de campos, en este caso nombre y edad, tendremos que cambiar la consulta cada vez que cambie la edad del usuario.

Clave primaria (PK)

La solución a todos los problemas de la actividad anterior consiste en agregar una nueva columna, y en ella guardar un valor que sea único por cada registro. Sin embargo, simplemente agregar una columna no resuelve completamente el problema, ya que no es correcto depender únicamente de las buenas prácticas de la organización. Dicho de otro modo, se trata de una solución "frágil".

Para asegurarnos que este valor realmente no pueda repetirse (y con ello amenazar la integridad de los datos), especificaremos que esta columna es **clave primaria** (primary key, abreviado PK). Para ello, previamente crearemos la columna.

id	Nombre	Edad
1	Consuelo	27
2	Consuelo	32
3	Francisco	27

Nota: No es necesario que la clave primaria se llame id ni que sea un valor serial. Una clave primaria solo asegura que los valores ingresados sean únicos y no nulos.

Para crear una clave primaria:

En una tabla nueva	En una tabla existente **con** la columna	En una tabla existente **sin** la columna

`CREATE TABLE users(`
`  id int primary key,`
`  nombre varchar,`
`  edad int`
`);`
	`ALTER TABLE users(`
	`  ADD PRIMARY KEY (nombre_columna)`
	`)`
		`ALTER TABLE users`
		`  ADD COLUMN nombre_columna PRIMARY KEY`
		`)`

Actividad guiada N° 2

1. Modifica la tabla creada agregando la columna (sin clave primaria).
2. Modifica los registros existentes agregando los ids 1, 2 y 2 respectivamente.
    a. Aquí es cuando debemos reflexionar que simplemente con tener la columna no basta.
3. Cambia el id del último registro a 3 (limpieza de datos).
4. Agrega una clave primaria sobre la columna id.
5. Intenta agregar otro registro con el id 2 y observar el error.
6. Intenta agregar un registro sin id y observa el error.

Ideas fuerza

- La clave primaria nos permite identificar de forma única un registro.
- Una clave primaria es una restricción que impide que los valores ingresados en esa columna estén repetidos o sean nulos.
- No es necesario que la clave primaria sea un campo llamado id. También puede ser un email, por ejemplo, pero debe cumplir que sea el único registro en esa tabla con dicho email.

Integridad referencial

En otras ocasiones nos interesará que no se borre un registro relacionado con otro registro. Por ejemplo, si tenemos dos tablas relacionadas - una de usuarios y otra de pagos - podríamos insertar un pago asociado a un usuario que no existe, o podríamos borrar un usuario y se perdería la información de quién realizó un pago. En ambos casos, se trata de un riesgo para el negocio.

Actividad guiada N° 3

Dada las siguientes tablas y datos:

Tabla usuarios:

Id	Nombre	Edad
1	Consuelo	27
2	Consuelo	32
3	Francisco	27

Tabla pagos:

Id	Monto	Usuario_id
1	1500	1
2	1300	2
3	2700	4

1. Crea la tabla pagos y la tabla usuarios. En ambas, id es clave primaria, nombre es varchar(255) y monto y usuario_id son integers.
2. Ingresa los mismos datos especificados en las tablas diagramadas en la actividad.
3. Borra el usuario 2 de la tabla.
4. Selecciona todos los pagos.
    a. ¿Tiene sentido que haya un pago a un usuario 4 que no existe y nunca ha existido?
    b. ¿Tiene sentido que haya un pago al usuario 2 que acabamos de borrar y, por lo tanto, tampoco existe?

Este tipo de problemas se conocen como “de integridad referencial”. La integridad de datos del modelo peligra debido a que se pueden producir referencias incorrectas entre los datos de las distintas tablas.

Agregando una FK

Para evitar problemas de integridad referencial agregaremos claves foráneas, estas claves son otro tipo de restricción que impedirán que se agregue un registro asociado a uno no existente, o que se borre un registro relacionado. Para lograrlo utilizaremos la siguiente sintaxis.

```
[CONSTRAINT fk_name]
   FOREIGN KEY(fk_columns)
   REFERENCES parent_table(parent_key_columns)
   [ON DELETE delete_action]
   [ON UPDATE update_action]
```

Donde los elementos que están entre corchetes son opcionales y no los consideraremos por el momento. Con ello obtenemos la siguiente tabla:

```
FOREIGN KEY (fk_columns)
REFERENCES parent_table(parent_key_columns)
```

Para agregar la clave foránea (FK) podemos modificar la tabla de pagos utilizando:

```
ALTER TABLE pagos ADD
FOREIGN KEY (usuario_id)
REFERENCES usuarios (id);
```

Pero si lo aplicamos directamente después de haber realizado la actividad anterior, tendremos un error, dado que efectivamente hay inconsistencia entre nuestros datos. En la siguiente actividad realizaremos un paso a paso donde borraremos los datos, agregaremos la clave foránea y la pondremos a prueba.

Actividad guiada N° 4

Primero tenemos que arreglar los datos para apuntar a usuarios que realmente existan dentro de la tabla.

1. Borramos los datos de la tabla pagos.

```
DELETE FROM pagos;
```

2. Alteramos la tabla de pagos agregando la FK, que esta vez sí funcionará.

3. Insertamos datos válidos en la tabla de pagos.

```
INSERT INTO pagos
VALUES (1, 1500, 1), (2, 2000, 3)
```

4. Intentamos insertar datos inválidos en la tabla de pagos.

```
INSERT INTO pagos
VALUES
  (3, 2000, 2) /* El usuario 2 lo habíamos borrado en la actividad */
```

```
ERROR:  insert or update on table "Pagos" violates foreign key
constraint "pagos_usuario_id_fkey"
DETAIL:  Key (usuario_id)=(2) is not present in table "usuarios".
```

El error dice exactamente lo que esperábamos: estamos violando la restricción de clave foránea porque no existe el usuario con id 2 en la tabla usuarios.

5. No podremos borrar usuarios si tienen pagos asociados.

```
Error: update or delete on table "usuarios" violates foreign key
constraint "pagos_usuario_id_fkey" on table "pagos"
DETAIL:  Key (id)=(1) is still referenced from table "pagos".
```

Ideas fuerza

- La clave foránea nos permite protegernos de problemas de integridad referencial
- La clave foránea tiene que ir asociada a una clave única o primaria de otra tabla. De preferencia debe ser primaria.
- Al utilizar una clave foránea no podremos tener elementos asociados a un registro no existente. Esto generará un error tanto si intentamos ingresar registros asociados a un registro que no existe, o si intentamos borrar un registro al cual tenemos registros asociados.

Borrado en cascada

Este concepto se refiere al borrado automático de todos los registros asociados a otro registro. Por ejemplo, en las actividades anteriores se podría lograr que al borrar un usuario se borren automáticamente todos los registros de pago asociado. Sin embargo… ¡es muy probable que NO queramos eso! Veremos, entonces, un ejemplo distinto.

Hemos visto que la sintaxis para agregar una clave foránea es la siguiente:

```
[CONSTRAINT fk_name]
   FOREIGN KEY(fk_columns)
   REFERENCES parent_table(parent_key_columns)
   [ON DELETE delete_action]
   [ON UPDATE update_action]
```

Mencionamos también que los elementos en corchetes eran opcionales. La primera línea `constraint fk_name` sirve para nombrar la clave, pero se ocupa un nombre por defecto si la omitimos. Veremos ahora que en la cuarta línea podemos escoger la acción CASCADE luego de ON DELETE. Veremos el resultado.

Actividad guiada N° 5

1. Crearemos las tablas posts y comentarios; cada comentario estará asociado a un post. Al borrar los posts borraremos los comentarios en cascada.

```
CREATE TABLE posts (
  "id" Integer,
  "title" Varchar(255),
  "content" text,
  PRIMARY KEY ("id")
);
```

```
CREATE TABLE comments (
  "id" Integer,
  "content" Varchar(255),
  "post_id" Integer,
  PRIMARY KEY ("id"),
    FOREIGN KEY ("post_id")
    REFERENCES posts ("id")
    ON DELETE CASCADE /* Con esto los datos se borrarán en cascada
automáticamente */
);
```

2. Vamos a insertar solo 1 post y 3 comentarios. Al borrar el post veremos que se borran automáticamente los 3 comentarios.

```
INSERT INTO posts VALUES (1, 'Post1', 'Lorem Ipsum');
```

```
INSERT INTO comments VALUES (1, 'Comentario 1', 1),
(2, 'Comentario 2' ,1),
(3,  'Comentario 3', 1);
```

3. Luego borramos el post.

```
DELETE FROM posts WHERE id = 1
```

Si seleccionamos todos los comentarios veremos que ya no hay ninguno, pues pertenecían todos al primer post y fueron borrados en cascada al borrar este.

Agregar borrado en cascada a una clave foránea existente

Para añadir borrado en cascada a una clave foránea ya existente, deberemos primero borrar la clave y crearla de nuevo. Si no conocemos el nombre de la clave podemos ejecutar el siguiente comando en el cliente de psql.

```
\d nombre_tabla
```

Y luego:

```
ALTER TABLE tabla DROP CONSTRAINT nombre_constraint,  ADD FOREIGN KEY
(fk) REFERENCES tabla(pk) ON DELETE CASCADE;
```

Ideas fuerza

- El borrado en cascada nos permite borrar registros asociados automáticamente.
- Depende de cada negocio y sus reglas si se debe realizar o no un borrado automático.

Otras Restricciones

Así como podemos implementar restricciones de clave primaría y de clave foránea, también es posible implementar otras restricciones.

- Not Null
- Unique
- Check

NOT NULL

Para añadir una restricción NOT NULL al crear una tabla nueva:

```
CREATE TABLE nombre_tabla(
   ...
   nombre_columna tipo_de_dato NOT NULL,
   ...
);
```

Para añadir una restricción NOT NULL a una columna de una tabla existente:

```
ALTER TABLE nombre_tabla ALTER COLUMN nombre_columna SET NOT NULL;
```

Al intentar insertar un valor nulo, ya sea directamente o porque no lo especificamos, obtendremos un error.

UNIQUE

Para añadir una restricción Unique en una tabla nueva:

```
CREATE TABLE nombre_tabla (
      ...
      nombre_columna tipo_de_dato,
        UNIQUE(nombre_columna),
      ...
);
```

Para añadir una restricción Unique a una columna de una tabla existente:

```
ALTER TABLE nombre_tabla
ADD CONSTRAINT nombre_restriccion UNIQUE (nombre_columna);
```

CHECKS

Los checks nos permiten agregar restricciones con base en alguna regla específica, por ejemplo, que un valor sea mayor que cero o que otro número, o que tenga que ser distinto de cierto texto.

```
CREATE TABLE productos(
"nombre" VARCHAR(200),
"stock" INTEGER CHECK (stock >= 0)
);
```

Lo probamos ingresando un valor negativo.

```
INSERT INTO productos values ('p1', -1);
```

```
ERROR:  new row for relation "productos" violates check constraint
"productos_stock_check"
DETAIL:  Failing row contains (p1, -1).
```

Modelos de datos

En ocasiones nos entregarán un diagrama del modelo de datos en lugar de una tabla. Dentro de los modelos de entidad relación (ER) existen 3 tipos de modelos más conocidos:

- Modelo conceptual
- Modelo lógico
- Modelo físico

Las diferencias entre ellos tienen que ver con lo que buscan lograr y comunicar:

- El modelo conceptual busca modelar adecuadamente las entidades y atributos.
- El modelo lógico define entidades transaccionales y operativas.
- Y el modelo físico ya presenta todos los detalles necesarios para la implementación del modelo de datos en un base de datos del motor escogido. Por lo mismo, el modelo físico es el que más se asemeja a su implementación final.

Modelo físico

El modelo físico de datos representa la forma en que se construirá el modelo de la base de datos. Aquí se especifican las tablas, columnas, claves primarias y foráneas, así como otras restricciones.

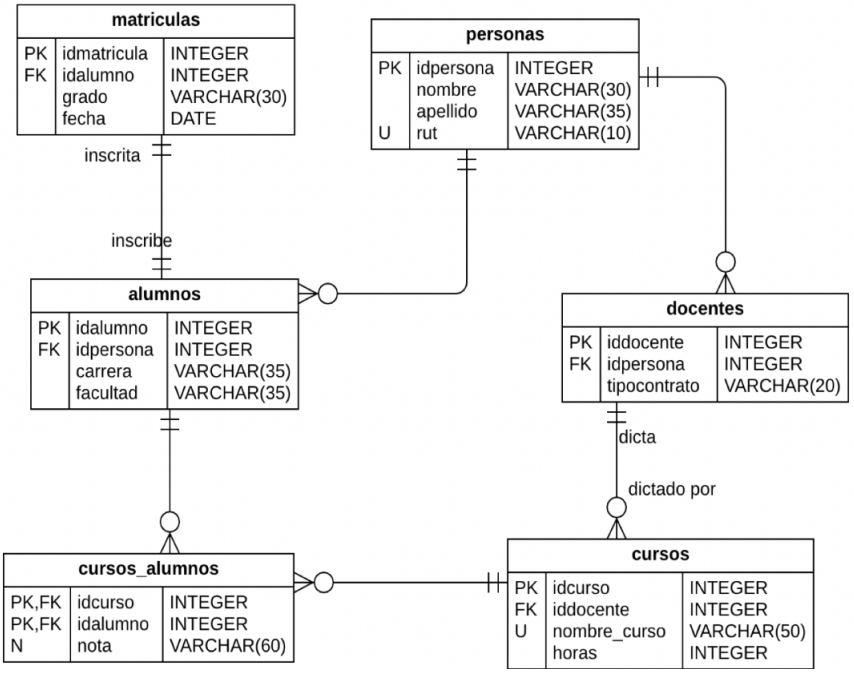


Imagen 1. Ejemplo de diagrama de modelo de datos
Fuente: Ebook: El camino de los datos a la información

En términos prácticos, es un diagrama que nos permite comunicar de forma gráfica nuestro modelo de datos. Las relaciones entre tablas se indican con una línea cuyas terminaciones indican el tipo de relación que existe.
A este tipo de notación de relaciones se le conoce como **crow foot** (pies de cuervo).

Tipos de Relaciones

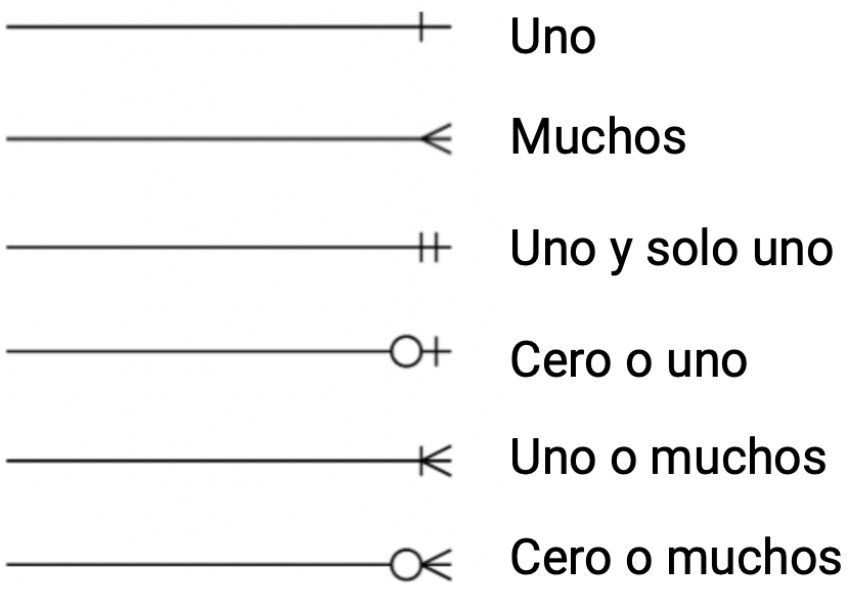


Imagen 2. Relaciones en la notación crowfoot.
Fuente: Elaboración propia.

Nos enfocaremos en las relaciones sin indicar si puede ser cero o no.

Cardinalidad

Hablaremos de cardinalidad cuando hablemos de **qué tipo de relación** hay entre dos tablas. Principalmente veremos 3 casos:

- 1 : 1 (Uno a uno)
- 1 : N (Uno a muchos)
- N : N (Muchos a muchos)

Las más utilizadas son las tablas de 1 a N y las de N a N. Una tabla de 1 a N se implementa directamente tal como hemos hecho hasta ahora.

Ejemplo de relación de 1 a N

Las relaciones 1 a N indican que un elemento de una tabla puede estar asociado a muchos elementos de otra tabla.

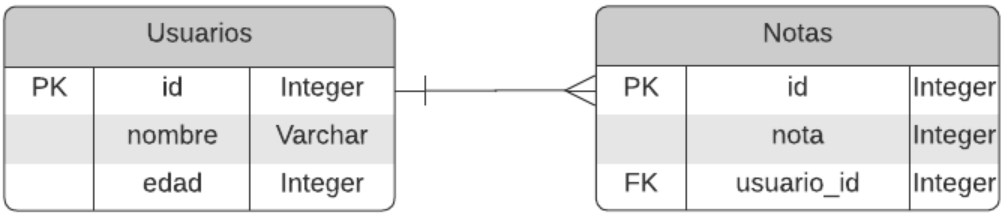


Imagen 3. Ejemplo de relación 1 a N
Fuente: Elaboración propia.

Por ejemplo, en este diagrama se indica que un usuario puede tener múltiples notas, pero una nota en particular le pertenece solo a un estudiante. Conviene aclarar que, si bien decimos "muchas", es posible que un estudiante tenga solo una nota. Lo importante es que **puede** tener más de una.

Relaciones N a N

Las relaciones N a N indican que un elemento de una tabla A puede estar asociado a múltiples elementos de la tabla B, y un elemento de la tabla B puede estar asociado a múltiples elementos de la tabla A.

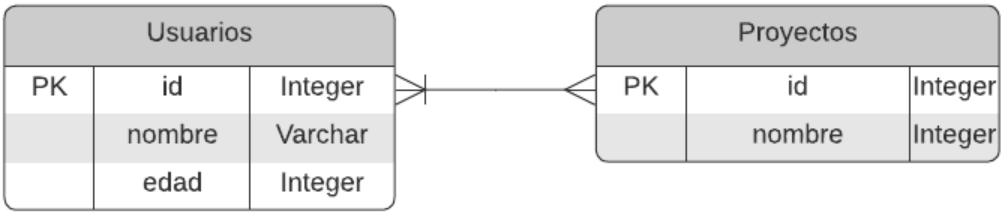


Imagen 4. Ejemplo de relación N a N
Fuente: Elaboración propia.

Por ejemplo, en una empresa un trabajador puede participar en varios proyectos, y en un proyecto pueden participar varios trabajadores. Así, Javiera puede participar en el proyecto1 y en el proyecto2, y en el proyecto1 pueden participar también Francisca, Javiera y Rodrigo. Como en el caso anterior, se trata de una posibilidad y no de una obligación.

Implementando un modelo N a N

Un modelo N a N no se puede implementar directamente en una base de datos relacional, pero podemos hacerlo creando una tabla intermedia.

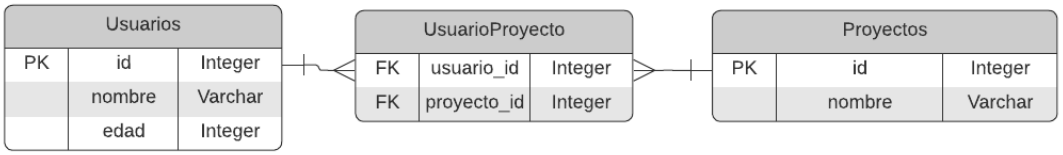


Imagen 5. Implementación de una relación N a N en dos relaciones 1 a N
Fuente: Elaboración propia.

La lógica de la tabla intermedia siempre es igual: entre 2 tablas de N a N agregamos una tabla que contiene las FK de las otras dos tablas. La tabla intermedia siempre queda del lado de "muchos".

Una vez que el modelo está presentado así es más sencillo integrarlo en SQL.

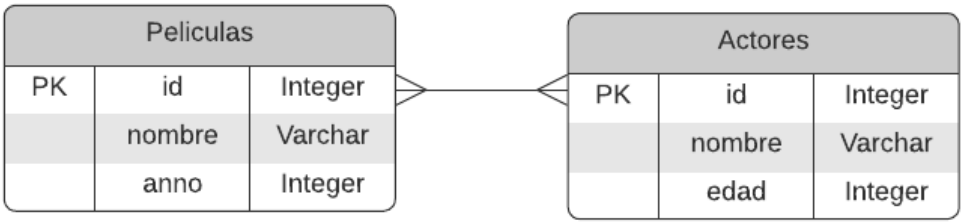
```
CREATE TABLE "Usuarios" (
  "id" Integer,
  "nombre" Varchar,
  "edad" Integer,
  PRIMARY KEY ("id")
);
```

```
CREATE TABLE "UsuarioProyecto" (
  "usuario_id" Integer,
  "proyecto_id" Integer,
    FOREIGN KEY ("usuario_id")
      REFERENCES "Usuarios"("id"),
    FOREIGN KEY ("usuario_id")
      REFERENCES "Proyectos"("id")
);
```

```
CREATE TABLE "Proyectos" (
  "id" Integer,
  "nombre" Varchar,
  PRIMARY KEY ("id")
);
```

Actividad guiada N° 6

1. Identifica el tipo de relación.



2. Dibuja el diagrama utilizando 3 tablas.
3. Escribe el código en SQL para crear las tablas.
4. Ingresa 2 películas y 2 actores en cada película.
5. Selecciona todas las películas junto con todos los actores.

Ideas fuerza

Un modelo de datos dice qué se puede hacer y qué no se puede hacer en un sistema. Por lo mismo, modelar es un aspecto fundamental al construir aplicaciones. En esta unidad nos enfocaremos en implementar modelos existentes, no a modelarlos.

Normalización de datos

La normalización es un proceso de eliminación de redundancia en una base de datos, cuyo objetivo es prevenir inconsistencias, y consta de etapas muy importantes que reciben el nombre de formas normales.

Verifiquemos la importancia de la normalización a partir de un análisis de caso, donde tenemos una tabla clientes con los campos cliente_id, nombre, teléfono, código_país.

- Un cliente puede tener 1 a N números telefónicos.
- Donde N corresponde a la cantidad y dicho valor es variable.

Cliente_id	Nombre	Teléfonos	Código_país
1	Francisco	+584121111111	+58 Venezuela
2	Felipe	**+56999999999 +56988888888 (Aquí tenemos datos repetidos)**	+56 Chile
3	Marcos	+56977777777	+56 Chile

Para que una tabla se encuentre normalizada debe procurar:

- Cada campo o atributo deben ser atómicos, es decir, debe contener un único valor.
- No puede haber grupos repetitivos.
- Debe existir un identificador único.
- Cada atributo debe depender de la clave primaria.

Nota: Los grupos repetitivos se refieren a la posibilidad de que un campo pueda tener 2 o más valores para la misma fila.

Con la tabla anterior, podríamos pensar en agregar un campo llamado teléfono 2.

Cliente_id	Nombre	Teléfono1	Teléfono2	Código	País
1	Francisco	+584121111111		+58	Venezuela
2	Felipe	+56999999999	+56988888888	+56	Chile
3	Marcos	+56977777777		+56	Chile

Esta no sería una solución óptima, recordemos que un cliente puede tener N cantidad de números, en este sentido, la tabla tendría N cantidad de campos asociados a teléfonos.
Para solucionar este problema, lo más idóneo es crear múltiples tablas en relacionar los datos a través de claves primarias y foráneas.

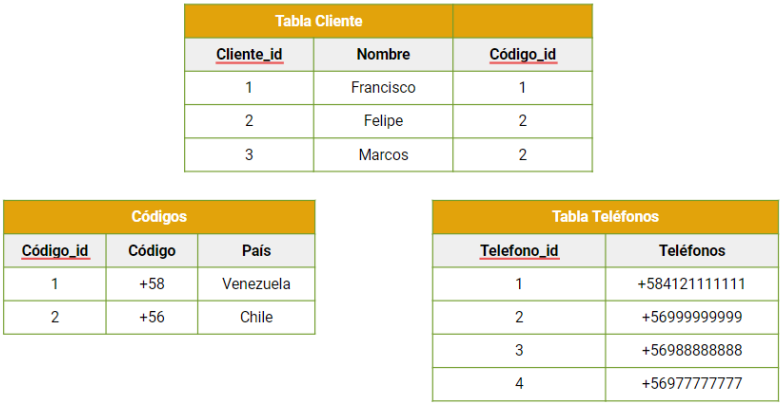


Tabla Cliente

Cliente_id	Nombre	Código_id
1	Francisco	1
2	Felipe	2
3	Marcos	2

Códigos

Código_id	Código	País
1	+58	Venezuela
2	+56	Chile

Tabla Teléfonos

Telefono_id	Teléfonos
1	+584121111111
2	+56999999999
3	+56988888888
4	+56977777777

Imagen 6. Normalización 2FN
Fuente: Desafío Latam

En la imagen anterior hay una distribución más óptima de las tablas:

1. Tenemos una tabla códigos.
2. Tenemos una tabla que almacena los teléfonos del cliente.
3. Tenemos una tabla intermedia llamada cliente que será el puente de conexión entre los códigos y los teléfonos.

Etapas de la normalización

- **1FN:** Para que una tabla sea normalizada deben existir datos atómicos, es decir, tendremos valores únicos y no repetitivos. Además, debe haber un identificador único o clave primaria.
- **2FN:** Debe cumplirse los requerimientos de la 1FN y además, cada atributo de la tabla debe depender de la clave primaria.
- **3FN:** Debe cumplirse la 1FN y la 2FN y además, los atributos que dependen de manera parcial de la clave primaria deben ser eliminados o almacenados en una nueva entidad.

Preguntas de preparación para una entrevista laboral a partir del módulo SQL

- ¿Cuál es la diferencia entre un left join y un inner join?
- ¿Cuál es la diferencia entre un left join y un full join?
- ¿Qué es un constraint?
- ¿Puede una clave primaria ser nula?
- ¿Qué entendemos por integridad de datos?
- ¿Es equivalente un valor nulo a uno falso?
- ¿Es equivalente un valor nulo a un string vacío?
- ¿Para qué sirve que una columna sea serial?

Temas interesantes para profundizar en bases de datos

- Modelamiento de problemas.
- Formas normales, normalización y desnormalización
- Procedimientos almacenados
- Índices
    - Ventajas y desventajas de agregar índices.
    - El plan de consultas.
    - Explain y analize.
- Vistas y vistas materializadas.
- Transacciones.

¡Continúa aprendiendo y practicando!