Bases de datos

Tutorías Ex. de Título 2024-2

Universidad Diego Portales
Prof. Víctor Reyes R.

Objetivos

- Proceso inicial de diseño de una BD relacional.
  - Modelo E-R
  - Modelo Relacional
- Proceso de normalización.
- Consultas a la BD relacional: SQL

Preguntas previas

- ¿De dónde nace una BD? ¿Son datos agrupados de manera aleatoria?
- Supongamos que creamos una BD:
  - ¿Qué pasaría si nos faltan incluir atributos/características?
  - ¿Qué pasa si existe mucha redundancia de los datos?

Preguntas previas

Proceso de creación de una BD

- Por tanto debemos seguir una “metodología” (o proceso) para:
  - Diseño conceptual (modelo E-R)
  - Diseño lógico (modelo relacional)
  - Diseño físico (acá interviene SQL)

Modelo E-R

- El primer paso es el modelo conceptual → Modelo E-R
- Modelamiento: proceso de plasmar el problema en términos abstractos, de modo que sean representables y luegos manejables.
- Para lograr lo anterior, es que debemos identificar los elementos involucrados:
  - Lo primero que debemos hacer es identificar las **entidades**.
- **Entidad**: Elemento del mundo real con existencia independiente. Puede tener existencia física o conceptual. Cada entidad tiene propiedad o atributos que nos interesa almacenar.

Ejemplo práctico: ¿Cuáles serían las entidades?

Luego de finalizar la carrera de Ingeniería Civil en Informática y Telecomunicaciones en la prestigiosa UDP, Bonifacio es llamado por una compañía de desarrollo de videojuegos en Kioto, Japón. Al conversar con el presidente de la compañía, el señor Furukawa, él le cuenta lo siguiente:

- La compañía está organizada en departamentos. Cada uno tiene nombre único, número único y un empleado que los dirige. Nos interesa además la fecha que empezó a dirigirlo.
- Cada departamento controla una serie de proyectos de desarrollo de videojuegos, como por ejemplo el Metroid Prime 4, dice Furukawa. Cada proyecto tiene un nombre y número únicos.
- De cada empleado de la compañía nos interesa el nombre y apellido, ID (como el RUT), dirección, teléfono, sueldo y fecha de nacimiento. Todo empleado estará asignado a un departamento y tendrá un supervisor. Puede trabajar en más de un proyecto, y trabajará un determinado número de horas a la semana en cada proyecto.

Modelo E-R

- Las entidades generalmente se relacionan de alguna forma.
- El siguiente paso es identificar las **relaciones** o **vínculos**.
- Relación/vínculo: Corresponde a una asociación o correspondencia entre las entidades. Cada relación tiene un nombre, cardinalidad, grado y puede tener también atributos.

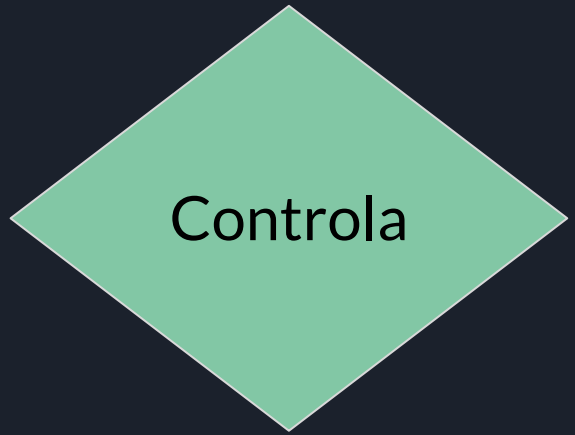
Ejemplo práctico: ¿Cuáles serían los vínculos?

Luego de finalizar la carrera de Ingeniería Civil en Informática y Telecomunicaciones en la prestigiosa UDP, Bonifacio es llamado por una compañía de desarrollo de videojuegos en Kioto, Japón. Al conversar con el presidente de la compañía, el señor Furukawa, él le cuenta lo siguiente:

- La compañía está organizada en departamentos. Cada uno tiene nombre único, número único y un empleado que los dirige. Nos interesa además la fecha que empezó a dirigirlo.
- Cada departamento controla una serie de proyectos de desarrollo de videojuegos, como por ejemplo el Metroid Prime 4, dice Furukawa. Cada proyecto tiene un nombre y número únicos.
- De cada empleado de la compañía nos interesa el nombre y apellido, ID (como el RUT), dirección, teléfono, sueldo y fecha de nacimiento. Todo empleado estará asignado a un departamento y tendrá un supervisor. Puede trabajar en más de un proyecto y trabajará un determinado número de horas a la semana en cada proyecto.

Notación del diagrama: E-R

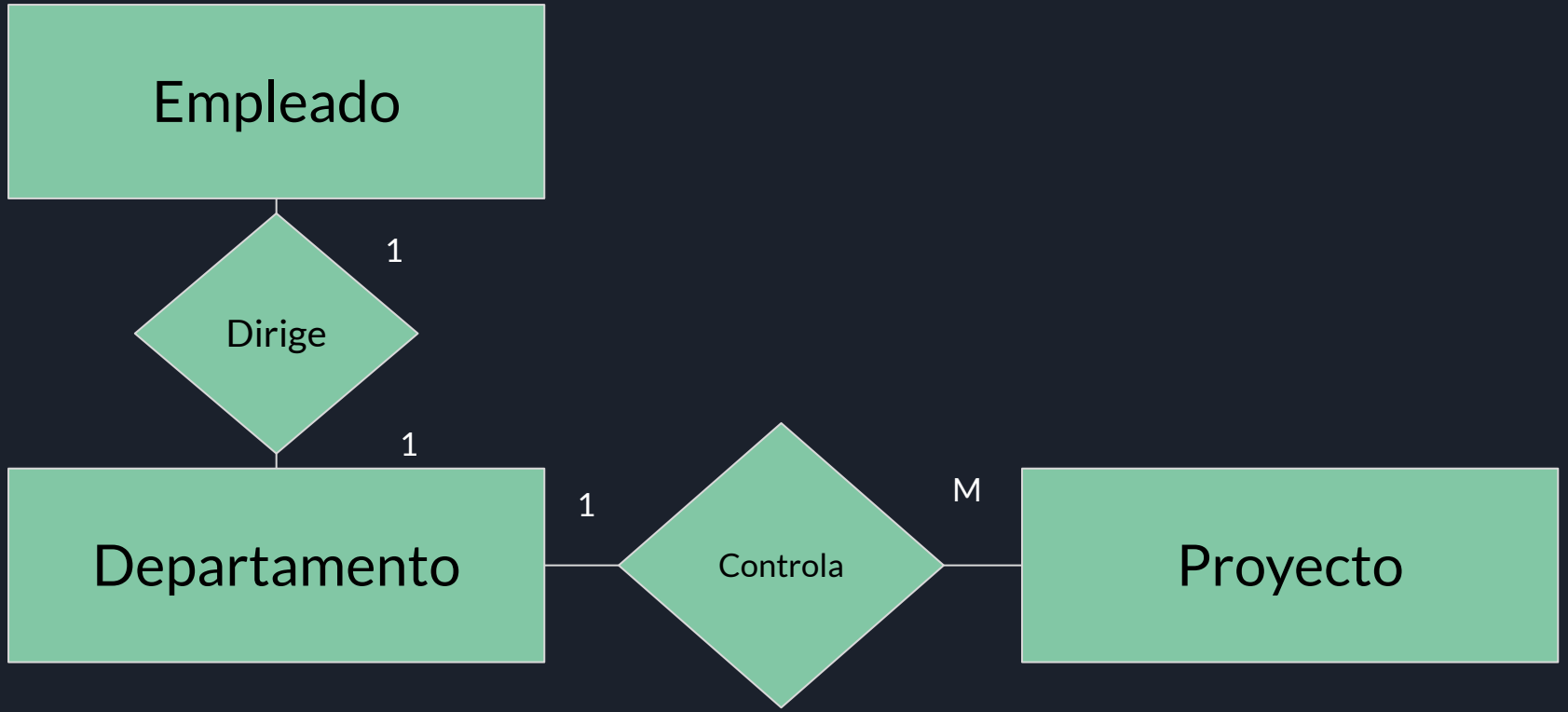
Entidad



Vínculo

Ejemplo: “Un empleado que los dirige”

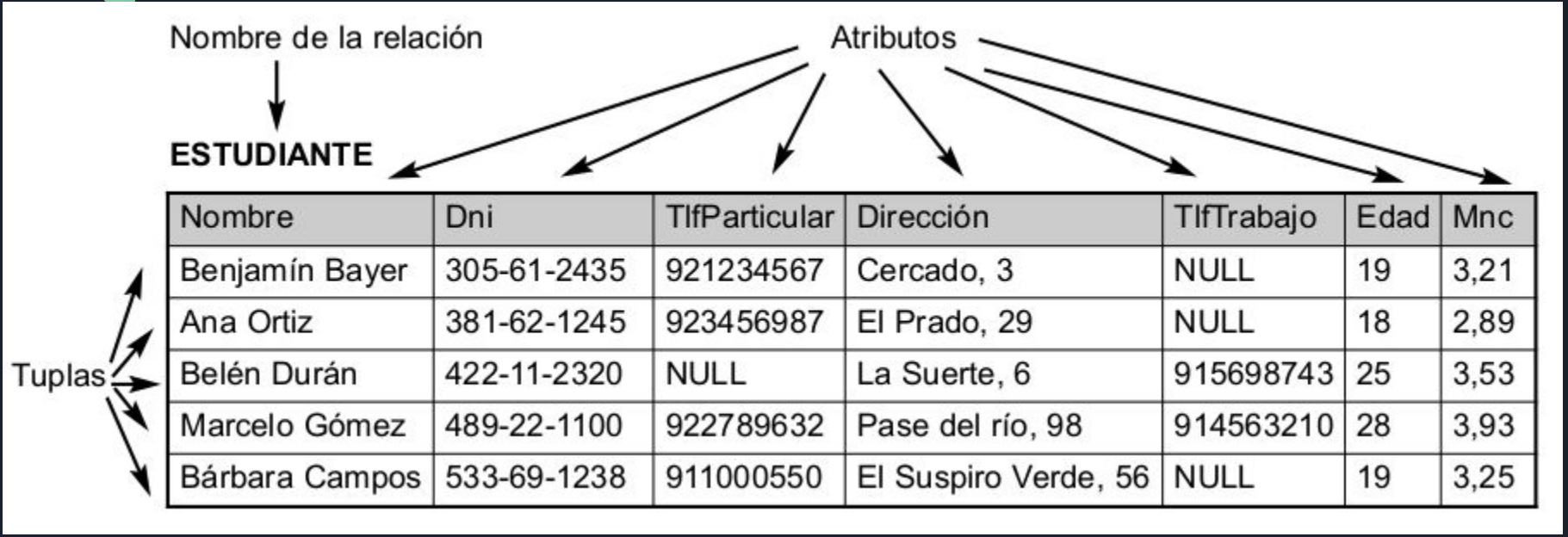
Ejemplo: “Cada departamento controla una serie de proyectos”



Conceptos del Modelo Relacional

- Representación de la BD como una colección de relaciones.
- Cada relación está pensada como una **tabla** de valores.
- Las cabeceras de las columnas se les conoce como **atributos**.
- A cada fila de las tablas se le conoce como **tupla**.
- El tipo de dato que describe los valores que pueden aparecer en cada columna está representado por un dominio de posibles valores

Conceptos del Modelo Relacional



Nombre	Dni	TlfParticular	Dirección	TlfTrabajo	Edad	Mnc
Benjamín Bayer	305-61-2435	921234567	Cercado, 3	NULL	19	3,21
Ana Ortiz	381-62-1245	923456987	El Prado, 29	NULL	18	2,89
Belén Durán	422-11-2320	NULL	La Suerte, 6	915698743	25	3,53
Marcelo Gómez	489-22-1100	922789632	Pase del río, 98	914563210	28	3,93
Bárbara Campos	533-69-1238	911000550	El Suspiro Verde, 56	NULL	19	3,25

La **cardinalidad** es el número de tuplas de la relación/tabla y el **grado** es el número de atributos.

Conceptos de clave

Surgen dos tipos de claves esenciales para el modelo relacional:

- **Clave primaria**: Es aquel (aquellos) atributos que se elige(n) para identificar a las tuplas en una tabla de forma única (ésta no puede tener valores nulos en ninguno de sus atributos).

- **Clave externa o foránea**: Es aquella clave que tiene valores que coinciden con valores de la clave primaria de otra tabla.

Hay más tipos de claves, como: superclave, claves candidatas y claves alternativas.

Restricciones del modelo

- **Valores atómicos**: Cada valor de la tabla debe ser simple.
- **Tuplas distintas**: No pueden existir dos tuplas iguales
- **Clave primaria**: Debe existir una clave que identifique de manera unívoca las tuplas (no se aceptan nulos ni repeticiones)

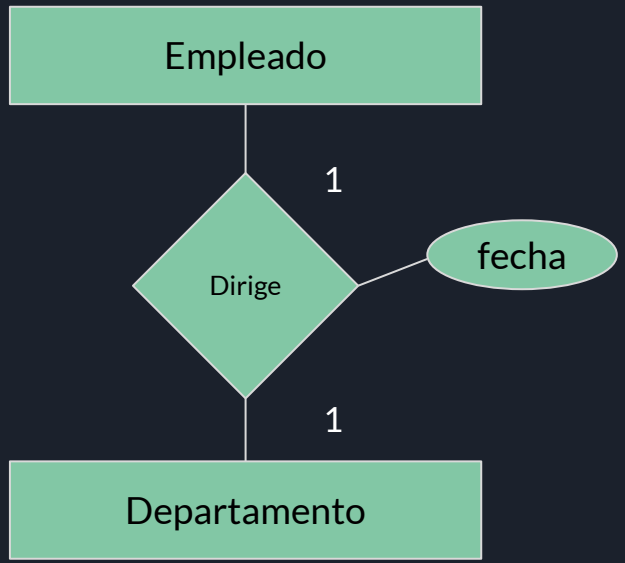
Diagrama relacional

Ahora veremos cómo transformar el diagrama ER a relacional:

1. Toda entidad se transformará en una Tabla, con todos sus atributos. Uno o varios como clave principal y los subrayamos.

2. Para cada vínculo 1-1, agregue los atributos de la clave primaria de una de las entidades asociadas a la otra como clave externa. Es conveniente elegir la entidad que recibe clave con participación total, es decir que todas sus ocurrencias participan en la relación.

Ejemplo



Departamento (<u>CodDep</u> (PK), NombreDep, RUT_Director (FK), Fecha)

Empleado (<u>RUT</u> (PK), Nombre, sueldo)

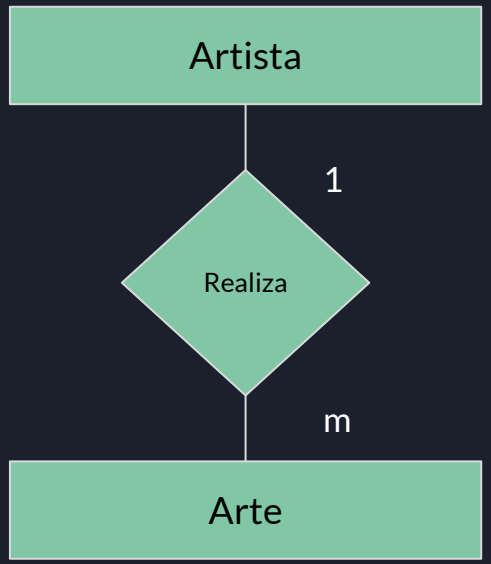
Importante: Si no hay participación completa, entonces se crea una nueva relación (tabla) que tiene por atributos la concatenación de claves primarias de las entidades involucradas.

Diagrama relacional

Ahora veremos cómo transformar el diagrama ER a relacional:

3. Para cada vínculo 1-n, agregue los atributos de la clave primaria de la entidad del lado 1 del vínculo a la relación de la entidad del lado n del vínculo como clave externa.

Ejemplo



Artista (RUT (PK), Nombre, Fecha_nacimiento)

Arte (CodArte (PK), Nombre, año, RUT_Artista (FK))

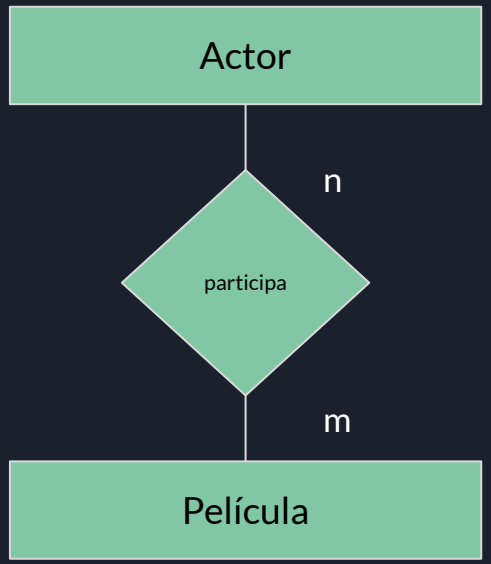
Importante: Si no hay participación completa, entonces se crea una nueva relación (tabla) que tiene por atributos la concatenación de claves primarias de las entidades involucradas.

Diagrama relacional

Ahora veremos cómo transformar el diagrama ER a relacional:

4. Para cada vínculo m-n, cree una nueva relación (tabla) con el nombre del vínculo y que tenga como atributos la concatenación de las claves primarias de ambas entidades involucradas. Esa concatenación será la clave primaria de la relación (tabla) y cada uno de los conjuntos originales será clave externa hacia las relaciones (tablas) de las entidades que conforman el vínculo.

Ejemplo



Actor (RUT (PK), Nombre, Fecha_nacimiento)

Película (CodPeli (PK), Nombre, año)

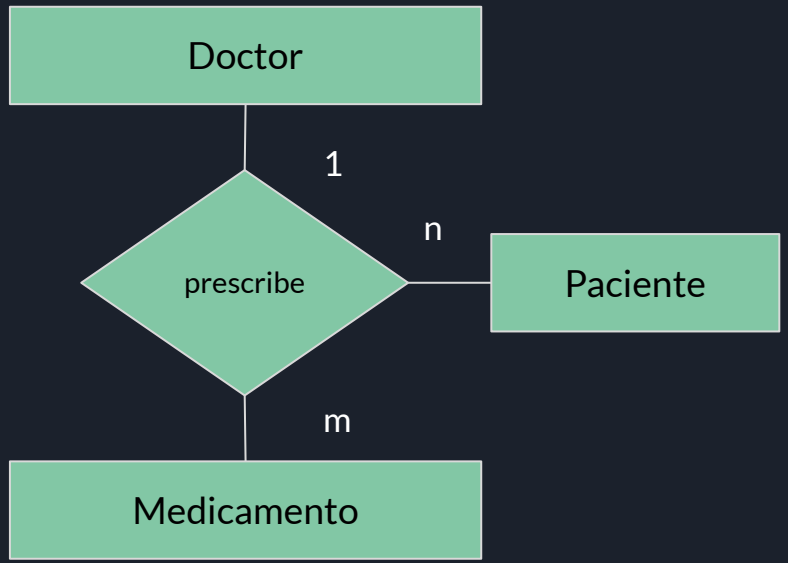
Participa (RUT (PK,FK), CodPeli (PK,FK))

Diagrama relacional

Ahora veremos cómo transformar el diagrama ER a relacional:

4. Para cada vínculo n-ario, cree una nueva relación (tabla) con el nombre del vínculo y que tenga como atributos la concatenación de las claves primarias de todas las entidades involucradas. Esa concatenación será la clave primaria de la relación (tabla) y cada una de los conjuntos originales será clave externa hacia las relaciones (tablas) de las entidades que conforman el vínculo.

Ejemplo



Doctor(RUT (PK), Nombre, Apellido, Fecha_nacimiento)

Paciente(RUT (PK), Nombre, año)

Medicamento(CodMed (PK), Nombre, año, laboratorio)

Prescribe(RUT_D (PK,FK), CodMed (PK,FK), RUT_P (PK,FK))

Normalización de una BD

- La teoría de la normalización apunta a entregar modelos relaciones de mayor rigurosidad y sujetos a ciertas condiciones.

- Estas condiciones apuntan esencialmente a una optimización del modelo relacional. El cumplimiento de las condiciones muchas veces implica cambios en el esquema relacional obtenido originalmente.

El proceso se resume en organizar la información de la BD para evitar redundancia, problemas de inserción, problemas de actualización y problemas al eliminar datos.

Ejemplo

ID_Trab	Nombre	Apellido	Nombre_Dpto	CodDpto	Tiempo_Trab
101	Martín	Gutiérrez	Alpha	D001	3
101	Martín	Gutiérrez	Beta	D002	1
144	Nicolás	Hidalgo	Gamma	D104	2
220	Víctor	Reyes	Epsilon	D095	5
55	Jonathan	Frez	Delta	D027	1
55	Jonathan	Frez	Mu	D033	1

Ejemplo

ID_Trab	Nombre	Apellido	Nombre_Dpto	CodDpto	Tiempo_Trab
101	**Martín**	**Gutiérrez**	**Alpha**	**D001**	**3**
101	**Martín**	**Gutiérrez**	**Beta**	**D002**	**1**
144	Nicolás	Hidalgo	Gamma	D104	2
220	Víctor	Reyes	Epsilon	D095	5
55	**Jonathan**	**Frez**	**Delta**	**D027**	**1**
55	**Jonathan**	**Frez**	**Mu**	**D033**	**1**

Problemas de la tabla anterior

- **Redundancia**: Por cada ID_Trab, se repite siempre el nombre, apellido.
- **Inserción:** Supongamos que un nuevo empleado entra a la compañía, el cual está en entrenamiento y no pertenece a ningún departamento. Luego no podríamos insertar la información en la tabla si CodDpto no admite nulos.
- **Actualización**: En la tabla tenemos dos filas para Martín y dos filas para Jonathan, pues trabajan en dos departamentos distintos. Si quisiéramos actualizar la información de Jonathan, luego tenemos que actualizar las 2 filas o tendríamos problemas de consistencia.
- **Eliminación**: Supongamos que en algún momento se cierra el departamento D095. Luego eliminar filas que contengan CodDept igual a 095 eliminaría también la información de Victor (una gran pérdida sin lugar a dudas....).

Problemas de la tabla anterior

Si pensamos en algo como lo siguiente:

- Trabajador(ID_Trab(PK),Nombre, Apellido)
- Depto(CodDpto(PK),Nombre_Dpto)
- Trab-Dpto(ID_Trab(PK,FK),CodDpto(PK,FK),Tiempo_trab)

Los problemas anteriores al parecer ya no existen…

Normalización

- La normalización ofrece un proceso sistemático en el cual se eliminan problemas como los que se mostraban anteriormente.

- Dentro de los problemas que estamos citando se encuentra ya sea la actualización de datos o la inserción, en ningún caso la consulta.

- El hecho de hacer estos cambios **pueden penalizar la eficiencia en la consulta** debido a un mayor trabajo en cuanto a la asociación entre tablas debido a la creación de nuevas relaciones.

1NF o Primera forma normal

- La regla de la primera forma normal es que, un atributo (columna) de una tabla no puede almacenar múltiples valores. Estos deben ser solo atómicos. Por ejemplo:

ID_Trab	Nombre	Apellido	Teléfono
101	Víctor	Reyes	935678951 ; 989618551
190	Jonathan	Frez	944444444
144	Martín	Gutiérrez	966666666 ; 966666667

No está en 1NF, pues Víctor y Martín rompen la regla.

1NF o Primera forma normal

ID_Trab	Nombre	Apellido	Teléfono
101	Víctor	Reyes	935678951
101	Víctor	Reyes	989618551
190	Jonathan	Frez	944444444
144	Martín	Gutiérrez	966666666
144	Martín	Gutiérrez	966666667

2NF o Segunda forma normal

- Un esquema está en 2NF o segunda forma normal si:
    - Está en 1NF
    - Todos los atributos no-primos entregan información de la clave completa.

Atributo no-primo: Atributo que no es parte de ninguna clave candidata.

2NF o Segunda forma normal

ID_Prof	Nombre	Apellido	Asignatura
101	Víctor	Reyes	Computación Evolutiva
101	Víctor	Reyes	Aprendizaje Reforzado
190	Martín	Gutiérrez	Computación Evolutiva
144	Nicolás	Hidalgo	Sistemas Distribuidos
144	Nicolás	Hidalgo	Sis. proc de Big Data

2NF o Segunda forma normal

ID_Prof	Nombre	Apellido	Asignatura
101	Víctor	Reyes	Computación Evolutiva
101	Víctor	Reyes	Aprendizaje Reforzado
190	Martín	Gutiérrez	Computación Evolutiva
144	Nicolás	Hidalgo	Sistemas Distribuidos
144	Nicolás	Hidalgo	Sis. proc de Big Data

- La tabla está en 1NF.
- Para 2NF:
  - Clave candidata (ID_Prof , Asignatura)
  - Atributos no primos: Nombre, Apellido

Nombre,Apellido es dependiente del ID_Prof, pero no de la asignatura. Por tanto no está en 2NF.

2NF o Segunda forma normal

ID_Prof	Nombre	Apellido
101	Víctor	Reyes
190	Martín	Gutiérrez
144	Nicolás	Hidalgo

ID_Prof	Asignatura
101	Computación Evolutiva
101	Aprendizaje Reforzado
190	Computación Evolutiva
144	Sistemas Distribuidos
144	Sis. proc de Big Data

Ahora el esquema está en 2NF

3NF o Tercera forma normal

- Un esquema está en 3NF o tercera forma normal si:
    - Está en 2NF
    - Todos los atributos no-primos únicamente entregan información de la clave completa y no de otros atributos.

3NF o Tercera forma normal

ID_Libro	Editorial	País
4259	Del Rey Books	USA
3869	Del Rey Books	USA
4863	Dolmen	España
215	Planeta De Agostini	España
364	Kodansha Comics	USA

- La tabla está en 2NF.
- Para 3NF:
  - Clave candidata (ID_Libro)
  - Atributos no primos: Editorial, País.

País entrega información de la editorial, por lo que no está en 3NF.

3NF o Tercera forma normal

ID_Libro	Editorial
4259	Del Rey Books
3869	Del Rey Books
4863	Dolmen
215	Planeta De Agostini
364	Kodansha Comics

Editorial	País
Del Rey Books	USA
Dolmen	España
Planeta De Agostini	España
Kodansha Comics	USA

Ahora el esquema está en 3NF

SQL

- SQL (por sus siglas en inglés Structured Query Language) es el lenguaje estándar de consulta que se usa hoy en BD relacionales.

- SQL tiene variaciones menores entre distintos DBMS (DataBase Management System). Algunos ejemplos: MySQL, PostgreSQL, Microsoft SQL Server, SQLite, etc…

- Es posible usar el lenguaje para distintos fines:
  - Consulta de datos
  - Creación, modificación e ingreso de datos
  - Administración de la BD: creación de cuentas, perfiles, roles, vistas, etc.

SQL

- Supongamos que ya tenemos llenada nuestra BD, nos interesa extraer información de ella, para lo cual elegimos que datos tomar.

- **SELECT** - **FROM** - **WHERE**: Esta es la sentencia que es la base de todas las consultas de información sobre la BD.

SQL: SELECT-FROM-WHERE

La sentencia consta de tres partes:

- La parte **SELECT** en que se le indica qué atributos se desean consultar. Estos atributos podrían estar asociados a una tabla particular.
- La parte **FROM** cumple la función de especificar de qué tablas se extraen los atributos en cuestión.
- La parte **WHERE** impone la condición lógica, sujeta a la cual vamos a extraer información.

SQL: Ejemplo

Supongamos que tenemos la relación VideoJuego(NombreJuego (PK), Consola (PK), Precio, Género, Año) y Tiene(ID_Jugador(PK,FK) , NombreJuego(PK,FK), Consola (PK,FK), Fecha). Supongamos que queremos conocer los ID de los jugadores que tengan juegos del tipo Soulslike en PC.

- ¿Qué atributos deseamos obtener?
- ¿Desde que tablas deben ser seleccionadas las tuplas?
- ¿Cuál es la condición de selección?

SQL: Ejemplo

Supongamos que tenemos la relación VideoJuego(NombreJuego (PK), Consola (PK), Precio, Género, Año) y Tiene(ID_Jugador(PK,FK) , NombreJuego(PK,FK), Consola (PK,FK), Fecha). Supongamos que queremos conocer los ID de los jugadores que tengan juegos del tipo Soulslike en PC.

- ¿Qué atributos deseamos obtener? ID_Jugador
- ¿Desde que tablas deben ser seleccionadas las tuplas? VideoJuego y Tiene
- ¿Cuál es la condición de selección? NombreJuego y Consola deben ser iguales, Consola= “PC” y Género = “Soulslike”
  Entonces la consulta quedaría: SELECT ID_Jugador FROM Tiene,VideoJuego WHERE Tiene.NombreJuego =VideoJuego.NombreJuego AND Tiene.Consola =VideoJuego.Consola AND VideoJuego.Consola = “PC” AND VideoJuego.Genero = “Soulslike”

SQL: Ejemplo

- Notar que hemos usado conectivos lógicos dentro del WHERE, para unir distintas sentencias lógicas. Estos son:
  - AND : conjunción lógica.
  - OR: disyunción lógica
  - NOT: negación lógica

- Así entonces, ya sabemos que se puede hacer una asociación entre tablas por medio de sus claves externas de forma explícita.

- Veremos asociaciones implícitas un poco más adelante.

- Además, se pueden utilizar las palabras claves UNION, INTERSECT, MINUS entre consultas, donde corresponden respectivamente a la unión, intersección y resta de relaciones.

Esquema

Dado el esquema:

VideoJuego(NombreJuego (PK), Consola (PK), Precio, Género, Año)

Jugador(ID_Jugador (PK), Nombre, Edad, País)

Tiene(ID_Jugador(PK,FK) , NombreJuego(PK,FK), Consola (PK,FK), Fecha)

SQL

- Supongamos que quisiéramos saber la lista de juegos que tienen los jugadores, pensaríamos en algo como:
  - SELECT NombreJuego FROM Tiene
- ¿Problema de la consulta anterior?: Hay repeticiones…
- Para este tipo de consultas utilizamos la palabra clave DISTINCT.
- Nuestra consulta cambiaría a:
  - SELECT DISTINCT NombreJuego FROM Tiene

SQL

- Ahora, supongamos que quisiéramos saber la lista de juegos que salieron entre el 2022 y el 2023, sería algo como:
  - SELECT NombreJuego FROM Juego WHERE Año=“2022” OR Año=“2023”

- Funciona perfectamente, pues la columna está definida como INT. En particular, las fechas pueden ser definidas como DATE. En este caso sigue el formato “año-mes-día”

- Para este tipo de casos usamos la palabra clave BETWEEN, la cual nos establece un rango de valores.

SQL

- Si miramos la tabla Tiene, la fecha está definida como DATE. Si quisiéramos saber los nombres de los juegos, en conjunto al jugador, que fueron comprados entre el 01 de Marzo del 2024 y el 01 de Julio del 2024 tendríamos:

    - SELECT NombreJuego,ID_Jugador FROM Tiene WHERE Fecha BETWEEN “2024-03-01” AND “2024-07-01”

SQL

- Supongamos que ahora queremos buscar la información de un juego en particular en la tabla Tiene…peeeero no nos acordamos bien del nombre del juego, solo que comienza con la palabra Elden

- Cuando ocurre una situación así, se hace necesaria la palabra clave LIKE. Esta palabra clave aplica sobre la parte WHERE de la sentencia.

- LIKE se usa además con los siguientes símbolos: % y _.

- % se refiere a sufijo, prefijo o expresión intermedia. _ se refiere a un carácter intermedio.

SQL

- En el caso anterior, la consulta sería:
  - SELECT * FROM Tiene WHERE NombreJuego LIKE “Elden%”

- Si quisiéramos buscar aquellos juegos que tienen los jugadores, y que tengan el conector “of” en su nombre sería:
  - SELECT * FROM Tiene WHERE NombreJuego Like “%of%”

- Juegos que tienen los jugadores con una ’i’ en la segunda posición del nombre:
  - SELECT * FROM Tiene WHERE NombreJuego Like “_i%”

SQL

- A veces también se quiere hacer una consulta que esté sujeta a que un atributo dentro del WHERE tome un valor dentro de un cierto conjunto de datos particular.

- Por ejemplo, ID de los jugadores de Chile que tengan el Elden Ring, God of War o Hell Divers 2.

- Para esto se usa la palabra clave IN que indica el rango de valores que puede tomar el atributo del WHERE.

- Nuestra consulta quedaría como: SELECT T.ID_Jugador FROM Tiene T,Jugador J WHERE T.ID_Jugador = J.ID_Jugador and Pais = “Chile” and NombreJuego IN (“Elden Ring”, “God of War”, “Hell Divers 2”)
- Es una manera corta de reemplazar varios OR.

SQL

- Además, el IN también puede usar una consulta.
- Por ejemplo la siguiente consulta, ¿qué hace?:
    - SELECT ID_Jugador FROM Tiene WHERE NombreJuego IN (SELECT NombreJuego from VideoJuego WHERE Precio >=44000)

SQL

- Supongamos que quisiéramos tener la lista de jugadores que han jugado los juegos Elden Ring o Persona 5.

- La consulta sería:
  - SELECT DISTINCT J.Nombre FROM Jugador E, Tiene EC WHERE ((EC.ID_Jugador = E.ID_Jugador) AND (EC.NombreJuego=“Elden Ring” OR EC.NombreJuego= “Persona 5”));

- Pero nos gustaría que esta lista estuviera ordenada por Nombre…Para esto debemos usar las palabras claves ORDER BY

SQL

- Las palabras ORDER BY toman el nombre de una columna según la cual se desee ordenar la respuesta a la consulta. Se puede ordenar de forma ascendente o descendente.

- La consulta quedaría entonces como:

    - SELECT DISTINCT J.Nombre FROM Jugador J, Tiene EC WHERE ((EC.ID_Jugador = J.ID_Jugador) AND (EC.NombreJuego=“Elden Ring” OR EC.NombreJuego= “Persona 5”)) ORDER BY J.Nombre ASC;

SQL

- También se pueden ordenar más de una columna, por ejemplo si quisiéramos ordenar de menor a mayor la tabla Curso por cantidad de créditos, y en caso de empate comparar el nombre de manera descendente, tendríamos:

SQL

- Funciones de agregación: funciones que hacen algún tipo de operación aritmética sobre un conjunto de datos (principalmente un atributo).

- Vamos a definir cinco funciones:

    - COUNT
    - MAX
    - MIN
    - AVG
    - SUM

SQL

- Estas funciones se usan en una sentencia SELECT sobre un atributo de columna.
  - COUNT: cuenta el número de instancias del atributo en cuestión. Puede ser combinado con la palabra DISTINCT.
  - MAX: entrega el valor máximo de la columna.
  - MIN: entrega el valor mínimo de la columna.
  - AVG: entrega el valor promedio de la columna.
  - SUM: entrega la suma de los valores de la columna.

SQL

- Ahora bien, ¿Qué pasa si quisiera agrupar los Juegos por Género, para hacer de esa manera una tabla de frecuencia?

- GROUP BY cumple la función de agrupar los atributos de una columna sobre la consulta de modo de poder aplicar sobre esa agrupación alguna función de agregación.

- La consulta quedaría de la siguiente manera: SELECT Género, COUNT(Género) AS Cantidad FROM Juego GROUP BY Género;

SQL

- Por último, supongamos que quisiéramos agregar la condición de sólo desplegar las agrupaciones con 5 juegos o más.
- Para este tipo de consulta, tenemos que incorporar una condición sobre una función de agregación.
- Con este fin, se usa la palabra clave HAVING.
- HAVING define una condición sobre una función aritmética.

SQL

- Entonces, para nuestra consulta lo podríamos hacer como sigue:
    - SELECT Género,COUNT(Género) AS Cantidad FROM Juego GROUP BY Género HAVING COUNT(Género) >= 5;