CERO. Duda de tarea programada.

Hay solo una interfaz de usuario, para administrativo, Y el administrativo puede pagar las factura de cualquier propiedad (la interfaz lo permite), a traves de la cedula del propietario o la propiedad.

Hay 2 formas de pagar, una porque al procesar una fecha de operación desde el XML <pagar numfinca="123" >, si no esta al día en fecha de operación, calcular interese moratorio, agregahr una linea en la factura por este concepto, modificar el total a pagar, cambiar el estado de la factura de pendiente a pagado.

La otra foma de pagar, es que el propietario hace fila en ventanilla, un usuario administrativo en su interfaz de usuario, localiza via cedula o numero de finca, la propiedad, se despliegan las facturas pendientes, y al dar click en boto "pagar", se calcula intereses moratorios, modifica el total a pagar, ese monto se muestra, y se finalmente se confirma el pago para cambiar el estado de la factura. Mas detalles en la especificación.

UNO. Normalización (continuación).

Objetivo principal es asegurar que un diseño físico tiene mínima redundancia y una buena representación respecto de los requerimientos. Como corolario, el diseño protege de problemas de ambigüedad e inconsistencia.

Cuando se desnormaliza o se toleran diseños no normalizados, se pone en riesgo la no-redundancia, la consistencia, el diseño mas ambiguo por lo tanto menos claro, etc…

Mucha de la teoría de la normalización utilizada para construir la representación, basada en algoritmos que manipulan dependencias funcionales, ha sido sustituida por buenas practicas desde el modelado conceptual. Lo cual es el enfoque de algunos libros de bases de datos.

El enfoque actual, o al menos en este curso, es que si se utilizan buenas prácticas en el modelado conceptual, cuando este se traslada al modelo físico, al menos cumplirá la 4ta forma normal.

Para que es útil la normalización:

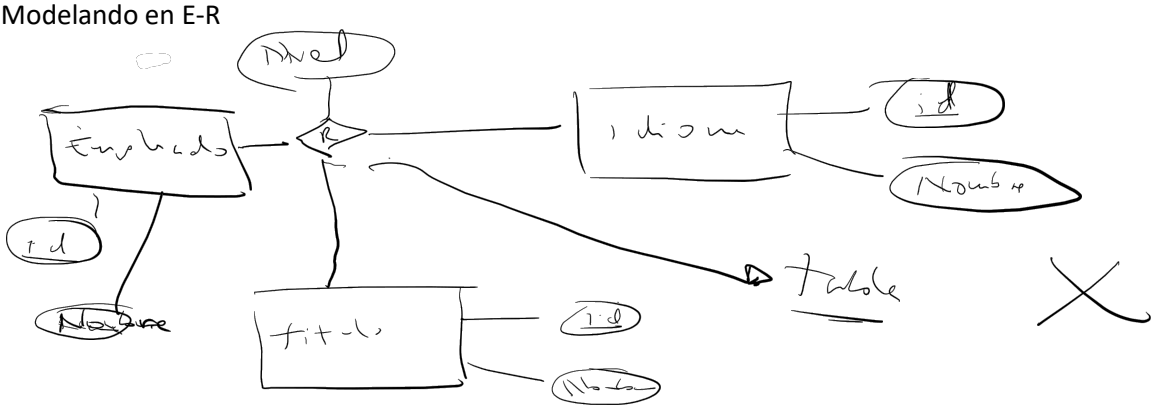
- Para ser conscientes en un diseño, de los riesgos de que el diseño no cumpla con las formas normales. Por ejemplo: una decisión de diseño físico es pasar por alto alguna de los requisitos de 1era forma normal para que el diseño se mantenga simple, aunque se incurren riesgos, por ejemplo pasar por alto que el diseño no debe tener campos variantes.
- Cuando se deben aplicar criterios técnicos, o sea criterios objetivos, para evaluar un diseño, por ejemplos para efectos de auditoria o diagnóstico, etc. Entonces la teoría de normalización crea esos criterios.

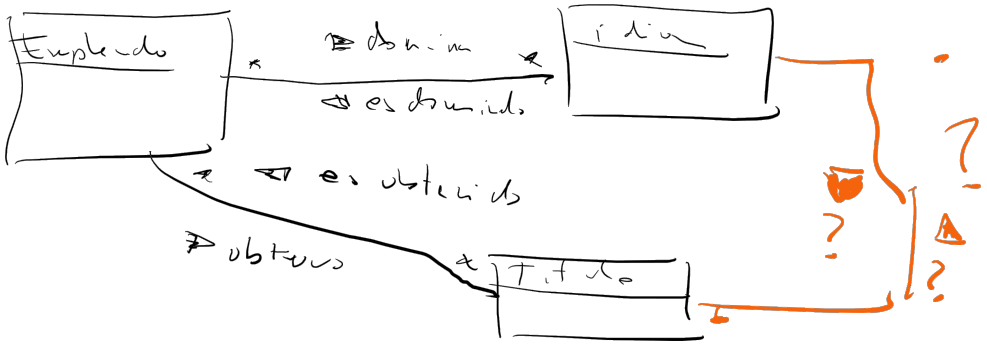
Empleado (idPersona PK, IdTitulo PK, IdIdiomaQueDomina PK, GradoTitulo, NivelUsoIdioma) …. PK compuesta por 3 ids) cumple 3era forma normal pero no 4ta forma normal.

.

Hay un problema conceptual, que es que Idioma no se relacion con Titulos (aunque hay casos especiales; para Doctor en Relaciones Internacionales debe hablar al menos ingles si su lengua materna es español, aunque esto es un caso especial que debería modelarse a traves de subclases)

Modelando en E-R





Un diseño está en 1NF si:

- Tiene PK
- No tiene atributos repetitivos
- No tiene atributos compuestos
- No tiene atributos rellenos
- No tiene atributos variantes.

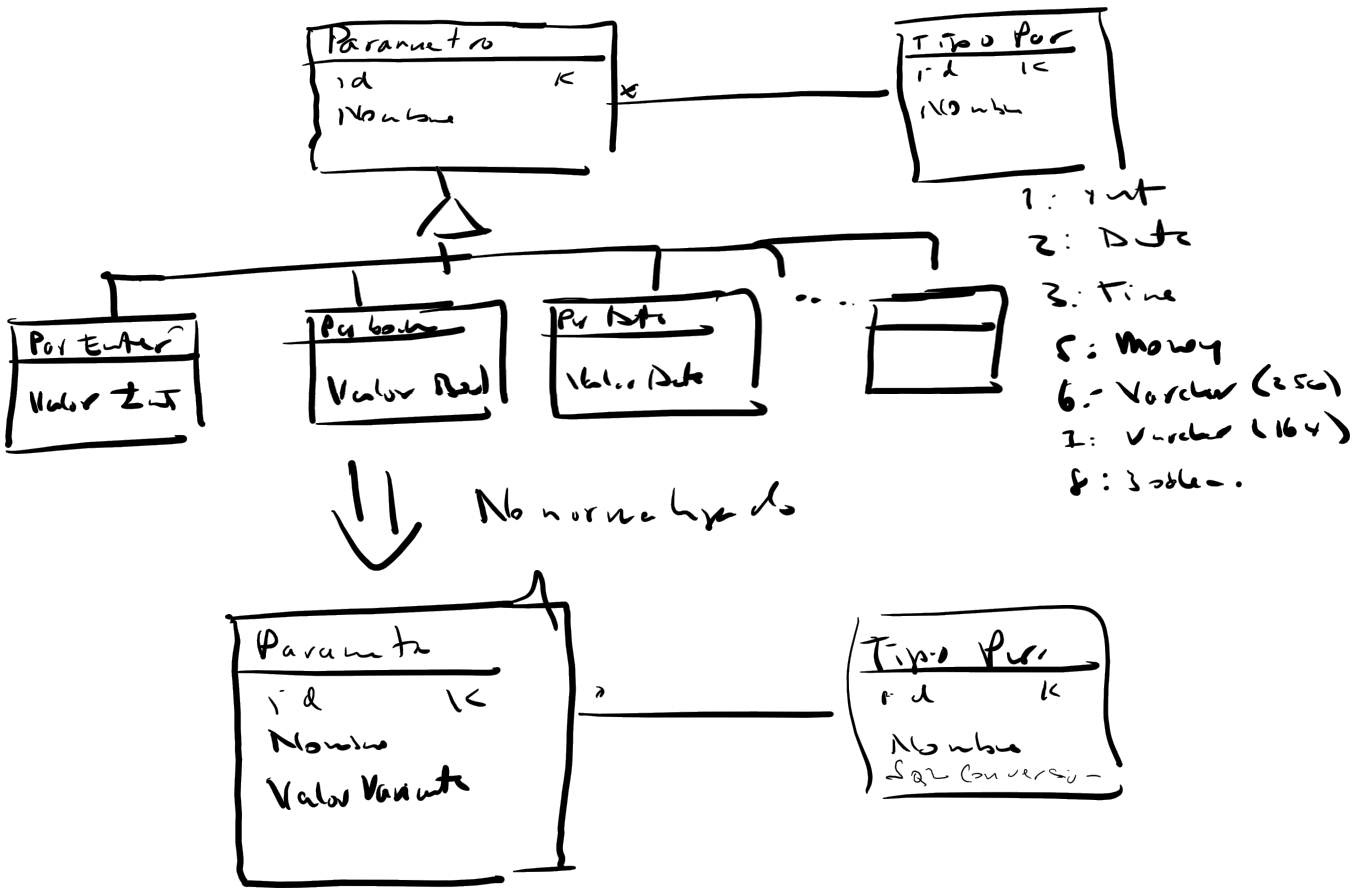
Un diseño que represente valores de configuración en una aplicación, requiere el identificador el valor de configuración, un nombre, y el valor de configuracion en si, que puede ser de cualquier tipo: boolean, date, integer, float, varchar ….

Nombre	Tipo	Valor
Nombre de la empresa	Varchar (64)	Empresa El Olvido
Ultimo consecutivo físico de las facturas	Int	5325

impreso (código de la factura), NO el Id.		
Tipo de cambio dólar compra	float	505.10
Cuenta de correo que envia comprobantes	Varchar (128)	facturasEmpresaX @xx,com
Dia de cierre del negocio	char	K
Dia de cierre mensual	Int	5

El diseño de los settings de una aplicación o sus propiedades, en general son de diferente naturaleza.

Estos requerimientos son similares al modelado de CC para la tarea programada.



Programáticamente, dependiendo del Tipo Parametro, hacemos una conversión para devolver un valor.

```
Create Procedure LeaParametroEnteroEnDisenoNormalizado @inId int, @outResult int, @outResultCode int output
AS
BEGIN
        Set @outResultCode = 0
        Select @outResult=P.ValorEntero
        from dbo.ParametroEntero P        -- tengo que ir a una de N tablas (subclase de Parametro)
        Where P.Id=@inId
        If (@Result is NULL)
        Begin
                Set @outResultCode = 50008
        end
END
```

El SP para leer un parámetro Date, lo buscaría en la tabla db.ParametroDate

Usando una sola tabla, con valor del parámetro variante, todos los parámetros se buscan en una sola tabla, y como el valor esta representado en un campo variante (en el caso de MSSQL es varchar(2000)).

```
Create Procedure LeaParametroEnteroEnDisenoNormalizado @inId int, @outResult int, @outResultCode int output
AS
BEGIN
        Set @outResultCode = 0
        Select @outResult=P.ValorEntero
        from dbo.ParametroEntero P        -- tengo que ir a una de N tablas (subclase de Parametro)
        Where P.Id=@inId
        If (@Result is NULL)
        Begin
                Set @outResultCode = 50008
        end
END
```

EL SP que obtiene parámetros de tipo date, usara Select @outResult=convert(date, P.ValorVariante)

En capa logica debe hacerse la selección del tipo de parámetro que se va a leer para llamar al SP que corresponde.

Desde el punto de vista de la normalización (que mejora la consistencia, reduce la ambigüedad, mejora la representación, etc, etc), la claridad (que el diseño es autoevidente o autoexplicativo), etc. El mejor diseño es el normalizado, aunque es mas complejo pues requiere de mas tablas.

```
Create Procedure LeaParametroEnteroEnNODisenoNormalizado @inId int, @outResult int, @outResultCode int output
AS
BEGIN
        Set @outResultCode = 0
        Select @outResult=convert(INT, P.ValorVariante)
        from dbo.Parametro P       -- solo tengo que recordar el nombre de una tabla
        Where P.Id=@inId
        If (@Result is NULL)
        Begin
                Set @outResultCode = 50008
        end
END
```

El diseño no normalizado, representa todo en una sola tabla, hay que tener cuidado en la programación en hacer el convert que corresponde, desde el punto de vista de la complejidad del modelo y de la programación es ventajoso.

Una programación ´sucia´, pero rápida y centralizada haría lo siguiente:

```
Create Procedure LeaParametroCualquierTipoDisenoNormalizado (@inId int, @outResultCode int output
AS
BEGIN
        -- ESTO NO ES CORRECTO, AUNQUE DA UNA IDEA DE SOBREDISEÑO APLICADO A PROGRAMACION
        Set @outResultCode = 0
        Select                                  -- RETORNA UN RECORDSET
                case when P.idTipoParametro = 1 then convert(INT, P.ValorVariante)   -- integer
                     when P.idTipoParametro = 2 then convert(DATE, P.ValorVariante)   -- DATE
                     when P.idTipoParametro = 3 then convert(BIT, P.ValorVariante)   -- BOOLEAN
                ....
                END

        from dbo.Parametro P
        Where P.Id=@inId

        If (@@ROWCOUNT<1)
        Begin
                Set @outResultCode = 50008   -- no encontró el parametro
        end
END
```


No es buena programación pues el valor del record set es variante, y se delega a la capa logica la interpretación del tipo de record set. Como aspecto positivo, hay un único SP para leer los parámetros, mientras que en las versiones arriba, hay un SP para leer entero, otro para leer fecha, otro para leer boolean.

```
Create Procedure LeaParametroCualquierTipoDisenoNONormalizado (@inId int, @outResultCode int output
AS
BEGIN
        -- ESTO NO ES CORRECTO, AUNQUE DA UNA IDEA DE SOBREDISEÑO APLICADO A PROGRAMACION
        Set @outResultCode = 0
        Select                                  -- RETORNA UN RECORDSET
                case when P.idTipoParametro = 1 then convert(INT, P.ValorVariante)   -- integer
                     when P.idTipoParametro = 2 then convert(DATE, P.ValorVariante)   -- DATE
                     when P.idTipoParametro = 3 then convert(BIT, P.ValorVariante)   -- BOOLEAN
                ....
                END

        from dbo.Parametro P
        Where P.Id=@inId

        If (@@ROWCOUNT<1)
        Begin
                Set @outResultCode = 50008   -- no encontró el parametro
        end
END
```


Este código permite tener centralizado la lectura de parámetros, un solo SP, pero es oscuro.

Desde el punto de la rapidez de la programación, es mejor esta ultima version, desde el punto de vista de la claridad son mejores las previas.

Dilema ético:

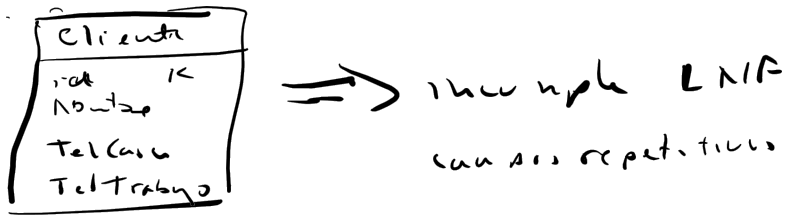
D1. Si hago las cosas de la mejor manera (buscando perfección), hay riesgo de que el entregable no esté listo para el jueves a las 10am, que hay que hacer el demo.

D2. Si hago las cosas, no elegantemente, favoreciendo la rapidez, y no preocupándome por consistencia, el entregable está listo, aunque probablemente habrá que recodificar y arreglar cosas despues.

Para el profe, el dilema ético se resuelve en D1, para reducir la posibilidad de no-entrega, que es lo poder que puede suceder.

Con respecto de otras formas normales, con frecuencia, conscientemente se traiciona alguna forma normal, para evitar particionamiento de tablas, lo cual conlleva a que en las consultas se

reducen la cantidad de joins y con ellos se ejecutan mas rápidamente. Tambien se traiciona normalización, agregando campos redundantes para que las consultas se ejecuten rápidamente.



La alternativa es diseñar adaptando el patrón de contactación: Cliente, teléfonos, uso de teléfono, ClienteXTelefonoActual, ClienteXTelefonoHistorico,….

El criterio de normalización se ha hecho relativo, debido a que en tiempos de internet y aplicaciones móviles, y el auge de app no transaccionales, hace que el principal criterio de diseño sea la velocidad de respuesta, y la flexibilidad (que la app funcione en muchos contextos), con lo cual se sacrifica la normalización.

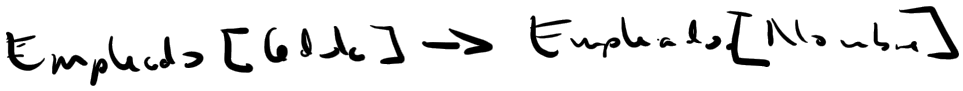
Un diseño está en 2NF: si el diseño está en 1NF y además todos los atributos NO PK dependen funcionalmente de la PK completa.

Aquí introducimos el concepto de dependencia funcional (DF), el cual se anota así A => B, el cual se lee: “B depende funcionalmente de A”, o “A determina B” (esta es mi preferido), donde A y B son conjuntos de atributos que pertenecen a la misma tabla (aunque en términos generales A y B no necesariamente estan en la misma tabla).

Las DF son de naturaleza inyectiva, esto es para un valor en A, solo existe un valor en B, o sea establecen una relacion de 1 a 1,

Con frecuencia, la notación incluye el nombre de tabla:

Empleado.Cedula => Empleado.Nombre es una DF o no lo es, SI es establece una relación inyectiva, o sea 1 a 1, (o sea no es 1 a N), referido a la misma entidad.



Sera que un valor en cedula, mapea un único en nombre?. Una empleado con cedula 100600039 mapea a un único nombre

Una cedula solo mapea un nombre, pues no existen 2 personas que tengan la misma cedula.

E.nombre => E.Cedula es DF? Esto representa un mapeo es 1 a 1? R/ No es!!

Si dos empleados se llaman igual (posible en amazon.com que tienen 1.5 millones de empleados)

Ariel Rojas Rojas (masculino, salvadoreño, de 40 años)

Ariel Rojas Rojas (Femenina, filipina, 21 años)

No es 1 a 1, pues un mismo nombre mapea a 2 o mas identificaciones (cedula de residencia en CR), o sea que es una relacion 1 a N. NO ES UNA DF

FechaNacimiento => Nombre, es DF?. No pues personas con diferente nombre pudieron nacer en la misma fecha, 1 fecha de nacimiento se mapea a muchos nombre

Nombre => FechaNacimiento, es DF? pues 2 o más personas pudieron nacer, llamándose igual pudieron nacer en la misma fecha de nacimientos, siendo una relacion de N a 1

Cedula => fechanacimiento, es DF?, Si lo es, pues una cedula refiere a una sola persona, y una persona solo nace en una fecha, entonces es 1 a 1.

FechaNacimiento => Nombre, dF?, no pues 1 fecha de nacimiento se mapea al nombre de muchas personas que nacieron el mismo día.

Cedula, FechaNacimiento = Nombre, es DF? Es una relación 1 a 1?, aunque la fecha de nacimiento solita no establece una funcion inyectiva (1 a 1), la unión con cedula si la garantiza.

En general si K es una superllave, implica que para todo B, se cumple que K=>B, donde B pertenece a R, tal que B es atributo y R un esquema

Empleado (id identity (1, 1) PK, cedula, nombre, Genero, fechaNacimiento)

Id => cedula (establece relacion 1 a 1), id => nombre, id => Genero, id => fechaNacimiento

Cedula => nombre, cedula => genero, cedula => fechanacimiento}

Si una atributo determina todos los demás atributos, entonces se dice que es una superllave.

Una superllave es una candidata a ser PK.

Nombre + Genero + FechaNacimiento es una llave candidata? No lo es si hay 2 o mas empleados que se llaman igual, tienen el mismo genero y nacieron la misma fecha.

Existe un algebra de las dependencias funcionales, basada en las propiedades del operador =>

A => A, es reflexivo

A => B, B=>A ¿? Es conmutativo? No es commutativo?

Si A=>B, entonces A,C = B, <no se como se llama esta propiedad del operador>, si lo cumple.

Sera transitiva?

A => B y B => C, entonces A => C ¿?...

Estudiante (Id PK, valordocidentidad, nombre, genero, idCarrera, nombreCarrera)

Id ➔ idCarrera, idCarrera => NombreCarrera implica que … Id => NombreCarrera

SI es transitiva