T7 - Introducción a las BBDD y al Modlo Relacional

Universidad de Sevilla

Departamento de Lenguajes y Sistemas Informáticos

Objetivos


Relacional
Anomalías
Integridad
Modelo
Está
Datos
Introducción
Funcionales
Relación
Atributos
Formas
Normales
Claves
Valor
Dependencias
Atributo
Clave
Manipulación
Tuplas
Valores

Contenido

Contenido

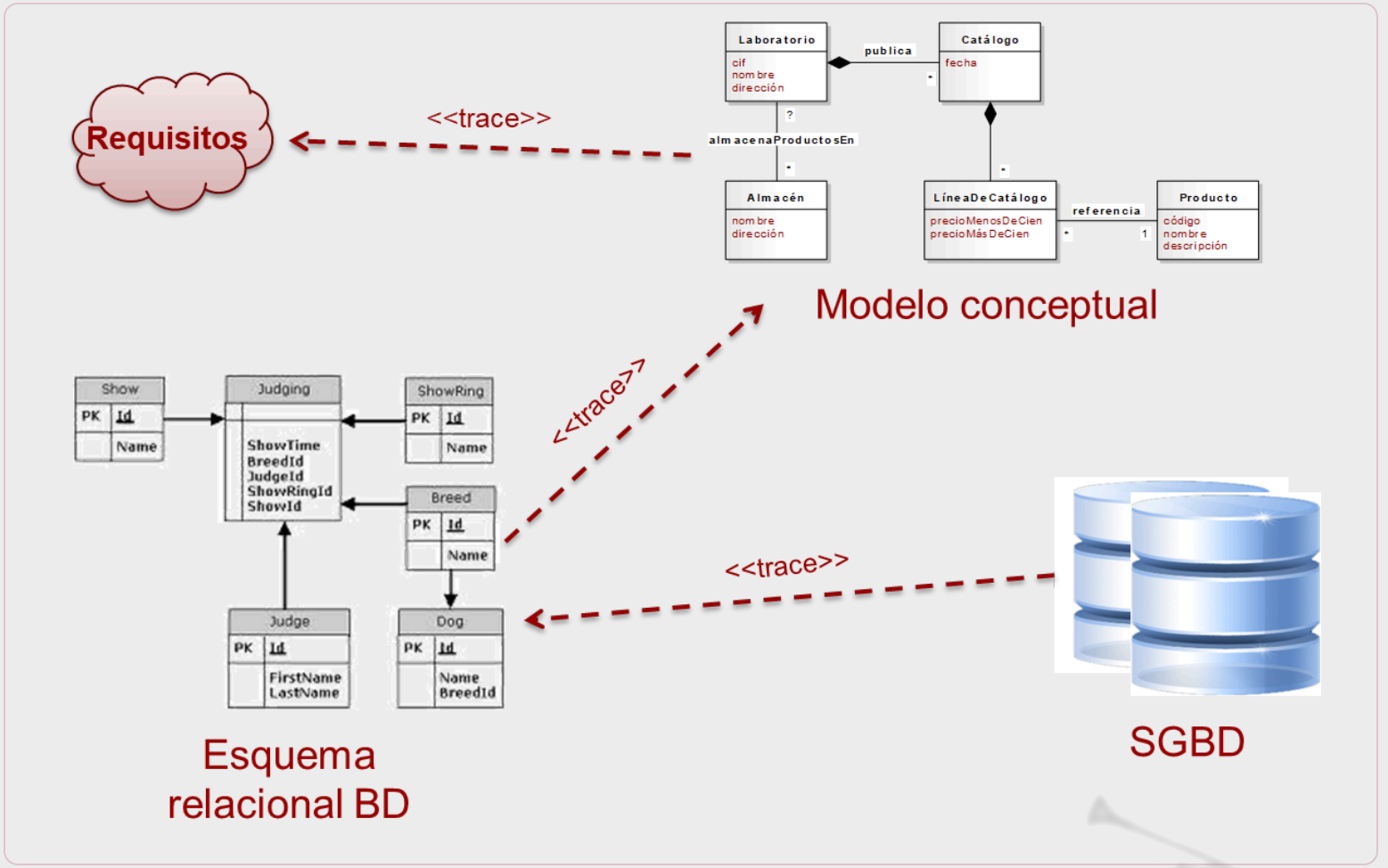
¿Qué es una base de datos?

Un sistema de gestión de base de datos (SGBD) es un sistema informático que:

- Implementa (principalmente) las funciones de memoria e informativa de un sistema de información.
- Almacena grandes volúmenes de datos.
- Gestiona el acceso concurrente a los datos.
- Mantiene la integridad semántica de los datos.
- Controla el acceso a los datos.

Trazabilidad

El producto previo al diseño de una BD es el modelo conceptual



Introducción. Pasado

Sistemas pre-relacionales (antes de 1970)

- Basados en archivos: secuenciales e indexados
- Bases de datos jerárquicas: IMS
- Bases de datos en red: CODASYL

Sistemas relacionales (desde 1970)

- Experimentales: RDMS, Ingres, ...
- Comerciales: Oracle, DB2, MS SQL Server, ...
- Open source: PostgreSQL, MySQL, MariaDB, SQLite, ...

Introducción. Presente

Hoy conviven varios tipos de bases de datos. La elección depende del problema.

- **Relacionales (SQL)**: Datos transaccionales con reglas de integridad fuertes. PostgreSQL, MySQL, SQL Server, Oracle, ...
- **Documentales**: Datos semiestructurados (JSON), esquemas flexibles.MongoDB, Couchbase, ...
- **Clave-valor**: Caché, sesiones, contadores, colas simples. Redis, DynamoDB (modo KV), ...
- **Grafos**: Relaciones complejas (redes sociales, fraude, recomendación). Neo4j, JanusGraph, ...
- Otras familias **NoSQL** frecuentes
  - *Columna ancha*: Cassandra, HBase.
  - *Series temporales*: InfluxDB, TimescaleDB.
  - *Búsqueda/analítica*: Elasticsearch, OpenSearch.

Introducción. Futuro

Tendencia: consumir capacidades de backend como servicio gestionado.

- Backend as a Service (**BaaS**)
  - Proporciona componentes listos: autenticación, base de datos, almacenamiento, API, notificaciones, funciones serverless.
  - Reduce tiempo de desarrollo y operaciones en proyectos pequeños y medianos.
  - Ejemplos de plataformas: Firebase, Supabase, Appwrite, AWS Amplify.
- Ventajas
  - Salida rápida al mercado.
  - Escalado y seguridad base gestionados por el proveedor.
  - Menor coste inicial de infraestructura.
- Riesgos y límites
  - Dependencia del proveedor (vendor lock-in).
  - Menor control fino de arquitectura y costes a gran escala.
  - Necesidad de diseñar portabilidad desde el inicio.

Contenido

Orígenes

Codd, E. F. A Relational Model of Data for Large Shared Data Banks. Communications of the ACM 13 (6): 377–387. Link

- Modelo sencillo y flexible basado en fundamentos matemáticos de teoría de conjuntos y lógica de predicados.
- Primeras implementaciones:
  - RDMS (MIT), primeros años de los 70
  - Ingres (Universidad de Berkeley), 1974
  - Oracle V2, 1979
  - IBM System R/DB2, 1979

¿Qué es una relación?

Una relación está compuesta por una intensión y por una extensión.

La intensión define un conjunto de atributos, cada uno de los cuales toma valores sobre un dominio.

Por ejemplo, para una relación denominada `Empleados`, su intensión podría ser:

- **nif**: Naturales de 8 dígitos seguidos de una letra mayúscula
- **nss**: Naturales de 12 dígitos
- **nombre**: Cadenas menores de 50 caracteres
- **edad**: Naturales menores o iguales a 200
- **salario**: Reales positivos
- **estadoCivil**: enumerado (soltero, casado, viudo, ...)

¿Qué es una relación?

La extensión es un conjunto de tuplas, cada una formada por conjuntos de pares (atributo, valor), de forma que a cada atributo de la intensión se le asocia un valor del dominio sobre el que está definido.

Por ejemplo, para la relación `Empleados`, su extensión podría ser:

```
-- Intensión
Empleados = { nif, nss, nombre, edad, salario, estadoCivil }

-- Extensión
Empleados = {
    ('12.345.678-Z', '123.456.789', 'Abel Abad', 21, 12000, 'Soltero'),
    ('23.456.789-D', '234.567.890', 'Braulio Brío', 32, 23000, 'Casado'),
    ('34.567.890-V', '345.678.901', 'Carlos Cepa', 43, 34000, 'Separado'),
    ('45.678.901-G', '456.789.012', 'David Díaz', 54, 45000, 'Divorciado'),
    ('56.789.012-B', '567.890.123', 'Enrique Estepa', 65, 56000, 'Casado')
}
```

Grado: número de atributos (columnas)
Cardinalidad: número de tuplas de la extesión (filas)

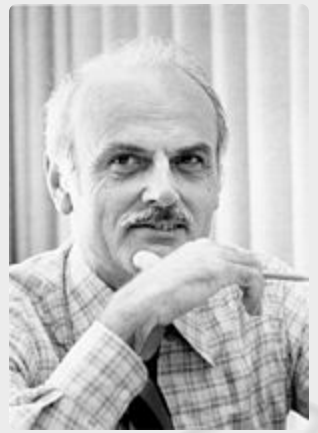
Implementación del concepto de relación

El concepto de relación se implementa mediante una tabla, cuyas columnas representan los atributos y las filas las tuplas.

Las diferencias con una relación son:

- Las filas están ordenadas, las tuplas no.
- Las filas pueden estar repetidas, las tuplas no.
- Las columnas (además de un nombre) tienen un orden, los atributos no.

Larry Ellison



Edgar F. Codd

¿Qué es un dominio?

Es el conjunto de valores admisibles que puede tomar un atributo de una relación.

Pueden ser:

- Un tipo básico sin restricciones: enteros, cadenas, …
- Un tipo básico con restricciones: reales < 100, cadenas de longitud ≤ 50, naturales > 20, …
- Un tipo enumerado: lunes, martes, miércoles, …
- Un tipo compuesto: fechas (dd/mm/aaaa), horas (hh:mm:ss), …

Todos los dominios deben tener definido el operador de igualdad ($=$).

Algunos dominios tienen su propia álgebra con operadores $+$, $-$, $*$, $/$, ...

¿Qué es el valor nulo?

Si un atributo tiene valor nulo (null), significa que no se conoce su valor, que es desconocido.

El valor nulo puede asignarse a atributos definidos sobre cualquier dominio, pero no pueden compararse valores nulos de atributos definidos sobre dominios diferentes.

La introducción del valor nulo implica la necesidad de una lógica trivaluada:

A	B	A OR B	A AND B	NOT A
true	null	true	null	false
false	null	null	false	true
null	null	null	null	null

Contenido

Superclaves - Unicidad

Una Superclave es un subconjunto de atributos (descriptor) de una relación que identifica de manera única a las tuplas de dicha relación.

Criterio de **unicidad**:

- Un descriptor es único si no tiene sentido* que existan dos tuplas distintas en una relación con los mismos valores de dicho descriptor.
- Toda Superclave cumple este criterio

> * Si no tiene sentido en el dominio del problema, asumiendo que una relación representa una parte de un modelo conceptual, como se verá en los próximos temas.

Clave Candidata

En cualquier relación, siempre existe una superclave formada por todos los atributos, ya que no puede haber tuplas repetidas.

Una relación puede tener varias superclaves.

Criterio de **Minimalidad**:

- Una Superclave es Mínima si al eliminar cualquier atributo de su descriptor deja de cumplir el criterio de unicidad.
- Las Superclaves que cumplen este criterio se denominan Claves Candidatas (cumplen unicidad y minimalidad)

Hay que evitar las Superclaves que no son Claves Candidatas.

Candidatas, Primarias y Alternativas

Claves candidatas

- Una relación siempre tiene al menos una clave candidata, aunque puede tener varias.

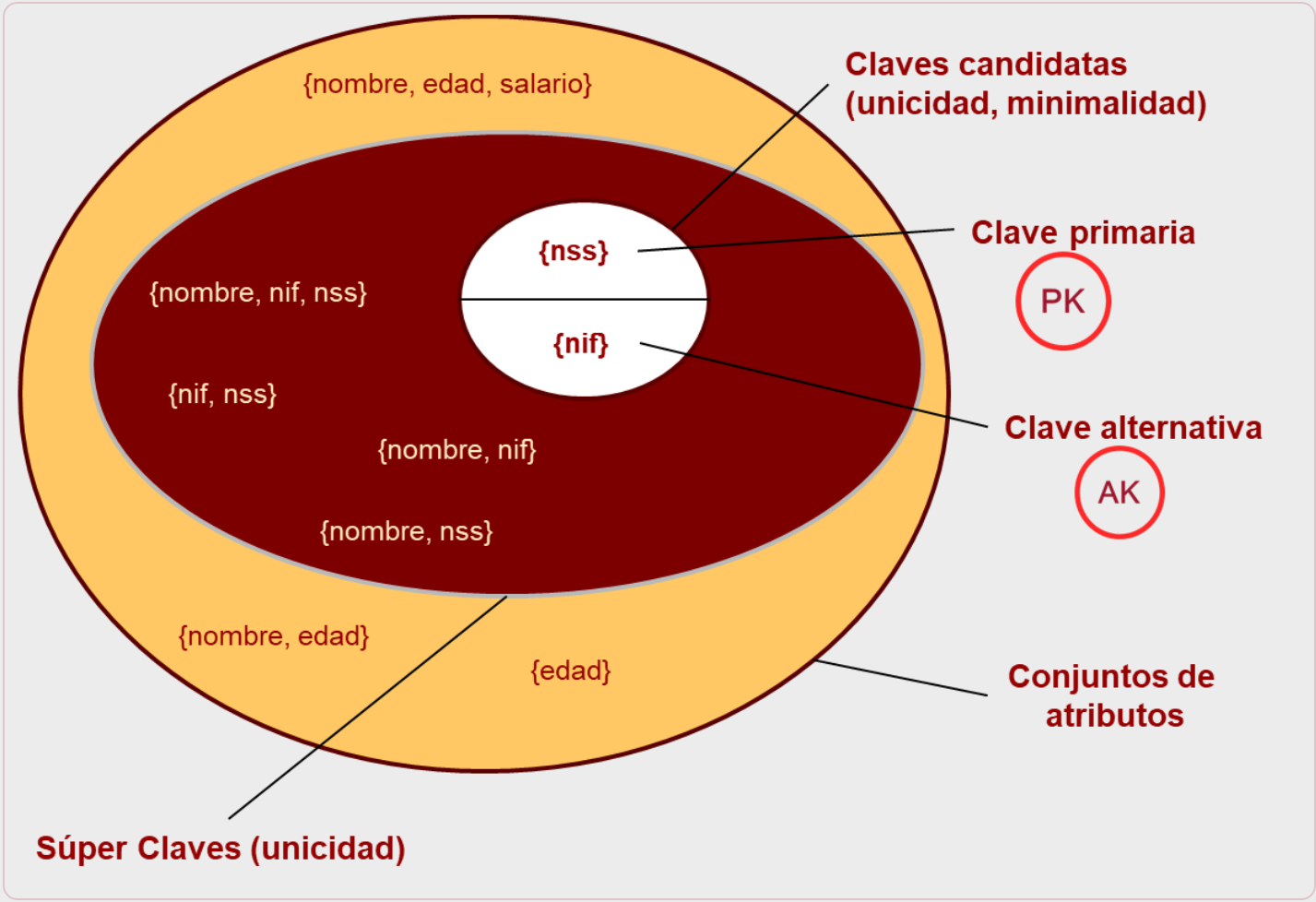
Clave primaria (**PK**)

- Clave candidata seleccionada arbitrariamente como mecanismo de identificación de la relación.

Clave alternativa (**AK**)

- Cualquier clave candidata no seleccionada como primaria.

Relaciones entre claves



Claves ajenas (FK)

Las claves ajenas son conjuntos de atributos de una relación cuyos valores deben coincidir con los de la clave primaria de otra relación.

Es la forma de representar las asociaciones del modelo conceptual en el modelo relacional.

Se debe evitar ponerles nombres distintos a la clave primaria

```
CentrosSalud = {centroSaludId, dirección, teléfono}
    PK(centroSaludId)
CentrosSalud ={
    (1, 'Carretera de Carmona 48,     Sevilla', '954 954 954'),
    (2, 'Avenida de la Innovación 12, Sevilla', '955 123 456')
}

Personas = {personaId, nif, nss, nombre, centroSaludId}
    PK(personaId)
    FK(centroSaludId) / CentrosSalud
    AK(nif)
    AK(nss)
Personas ={
    (1, '28456319H', '281234567890', 'Ana Romero',  1),
    (2, '51789234M', '281234567891', 'Luis Ortega', 1)
}
```

Contenido

Integridad de la entidad

Ningún atributo que forme parte de la clave primaria de una relación puede tomar el valor nulo.

Esta regla de integridad garantiza la identificación de tuplas mediante valores de la clave primaria.

```
Empleados = {nif, nss, nombre, edad, salario, estadoCivil}
    PK(nif)
    AK(nss)

Empleados ={
    ('12.345.678-Z', '123.456.789', 'Abel Abad',      21, 12000, 'soltero'),
    (null,           '234.567.890', 'Braulio Brío',   32, 23000, 'casado'), -- viola regla
    ('34.567.890-V', '345.678.901', 'Carlos Cepa',    43, 34000, 'separado'),
    ('45.678.901-G', '456.789.012', 'David Díaz',     54, 45000, 'divorciado'),
    ('56.789.012-B', '567.890.123', 'Enrique Estepa', 65, 56000, 'casado')
}
```

> El valor de una PK nunca puede ser nulo

Integridad referencial

Todos los atributos de una clave ajena deben tomar valores que coincidan con valores de la clave primaria correspondiente o bien tomar valores nulos.

Esta regla de integridad garantiza que todas las tuplas con claves ajenas se relacionan con otras tuplas existentes o bien con ninguna.

```
Empleados = {nss, nif, nombre, edad, centroSaludId}
    PK(nif)
    AK(nss)
    FK(centroSaludId) / CentrosSalud
Empleados ={
    ('123.456.789', '12.345.678-Z', 'Abel',    21, 48),
    ('234.567.890', '23.456.789-D', 'Braulio', 32, 1),
    ('345.678.901', '34.567.890-V', 'Carlos',  43, 2),
    ('456.789.012', '45.678.901-G', 'David',   40, 4),
    ('567.890.123', '56.789.012-B', 'Enrique', 65, null)
}
CentrosSalud = {centroSaludId, dirección, teléfono}
    PK(centroSaludId)
CentrosSalud ={
    (1, 'c/ Primera 10, Sevilla', '954 111 111'),
    (2, 'c/ Segunda 22, Sevilla', '954 222 222'),
    (3, 'c/ Tercera 8, Sevilla',  '954 333 333'),
    (4, 'c/ Cuarta 15, Sevilla',  '954 444 444')
}
```

El centro de salud con ID=48 no existe

Enrique no tiene asignado centro de salud

Contenido

Calidad de los modelos relacionales

La calidad de un modelo relacional depende, entre otros factores, de las anomalías de manipulación que presente.

La forma de asegurar la calidad de un modelo relacional frente a las anomalías de manipulación es comprobar que está al menos en tercera forma normal (3FN).

Anomalías de manipulación

Supongamos una relación que contiene los datos de los inmuebles de una agencia de alquiler.

Cada inmueble tiene un código, una dirección, un precio de alquiler, una lista de propietarios con el porcentaje de propiedad del inmueble, y el código, nombre y cargo del empleado que lo gestiona.

```
Inmuebles = {inmuebleId, dirección, precio, propietarios, empleadoId, nombre, cargo}

Inmuebles ={
    ('10A', 'C/ Norte, 15', 600, 'P. Río, 70% / D. Páez, 30%', 3, 'S. Blas', 'Resp. Zona'),
    ('30A', 'C/ Sur, 15',   500, 'E. Ruz, 100%',               3, 'S. Díaz', 'Resp. Zona'),
    ('87B', 'C/ Este, 8',   700, 'R. Bas, 50% / P. Río, 50%',  5, 'N. Clos', 'Resp. Zona'),
    ('91A', 'C/ Oeste, 10', 650, 'M. Gil, 40% / M. Quer, 60%', 8, 'G. Peña', 'Comercial'),
    ('23B', 'C/ Sol, 14',   800, 'R. Mel, 70% / J. Val, 30%',  8, 'G. Peña', 'Comercial')
}
```

¿Qué problemas presenta la relación?

Datos redundantes: el nombre y el cargo de cada empleado se repita tantas veces como inmuebles gestione, malgastando espacio.

Riesgos de incoherencia: la redundancia de datos implica el riesgo de que se vuelvan incoherentes si no se actualizan todas las ocurrencias a la vez.

Anomalías de inserción: hasta que un empleado no gestione un inmueble no se puede registrar en el sistema de información.

Anomalías de actualización: si un empleado cambia de cargo hay que actualizarlo múltiples veces en lugar de hacerlo una sola vez.

Anomalías de eliminación: si un empleado deja de gestionar inmuebles, sus datos desaparecen del sistema de información.

Problemas de consulta: ¿cómo se podrían conocer todos los inmuebles de un determinado propietario?

¿Qué problemas presenta la relación?

¿Cuántos problemas de los anteriores se evitan con el nuevo modelo relacional de tres relaciones?

```
-- Intensión
Empleados = { empleadoId, nombre, cargo }
    PK(empleadoId)
Inmuebles = { inmuebleId, propietarioId, empleadoId, dirección, precio}
    PK(inmuebleId)
    FK(empleadoId) / Empleados
Propietarios = {propietarioId, inmuebleId, nombre, porcentaje}
    PK(propietarioId)
    FK(inmuebleId) / Inmuebles
-- Extensión
Inmuebles = {
    ('10A', 1, 3, 'C/ Norte, 15', 600),
    ('30A', 3, 3, 'C/ Sur, 15',  500), ... }
Empleados = {
    (3, 'S. Blas', 'Resp. Zona'), ... }
Propietarios = {
    (1, '10A', 'P. Río',  70),
    (2, '10A', 'D. Páez', 30),
    (3, '30A', 'E. Ruz',  100), ... }
```

Contenido

Notación

$I(R)$: intensión de R (conjunto de atributos).

$E(R)$: extensión de R (conjunto de tuplas).

$t_1, t_2 \in \text{ext}(R)$: dos tuplas de R.

$t_1[X]$: proyección de la tupla t_1 sobre el conjunto de atributos X (valores de X en t_1).

$t_1[Y]$: proyección de la tupla t_1 sobre Y.

¿Qué es una dependencia funcional?

Si R es una relación y X e Y son dos subconjuntos de los atributos de R, se dice que X **determina funcionalmente a** Y o que Y **depende funcionalmente de** X, si y sólo si **siempre** que dos tuplas tienen los mismos valores de X, también tienen los mismos valores de Y.

$$\forall t_1, t_2 \in \text{ext}(R) \cdot ((t_1[X] = t_2[X]) \Rightarrow (t_1[Y] = t_2[Y]))$$

Se denota como: $X \rightarrow Y$

En otras palabras: **nunca** dos tuplas con los mismos valores de X pueden tener distintos valores de Y.

$$\nexists t_1, t_2 \in \text{ext}(R) \cdot ((t_1[X] = t_2[X]) \wedge (t_1[Y] \neq t_2[Y]))$$

¿Cómo se identifican las dependencias funcionales?

Las dependencias funcionales **no pueden deducirse de los datos** de la extensión de una relación.

Sólo podría descartarse su existencia si los datos de la extensión las contradijeran.

Por lo tanto...

> Las dependencias funcionales dependen de la semántica de los atributos de las relaciones en el modelo conceptual y, por extensión, en el dominio del problema.

En el ejemplo anterior…

$$inmuebleId \rightarrow dirección$$
$$inmuebleId \rightarrow \{dirección,\ precio,\ propietarios\}$$
$$inmuebleId \rightarrow \{empleadoId,\ nombre,\ cargo\}$$
$$\{inmuebleId,\ precio\} \rightarrow empleadoId$$
$$empleadoId \rightarrow \{nombre,\ cargo\}$$
$$\{empleadoId,\ nombre\} \rightarrow cargo$$

```
Inmuebles = {inmuebleId, dirección, precio, propietarios, empleadoId, nombre, cargo}
    PK(inmuebleId)

Inmuebles ={
    ('10A', 'C/ Norte, 15', 600, 'P. Río, 70% / D. Páez, 30%', 3, 'S. Blas', 'Resp. Zona'),
    ('30A', 'C/ Sur, 15',   500, 'E. Ruz, 100%',               3, 'S. Díaz', 'Resp. Zona'),
    ('87B', 'C/ Este, 8',   700, 'R. Bas, 50% / P. Río, 50%',  5, 'N. Clos', 'Resp. Zona'),
    ('91A', 'C/ Oeste, 10', 650, 'M. Gil, 40% / M. Quer, 60%', 8, 'G. Peña', 'Comercial'),
    ('23B', 'C/ Sol, 14',   800, 'R. Mel, 70% / J. Val, 30%',  8, 'G. Peña', 'Comercial')
}
```

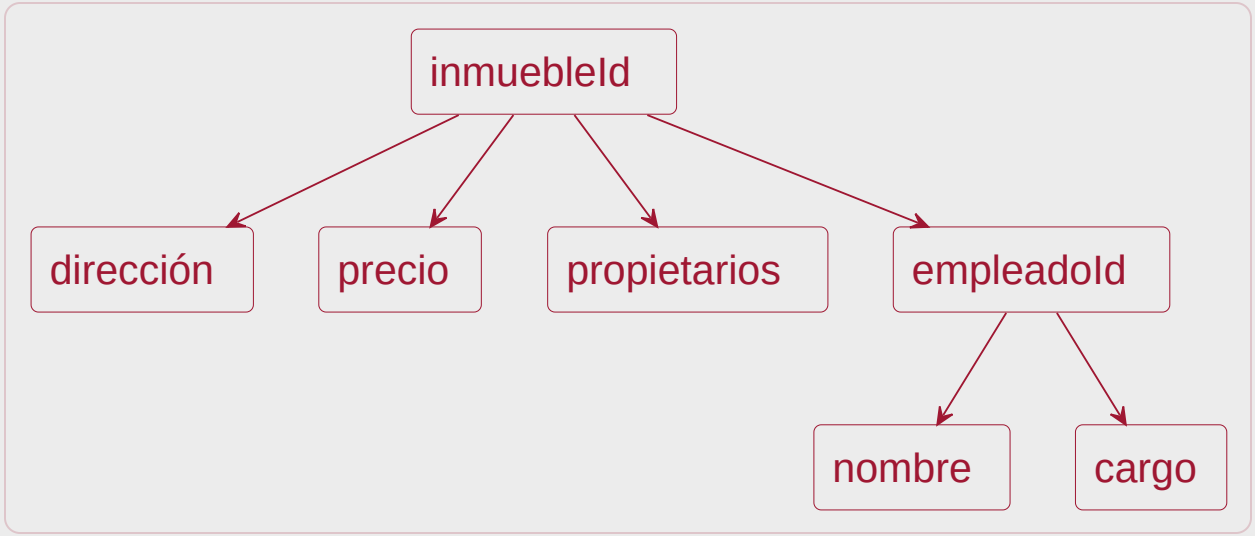
Grafo de dependencias funcionales

Forma gráfica de representar las dependencias funcionales de un modelo relacional.

Los **nodos** son **atributos** o conjuntos de atributos.

Los **arcos** son las **dependencias funcionales**.

Normalmente sólo se representan dependencias funcionales que determinan a un solo atributo.



Contenido

Formas normales

Son **condiciones**, basadas en las dependencias funcionales, que debe cumplir un modelo relacional **para estar exento de anomalías de manipulación**.

Originalmente, Codd propuso tres formas normales: **1FN, 2FN y 3FN**.

Posteriormente, se han propuesto otras tres: Boyce-Codd FN, 4FN y 5FN.

Cada FN incluye a la anterior, por lo que un modelo relacional en 3FN está también en 2FN y en 1FN.

Primera forma normal (1FN)

Una relación está en 1FN si en cada tupla se le asigna a cada atributo **un solo valor del dominio** sobre el que está definido.

Esto implica la ausencia de **grupos repetidos**.

Ejemplo:

```
-- 1FN OK (una tupla por teléfono)
Clientes = {clienteId, nombre, teléfono}
    PK(clienteId, teléfono)

Clientes ={
    (1, 'Abel Abad',    '666111222'),
    (2, 'Braulio Brío', '666222333'),
    ...
}
```

Deja de estar en 1FN por pasar de uno a varios teléfonos por cliente

```
-- 1FN FAIL (grupo repetido en teléfono)
Clientes = {clienteId, nombre, teléfono}
    PK(clienteId)

Clientes ={
    (1, 'Abel Abad',    '666111222'),
    (2, 'Braulio Brío', '666222333 / 666555666 '),
    ...
}
```

Ejemplo 1FN - Clientes

Ejemplo: Pasar de un teléfono por cliente a varios, sin conocer cuántos teléfonos puede tener un cliente

```
Clientes = {clienteId, nombre, teléfono}
    PK(clienteId)

Clientes ={
    (1, 'Abel Abad',    '666111222'),
    (2, 'Braulio Brío', '666222333'),
    (3, 'Carlos Cepa',  '666333444'),
    ...
}
```

Opción 1: cumple la 1FN, aunque no es la mejor:

```
Clientes = {clienteId, nombre, teléfono1, teléfono2, teléfono3}
    PK(clienteId)

Clientes ={
    (1, 'Abel Abad',    '666111222', null,        null),
    (2, 'Braulio Brío', '666222333', '666555666', '954456789'),
    (3, 'Carlos Cepa',  '666333444', '954123123', null),
    ...
}
```

Ejemplo 1FN - Clientes

Ejemplo: Pasar de un teléfono por cliente a varios, sin conocer cuántos teléfonos puede tener un cliente

```
-- Intensión
Clientes = {clienteId, nombre}
    PK(clienteId)
Clientes ={
    (1, 'Abel Abad'),
    (2, 'Braulio Brío'),
    (3, 'Carlos Cepa'), ... }
-- Extensión
Teléfonos = {clienteId, teléfono}
    PK(clienteId, teléfono)
    FK(clienteId) / Clientes
Teléfonos ={
    (1, '666111222'),
    (2, '666222333'),
    (2, '666555666'),
    (2, '954456789'), ... }
```

Ejemplo 1FN - Inmuebles

```
Inmuebles = {inmuebleId, dirección, precio, propietarios, empleadoId, nombre, cargo}
    PK(inmuebleId)
Inmuebles ={
    ('10A', 'C/ Norte, 15', 600, 'P. Río, 70% / D. Páez, 30%', 3, 'S. Blas', 'Resp. Zona'),
    ('30A', 'C/ Sur, 15',   500, 'E. Ruz, 100%',               3, 'S. Díaz', 'Resp. Zona'),
    ...
}
```

Partiendo el atributo propietarios en dos atributos con el nombre del propietario y su porcentaje se cumple con la 1FN.

```
Inmuebles = {inmuebleId, dirección, precio, propietario, porcentaje, empleadoId, nombre, cargo}
    PK(inmuebleId, propietario)
Inmuebles ={
    ('10A', 'C/ Norte, 15', 600, 'P. Río',  70,  3, 'S. Blas', 'Resp. Zona'),
    ('10A', 'C/ Norte, 15', 600, 'D. Páez', 30,  3, 'S. Blas', 'Resp. Zona'),
    ('30A', 'C/ Sur, 15',   500, 'E. Ruz',  100, 3, 'S. Díaz', 'Resp. Zona'),
    ...
}
```

- Nótese que ahora la PK es la pareja {inmuebleId,propietario}
- Cuando el nombre de un atributo está en plural es señal de que el modelo está mal

Segunda forma normal (2FN)

Una relación está en 2FN si está en 1FN y todos los atributos no primos son completamente dependientes de las claves candidatas de la relación.

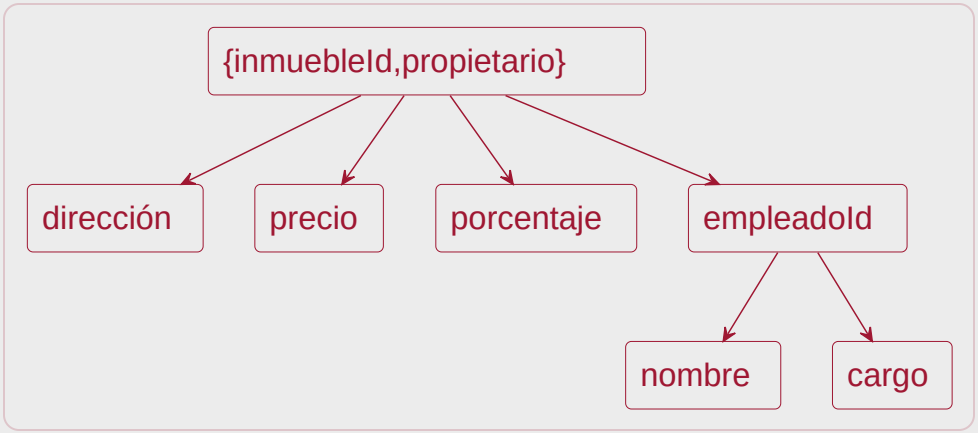
Los atributos no primos son los que no forman parte de ninguna clave candidata.

Normalmente una relación no está en 2FN porque está representando varias entidades y asociaciones a la vez.

Siempre se puede transformar un modelo relacional que no esté en 2FN en otro que sí lo esté sin pérdidas de información ni dependencias.

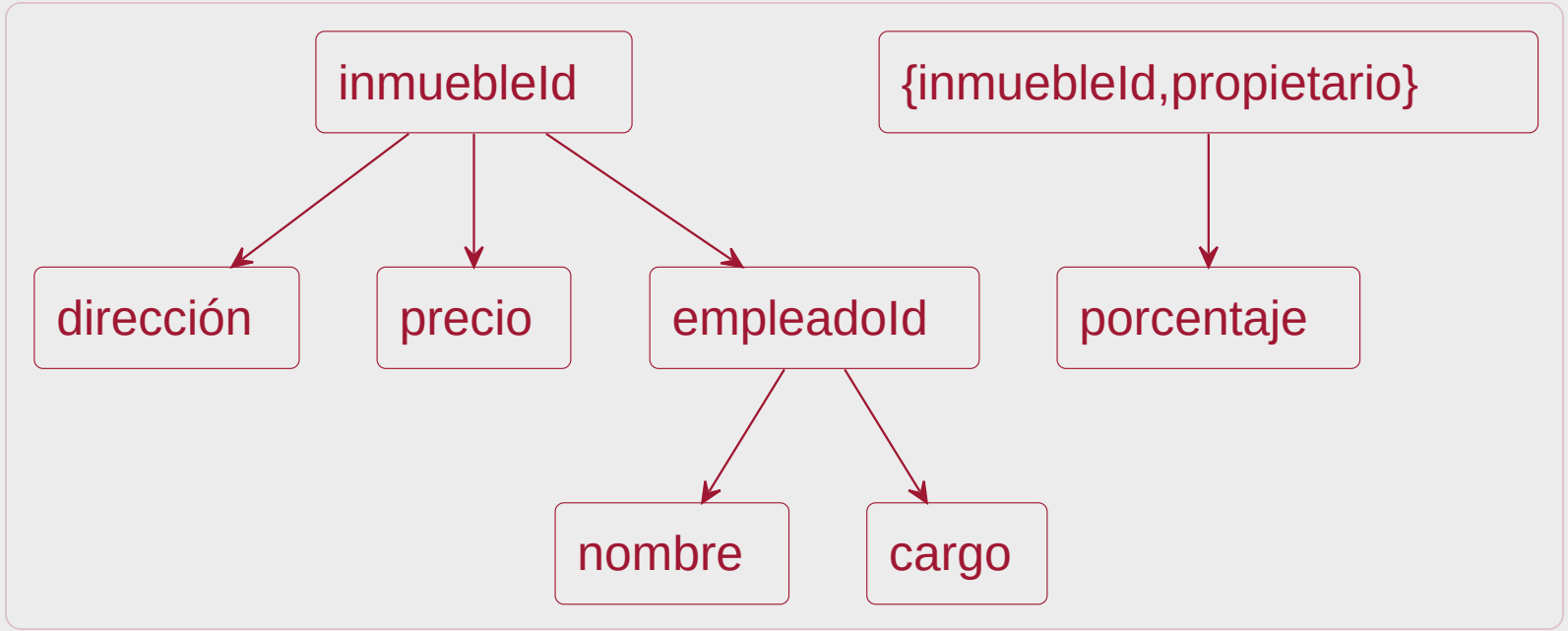
Ejemplo 2FN - Inmuebles

```
Inmuebles = { inmuebleId, dirección, precio, propietario, porcentaje, empleadoId, nombre, cargo }
    PK(inmuebleId, propietario)
Inmuebles = {
    ('10A', 'C/ Norte, 15', 600, 'P. Río', 70,  3, 'S. Blas', 'Resp. Zona'),
    ('10A', 'C/ Norte, 15', 600, 'D. Páez', 30, 3, 'S. Blas', 'Resp. Zona'),
    ('30A', 'C/ Sur, 15',   500, 'E. Ruz', 100, 3, 'S. Díaz', 'Resp. Zona'),
    ...
}
```



> Dirección/precio depende sólo de inmuebleId (dependencia parcial)
> Nombre/cargo depende sólo de empleadoId (dependencia parcial)

Ejemplo 2FN - Inmuebles




```
Inmuebles = { inmuebleId, dirección, precio, empleadoId, nombre, cargo }
    PK(inmuebleId)

Propietarios = { inmuebleId, propietario, porcentaje }
    PK(inmuebleId, propietario)
    FK(inmuebleId) / Inmuebles
```

Regla general para la 2FN

Si en la relación $R(K_1, K_2, X, Y)$ se tienen:

- los conjuntos de atributos primos: K_1 y K_2
- los conjuntos de atributos no primos: X e Y
- las dependencias funcionales: $K_1 \rightarrow X$ y $\{K_1, K_2\} \rightarrow Y$

Entonces:

- R no está en 2FN porque X no depende completamente de las claves candidatas, pero...
- La siguiente descomposición sí está en 2FN:
  - $R_1(K_1, X)$ con $K_1 \rightarrow X$
  - $R_2(K_1, K_2, Y)$ con $\{K_1, K_2\} \rightarrow Y$

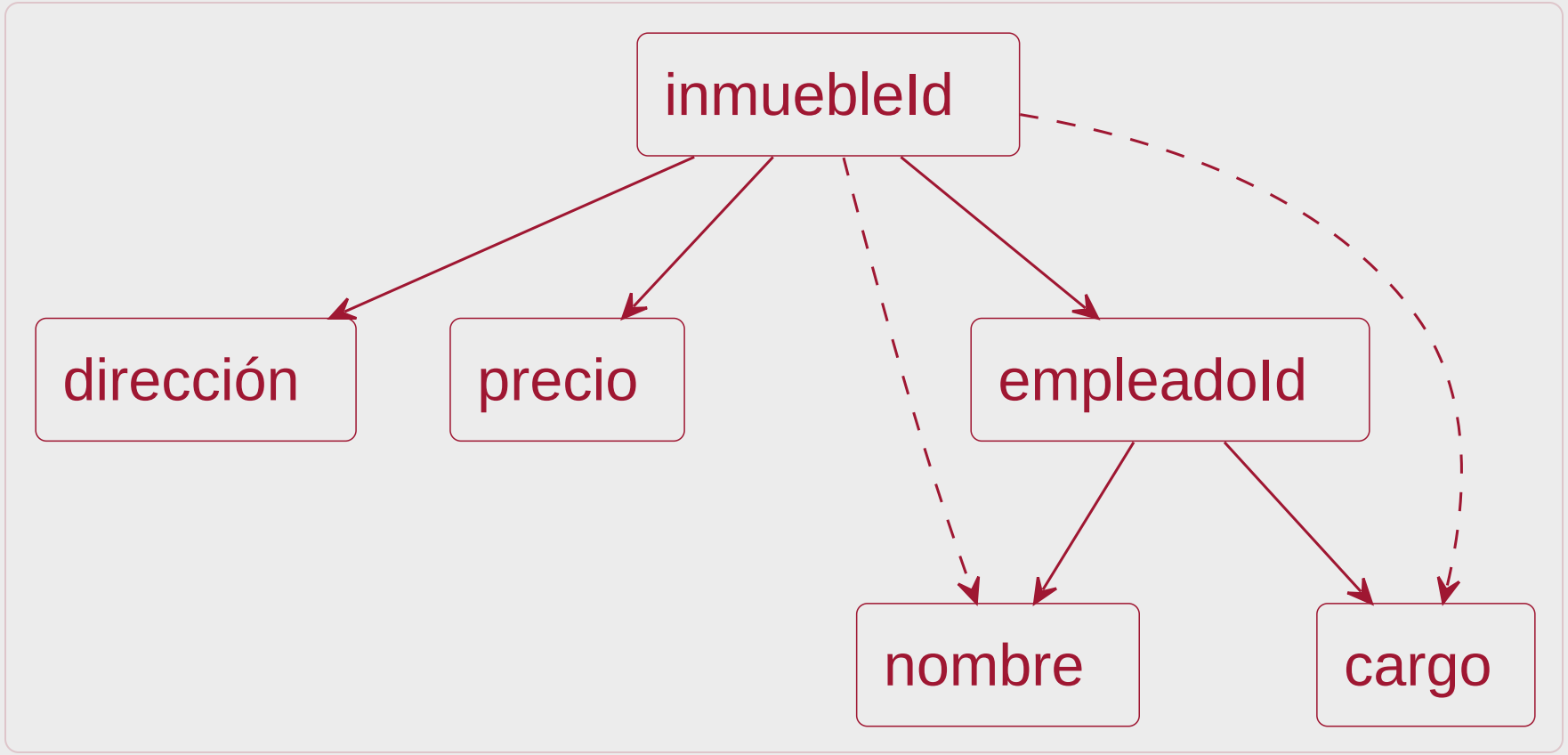
Tercera forma normal (3FN)

Una relación está en 3FN si está en 2FN y ningún atributo no primo depende **transitivamente** de ninguna clave candidata.

Justificación de la 3FN

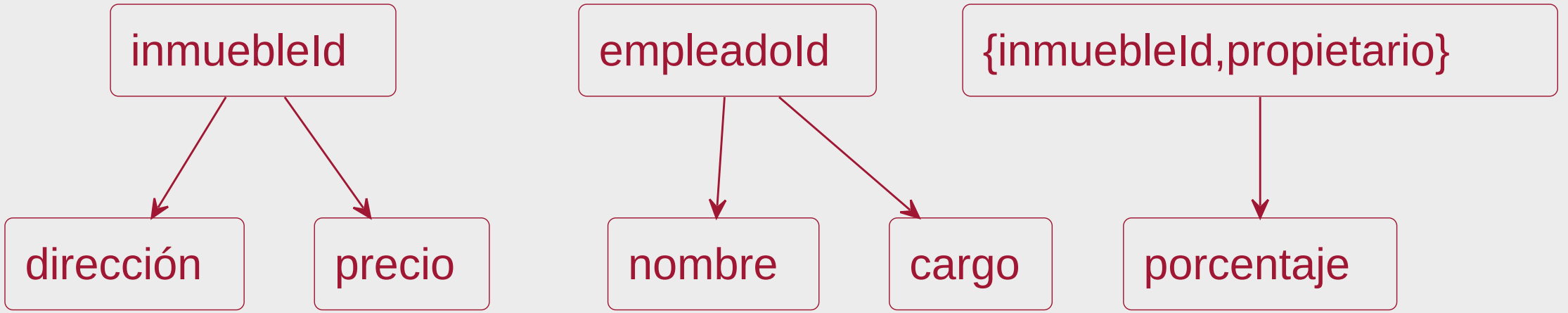
- Todos los atributos no primos deben representar un hecho sobre la clave, toda la clave y nada más que la clave.
- Normalmente una relación no está en 3FN porque está representando varias entidades asociadas a la vez.

Ejemplo 3FN - Inmuebles

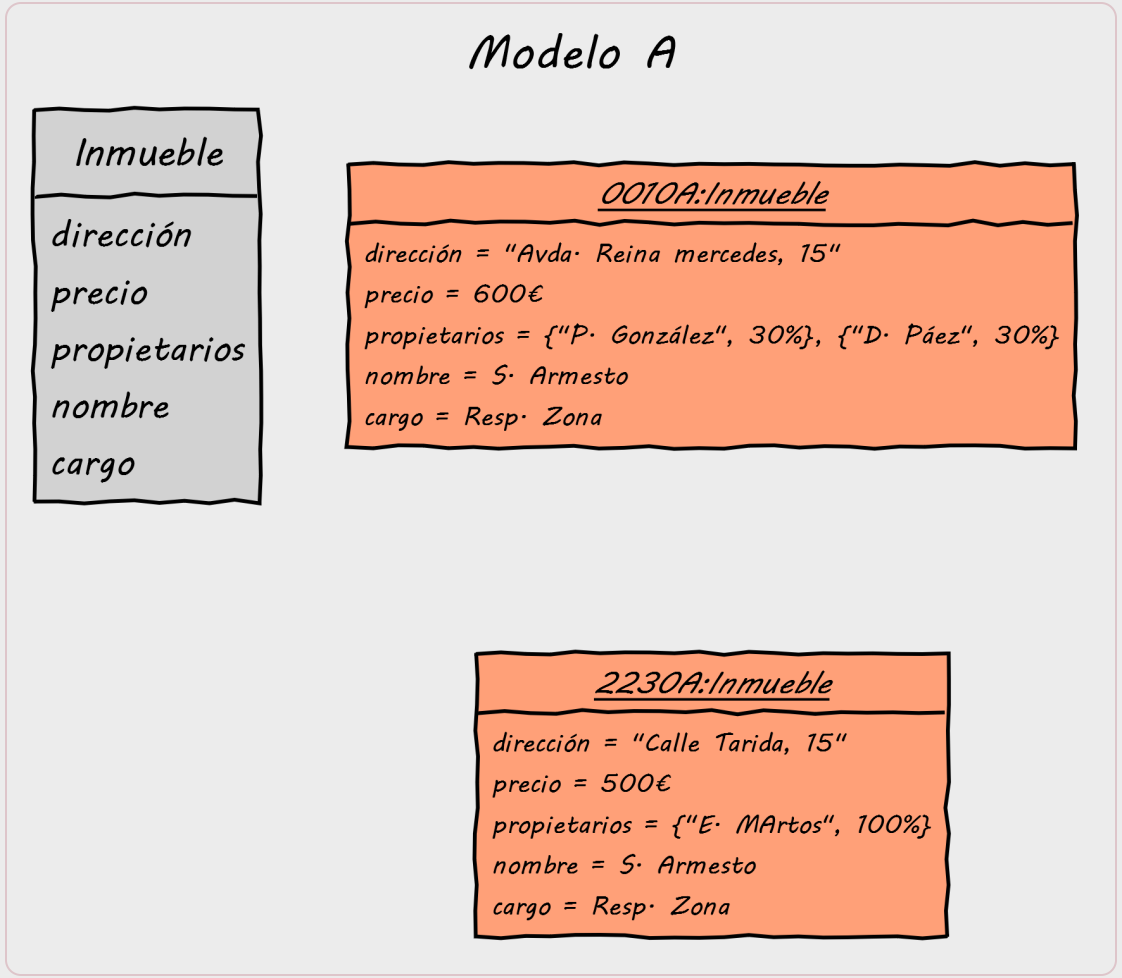


Existen dos dependencia transitivas

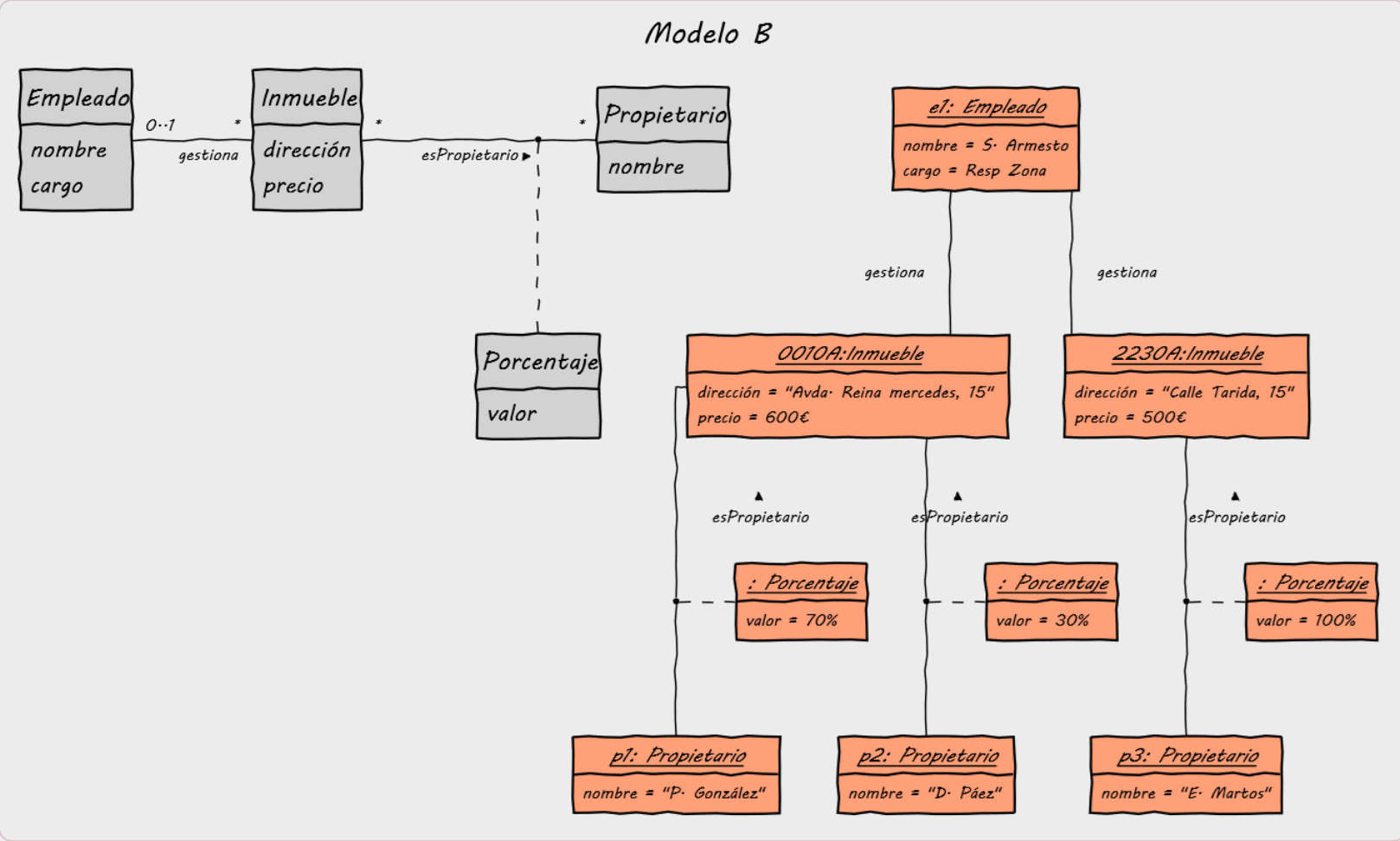
Ejemplo 3FN:


inmuebleId
empleadoId
{inmuebleId,propietario}
dirección
precio
nombre
cargo
porcentaje

El fallo estaba en el modelo



El fallo estaba en el modelo



Conclusiones

Una **base de datos** organiza datos del mundo real para soportar procesos de **consulta** y **gestión**.

El **modelo relacional** representa la información mediante **relaciones**, **tuplas** y **atributos**.

La calidad del modelo depende de su capacidad para evitar **redundancia** y **anomalías de manipulación**.

El diseño relacional debe mantener **trazabilidad** con los requisitos y con el modelo conceptual previo.

Conclusiones

Las **claves candidatas** permiten identificar unívocamente tuplas; la **clave primaria** es una de ellas.

Las **claves ajenas (FK)** conectan relaciones y hacen explícitas las asociaciones del dominio.

La **integridad de entidad** exige PK no nula y sin duplicados.

La **integridad referencial** obliga a que toda FK apunte a una PK existente (o sea nula si procede).

Conclusiones

Las **dependencias funcionales** capturan restricciones semánticas entre atributos.

La **1FN** elimina grupos repetidos y obliga a valores **atómicos**.

La **2FN** elimina dependencias **parciales** respecto a claves compuestas.

La **3FN** elimina dependencias **transitivas** y ayuda a obtener modelos más consistentes.

Gracias

Universidad de Sevilla

Departamento de Lenguajes y Sistemas Informáticos