Basic relational algebra

The relational algebra operates on relations, which are sets of tuples of the same arity, which is to say, collections of lists of the same length. Here are two 4-tuples:

(1, 2, 3, 4)
(8, 7, 9, 4)

Relations are commonly represented as tables.

There are 5 primitive operations within the relational algebra:

Projection: extract specific columns from a relation

Selection: extract specific rows

Set union: create a new table composed of all the rows of two other tables

Set difference: remove the rows in one relation that appear in another

Cartesian product: "multiply" two tables to create a third

Cartesian product in more detail

Relation1 (arity 4; length 3)

8	7	9	1
1	2	3	4
7	6	2	3

Relation 2 (arity 3; length 2)

3	**4**	**7**
1	**9**	**8**

8	7	9	1	**3**	**4**	**7**
8	7	9	1	**1**	**9**	**8**
1	2	3	4	**3**	**4**	**7**
1	2	3	4	**1**	**9**	**8**
7	6	2	3	**3**	**4**	**7**
7	6	2	3	**1**	**9**	**8**

Cartesian product (arity: 4 + 3; length: 3 * 2)

Relational databases and query languages

Database management systems based on the relational algebra were described by Edward F. Codd working for IBM in the early 1970s.

Codd's formulation included:

- indexes and keys,
- decomposition into normal forms, and
- integrity constraints.

Multiple languages and interfaces were developed to query and modify collections of relations, among them the Structured English Query Language, SEQUEL, developed by Chamberlain and Boyce.

SQL as an implementation of the relational algebra

The most successful such language,SQL, was based on SEQUEL.
SQL requires that each relation has a tablename, and each tuple position has a “fieldname”:

Players (arity 4; length 3)

Player	Innings	Hits	Teamnumber
8	7	9	1
1	2	3	8
7	6	2	3

Teams (arity 3; length 2)

t_num	games	rank
3	4	7
1	9	8

SQL as an implementation of the relational algebra

SQL commands map to the relational primitives as follows, where "*" stands for all fields in a table:

Projection select fieldname_list from tablename
ex: select tnum,rank from Teams

Selection **select** * **from** tablename **where** <logical expression>
ex: select * from Players where Teamnumber = 1

Union (select fieldname_list from tablename1)
union
(select fieldname_list from tablename2)
use ALL to keep duplicates

Set difference select * from (tablename1 except tablename2)

Cartesian product select * from tablename1, tablename2

Note that SQL does not specify how to perform a query; only what the result should be. It is a "declarative," rather than "procedural," language.

The relational join operation

An SQL “join” is a Cartesian product followed by a selection, as in:

```
select * from Players, Teams
    where Players.Teamnumber = Teams.t_num
```

which results in a Cartesian product table with only 2 (red) rows:

Player	Innings	Hits	Teamnumber	t_num	games	rank
8	7	9	1	3	4	7
8	**7**	**9**	**1**	**1**	**9**	**8**
1	2	3	4	3	4	7
1	2	3	4	1	9	8
7	**6**	**2**	**3**	**3**	**4**	**7**
7	6	2	3	1	9	8

normalización de bases de datos relacionales

La normalización de bases de datos relacionales toma un esquema relacional y le aplica un conjunto de técnicas para producir un nuevo esquema que representa la misma información pero contiene menos redundancias y evita posibles anomalías en las inserciones, actualizaciones y borrados

Breve recordatorio del modelo (formal) relacional

El modelo relacional de bases de datos se basa en un modelo formal especificado de acuerdo a la teoría de conjuntos.

Una base de datos relacional puede considerarse como un conjunto de relaciones o tablas de la Forma

`R(A1, ..., An)` , donde
`R` es el nombre de la relación, que se define por una serie de atributos `Ai`.

Sobre las tablas relacionales se pueden definir diferentes restricciones.

La integridad de entidad es una restricción que nos indica que cada entidad representada por una tupla tiene que ser diferente de las demás en su relación, es decir, debe haber algunos atributos cuyos valores identifiquen unívocamente las tuplas.

La integridad referencial indica que una clave a jena solo debe contener valores que o bien sean nulos, o bien existan en la relación referenciada por la clave ajena

Proceso de normalización

El proceso de normalización consiste en comprobar en secuencia si el esquema original está en 1FN, 2FN y 3FN, analizando las dependencias funcionales en cada paso.

EJEMPLO

Tenemos una empresa pública donde los puestos de trabajo están regulados por el Estado, de modo que las condiciones salariales están determinadas por el puesto.

Se ha creado el siguiente esquema relacional:
`EMPLEADOS(nss, nombre, puesto, salario, emails)`
Con `nss` como clave primaria.

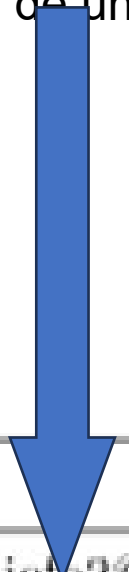
Primera forma normal (1FN)

Una tabla está en 1FN si sus atributos contienen valores atómicos.

En el ejemplo, podemos ver que el atributo `emails` puede contener más de un valor, por lo que **viola** 1FN.

En general, tenemos una relación `R` con clave primaria `K`

Si un atributo `M` viola la condición de 1FN, tenemos dos opciones



nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es; jefe2@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es; ana32@gmail.com
...	...	...	...	...

OPCION 1

Duplicar los registros con valores repetidos

En general, esta solución pasa por sustituir **R** por una nueva relación modificada **R′**, en la cual el atributo **M** que violaba 1FN se elimina.

Se incluye un nuevo atributo **M′** que solo puede contener valores simples, de modo que si **R'[M']** es uno de los valores que teníamos en **R[M]**, entonces **R'[K] = R[K].**

En otras palabras, para una tupla con n **valores duplicados** en **M** , en la nueva relación habrá n tuplas, que sólo varían en que cada una de ellas.

La clave primaria de **R′** es **(K, M')**, dado que podrá haber valores de K repetidos, para los valores multivaluados en **M**

.

Siguiendo el ejemplo, tendríamos el siguiente esquema para la nueva tabla EMPLEADOS′(a) con clave primaria (**nss, email**)

nss	nombre	puesto	salario	email
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es
111	Juan Pérez	Jefe de Área	3000	jefe2@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es
333	Ana Díaz	Administrativo	1500	ana32@gmail.com
...	...	...	...	...

OPCION 2

Separar el atributo que viola 1FN en otra tabla

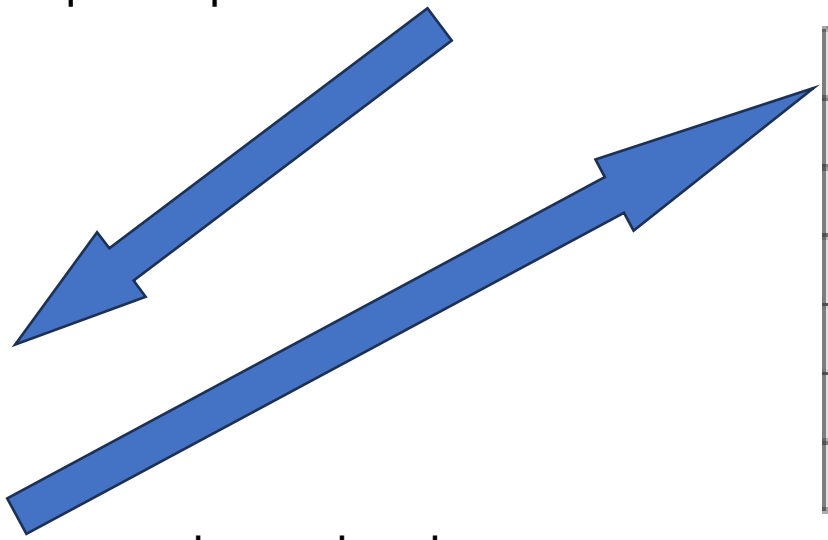
En general, esta solución pasa por sustituir **R** por una nueva relación modificada **R′** que no contiene el atributo M

.Crear una nueva relación **N(K, M′)** , es decir, una relación con una clave ajena **K** referenciando **R'**, junto al atributo **M′** , que es la variante mono-valuada del atributo M

La nueva relación **N** tiene como clave **(K, M')**

.Siguiendo el ejemplo, tendríamos el siguiente esquema para la **nueva tabla EMPLEADOS'(**b)

nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500
...	...	...	...



nss	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

Y además tendríamos **una nueva tabla EMAILS** con clave primaria **(nss,email**

Segunda forma normal (2FN)

Un esquema está en 2FN si está en 1FN y todos sus atributos que no son de la clave principal tienen dependencia funcional completa respecto de todas las claves existentes en el esquema.

En otras palabras, para determinar cada atributo no clave se necesita la **clave primaria completa**, no vale con una sub clave.

La 2FN se aplica a las relaciones que tienen claves primarias compuestas por dos o más atributos.
Si una relación está en 1FN y su clave primaria es simple (tiene un solo atributo), entonces también está en 2FN.
Por tanto, de las soluciones anteriores, la tabla `EMPLEADOS'` está en 1FN (y la tabla EMAILS no tiene atributos no clave), por lo que el esquema está en 2FN.

Sin embargo, tenemos que examinar las dependencias funcionales de los atributos no clave de `EMPLEADOS'(a).`

Las dependencias funcionales que tenemos son las siguientes: `nss->nombre, salario, email puesto->salario`

Como la clave es `(nss, email)` , las dependencias de nombre, salario y email son incompletas, por lo que la relación **no está en 2FN**

Segunda forma normal (2FN) (cont.)

En general, tendremos que observar los atributos no clave que dependan de parte de la clave.

Para solucionar este problema, tenemos que hacer lo siguiente para los grupos de atributos con dependencia incompleta `M` :

Eliminar de `R` el atributo `M`

.

Crear una nueva relación `N` con el atributo `M` y la parte de la clave primaria `K` de la que depende, que llamaremos `K'`

.

La clave primaria de la nueva relación será `K'`

.

Siguiendo el ejemplo anterior, crearíamos una nueva relación con los atributos que tienen dependencia incompleta

nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500
...	...	...	...

al eliminar de la tabla original estos atributos nos quedaría

nss	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

Como vemos, la solución a la que llegamos es la misma que en la otra opción de solución para el problema

Tercera forma normal (3FN)

Una relación está en tercera forma normal si, y sólo si está en 2FN y además, cada atributo que no está incluido en la clave primaria no depende transitivamente de la clave primaria.

Por lo tanto, a partir de un esquema en 2FN, tenemos que buscar dependencias funcionales entre atributos que no estén en la clave.

En general, tenemos que buscar dependencias transitivas de la clave, es decir, secuencias de dependencias como la siguiente: **`K-> A y A-> B`** , donde **`A`** y **`B`** no pertenecen a la clave.

La solución a este tipo de dependencias está en separar en una tabla adicional `N` el/los atributos `B` , y poner como clave primaria de **`N`** el atributo que define la transitividad **`A`**

.

Siguiendo el ejemplo anterior, podemos detectar la siguiente transitividad: **`nss->puesto puesto-> salario`**

Por lo tanto la descomposición sería la siguiente

nss	nombre	puesto
111	Juan Pérez	Jefe de Área
222	José Sánchez	Administrativo
333	Ana Díaz	Administrativo
...	...	...

En la nueva tabla `PUESTOS` , la clave sería el puesto, que también queda como clave ajena referenciandola tabla `EMPLEADOS`.

. El resto de las tablas quedan como estaban.

DISEÑO FINAL

nss	nombre	puesto
111	Juan Pérez	Jefe de Área
222	José Sánchez	Administrativo
333	Ana Díaz	Administrativo
...	...	...

nss	email
111	juanp@ecn.es
111	jefe2@ecn.es
222	jsanchez@ecn.es
333	adiaz@ecn.es
333	ana32@gmail.com
...	...

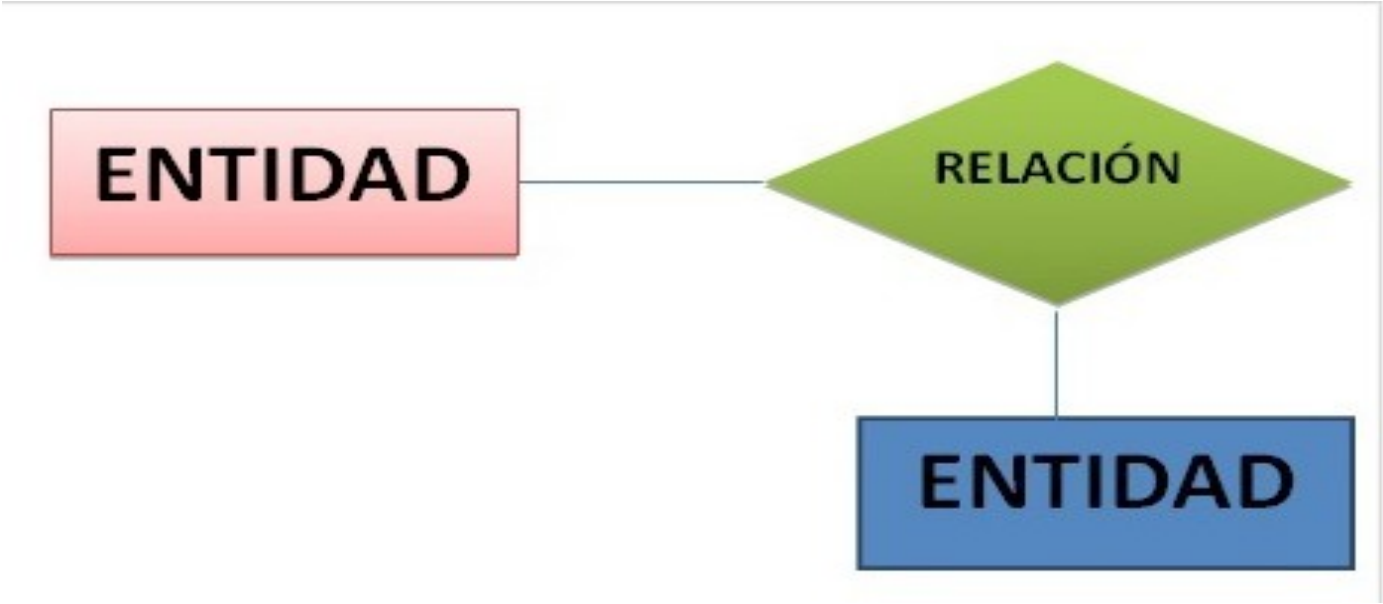
nss	nombre	puesto	salario
111	Juan Pérez	Jefe de Área	3000
222	José Sánchez	Administrativo	1500
333	Ana Díaz	Administrativo	1500
...	...	...	...

ORIGINAL

nss	nombre	puesto	salario	emails
111	Juan Pérez	Jefe de Área	3000	juanp@ecn.es; jefe2@ecn.es
222	José Sánchez	Administrativo	1500	jsanchez@ecn.es
333	Ana Díaz	Administrativo	1500	adiaz@ecn.es; ana32@gmail.com
...	...	...	...	...

MODELO ENTIDAD RELACION

El Modelo Entidad - Relación permite Modelar las Bases de Datos, desde lo conceptual, utilizando Entidades, Relaciones y Atributos, para después Modelar el esquema Lógico a través de las Tablas, trasformando los atributos en campos y las entidades en Tablas.



MODELO ENTIDAD RELACION

Una entidad caracteriza a un tipo de objeto, real o abstracto, del problema a modelizar. Toda entidad tiene existencia propia, es distinguible del resto de las entidades, tiene nombre y posee **atributos** definidos en un dominio determinado. Una entidad es todo aquello de lo que se desea almacenar información. En el diagrama E-R las entidades se representan mediante rectángulos.

Una relación es una asociación o relación matemática entre varias entidades. Las relaciones también se nombran. Se representan en el diagrama E-R mediante flechas y rombos. Cada entidad interviene en una relación con una determinada **cardinalidad**. La cardinalidad (número de instancias o elementos de una entidad que pueden asociarse a un elemento de la otra entidad relacionada) se representa mediante una pareja de datos, en minúsculas, de la forma *(cardinalidad mínima, cardinalidad máxima)*, asociada a cada uno de las entidades que intervienen en la relación. Son posibles las siguientes cardinalidades:*(0,1), (1,1), (0,n), (1,n), (m,n)*. También se informa de las cardinalidades máximas con las que intervienen las entidades en la relación.

MODELO ENTIDAD RELACION

1. El **tipo de relación** se define tomando los máximos de las cardinalidades que intervienen en la relación.

2. Hay cuatro tipos posibles:
   Una a una (1:1). En este tipo de relación, una vez fijado un elemento de una entidad se conoce la otra. Ejemplo: nación y capital.

3. Una a muchas (1:N). Ejemplo: cliente y pedidos.

4. Muchas a una (N:1). Simetría respecto al tipo anterior según el punto de visto de una u otra entidad.

5. Muchas a muchas (N:N). Ejemplo: personas y viviendas.

MODELO ENTIDAD RELACION

- **Ejemplo**

  Crear un diseño entidad relación que permita gestionar los datos de una biblioteca de modo que

  Las personas socias de la biblioteca disponen de un código de socio y además necesitar almacenar su identificación o ID, dirección, teléfono, nombre y apellidos

- La biblioteca almacena libros que presta a los socios y socias, de ellos se almacena su título, su editorial, el año en el que se escribió el libro, el nombre completo del autor (o autores), el año en que se editó y en qué editorial fue y el ISBN.

- Necesitamos poder indicar si un volumen en la biblioteca está deteriorado o no

- Queremos controlar cada préstamo que se realiza almacenando la fecha en la que se realiza, la fecha tope para devolver (que son 15 días más que la fecha en la que se realiza el préstamo) y la fecha real en la que se devuelve el libro.

MODELO ENTIDAD RELACION