# 2.4.- Integrity Constraints

**TABLE "Publications"**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | GAGA |
| B-000017 | ? | Theatre | GAGA |
| B-000008 | Doce cuentos peregrinos | Tale | GAGA |
| B-000001 | El club de los suicidas | BLUE | ROST |
| B-000001 | Poemas | Poetry | XXXX |

**AUTHOR**

| autHor_id | name |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

- Can two books have the same value for **book_id**?
- Can a book appear with no value in the attribute **title**?
- Is it possible to have the value "XXXX" in the attribute **author_id**?
- Does it make sense to have the value "BLUE" for the attribute **kind**?

# 2.4.- Integrity Constraints

**Solution**

- Definition of *domains*
- *Uniqueness constraint,*
- *Not null constraint.*
- Definition of *primary key*
- Definition of *foreign keys.*
- General integrity constraints.

- They are specified together with the database schema. The responsible for ensuring them is the DBMS.

# 2.4.1.- Constraints over attributes

**Definition of a domain**

- When we associate a domain to each attribute we restrict the set of possible values that this attribute may take.

**Example:**

"The kind of a publication can only be Novel, Tale, Theatre or Poetry".

- Domains definition:

  Kind_Dom : {Novel, Tale, Theatre, Poetry, ...}

- Relation definition:

  Publication(pub_id:*pub_dom*, title:*title_dom*, kind:*kind_dom*,

  author_id:*author_dom*);

# 2.4.1.- Constraints over attributes

NNV: $\{A_o,\ldots, A_p\}$

**Not null constraint**

- The definition of a ***not null constraint*** over a set of attributes *K* of the relation *R* expresses the following property: "there cannot be a tuple in *R* having the null value in some attribute of *K*".

**Example: NNV: { title }**

- "there cannot be a tuple in *Publication* which has the null value in the *title* attribute".
- Formally, this constraint is defined as:

$$\forall t{:}\text{Publication}\ (\neg\text{null}(t.\text{title}))$$

# 2.4.2.- Uniqueness constraint

Uni: $\{A_o,\ldots, A_p\}$

- The definition of a ***uniqueness constraint*** over a set of attributes *K* from the relation *R* expresses the following property: "there cannot be two tuples in *R* with the same values in all the attributes in set *K*".

**Example: Uni: {book_id}**

- "There cannot be two tuples in *Publication* which have the same value for the attribute *book_id"*.
- Formally, this constraint is defined as:

$$\neg(\exists t_1{:}\text{Publication}\ (\exists t_2{:}\text{Publication}\ (t_1 \neq t_2 \wedge t_1.\text{book\_id} = t_2.\text{book\_id} \wedge \neg\text{null}(t_1.\text{book\_id}) \wedge \neg\text{null}(t_2.\text{book\_id}))))$$

# 2.4.3.- Notion of primary key

PK: $\{A_o,\ldots, A_p\}$

- A **primary key** in a relation is a set of attributes which are chosen to serve as unique identifier for its tuples:
  - Must be minimal,
  - Its attributes must always have a value for each tuple (Not null constraint)
  - The values must be unique for each tuple (Uniqueness constraint).

**Example: PK: {book_id}**

*"book_id* is the primary key for *Publication"*

Formally, this can be defined as:

$$\neg(\exists t_1\text{:Publication } (\exists t_2\text{:Publication } (t_1 \neq t_2 \wedge t_1.\text{book\_id} = t_2.\text{book\_id} \wedge \neg\text{null}(t_1.\text{book\_id}) \wedge \neg\text{null}(t_2.\text{book\_id})))) \wedge \forall t\text{:Publication } (\neg\text{null}(t.\text{book\_id}))$$

# 2.4.3.- Notion of primary key

**FORMALLY, in general:**

- Given a set of attributes *PK* which has been defined as a primary key for *R*, we say that *R* satisfies the ***primary key integrity constraint*** if the following properties are met:
    - *R* satisfies a non -null constraint over *PK*, and
    - *R* satisfies a uniqueness constraint over *PK*

  otherwise *R* violates this constraint.

- Additionally, *PK* must be minimal; i.e., there cannot be any proper subset that could also be primary key for *R*.

# 2.4.4.- Notion of foreign key

$$FK: \{A_o, \ldots, A_p\} \rightarrow S$$

- The use of **foreign keys** is the mechanism provided by the relational model to express associations between the objects in a database schema. This mechanism is defined such that these associations, if performed, would be carried out adequately.

- With this goal, we can add to the schema of a relation R, a set of attributes which refer to a set of attributes of a relation *S*

- This set of attributes is called *foreign key in relation R which refers to relation S.*

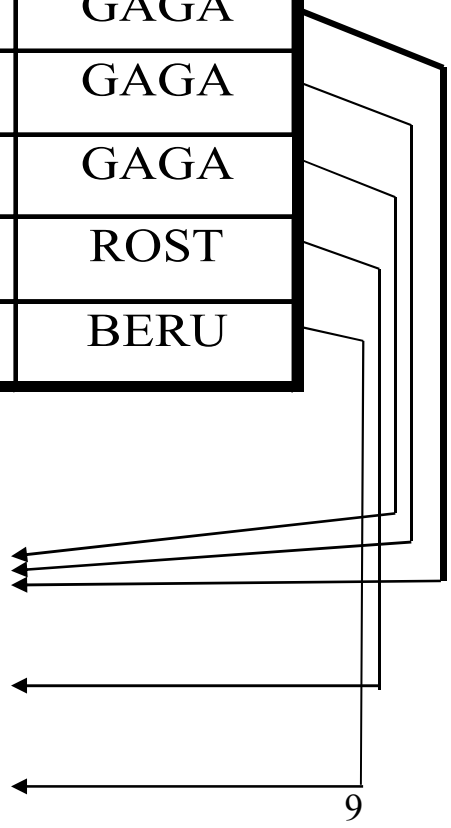
# 2.4.4.- Notion of foreign key

***PUBLICATION TABLE***

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | GAGA |
| B-000017 | Siempre NO | Theatre | GAGA |
| B-000008 | Doce cuentos peregrinos | Tale | GAGA |
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

***AUTHOR TABLE***

| author_id | name |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |



# 2.4.4.- Notion of foreign key

Is it possible to have the value "XXXX" for the attribute **author_id**?

No, the foreign key constraints states:

- If there is a tuple in "Publication" such that the value *author_id* is not null, then **there must be one and only one** tupla in "Author" such that the value of *author_id* in "Publication" matches the value *author_id* in "Author"

Formally:

$$\forall t{:}\text{Publication}\ ((\neg \text{null}(t.\text{author\_id})) \rightarrow (\exists t_1{:}\text{Author}\ (t_1.\text{author\_id} = t.\text{author\_id} \wedge \neg\exists t_2{:}\text{Author} \wedge t_1 \neq t_2 \wedge t_2.\text{author\_id} = t.\text{author\_id}))\ )$$

# 2.4.4.- Notion of foreign key

**FORMALLY:**

Given a foreign key *FK* in *R* which refers to *S*, this is defined as:

- A subset $K = \{A_i, A_j, \ldots, A_k\}$ in the schema of R
- A bijection $f$: $K \rightarrow J$ such that:
  - » $J$ is a set of attributes in S
  - » $J$ has a uniqueness constraint, and
  - » $\forall A_i\ (A_i \in K) \rightarrow A_i$ and $f(A_i)$ have the same domain.
- A type of *referential integrity*.

This *referential integrity (R.I.)* can be: *weak*, *partial* or *complete.*

# 2.4.4.- Notion of foreign key

- *R* satisfies the ***referential integrity constraint*** over the *FK* if, depending on the chosen type, the following property is met:
  - *Weak R.I.*: if in a tuple of *R* all the values for the attributes of *K* have a non null value, then there must exist a tuple in *S* taking the same values for the attributes of *J* that the values in the attributes of *K*.
  - *Partial R.I.*: if in a tuple of *R* one or more attributes of *K* have a non null value, then there must exist a tuple in *S* taking the same values for the attributes of *J* as the values in the non-null attributes of *K*.
  - *Complete (or Full) R.I.*: in a tuple of *R* all the values must have null value or none of them. In the latter case, there must exist a tuple in *S* taking the same values for the attributes in *J* as the values in the attributes of *K*.

# 2.4.4.- Notion of foreign key

**Foreign key: Simplified Notation**

- The bijection $f$: $K \rightarrow J$ can be omitted when $J$ is the primary key of $S$ and we have one of the following two cases:
  - The set $K$ has only one attribute, or
  - the bijection is defined by the syntactic equality between the attribute names in $K$ and $J$.
- The type of referential integrity (weak, partial, complete) can be omitted in any of these cases:
  - The foreign key $K$ has only one attribute, or
  - When all of them have a not null constraint,

  Since in these cases the three types of referential integrity match.

# 2.4.4.- Example

**PROVIDER** (**vcode**: vcode_d, **name**: name_d1, **city**: city_d)

PK: {vcode}

**PIECE(zcode**: zcode_d, **name**: name_d2, **colour**: colour_d, **weight**: weight_d, **city**: city_d)

PK: {zcode}

**PROJECT(ycode**: ycode_d, **name**: name_d3, **city**: city_d)

PK: {ycode}

**ORDER** (**vcode**: vcode_d, **zcode**: zcode_d, **ycode**: ycode_d, **date**: date_d, **quant**: quant_d)

PK: {vcode, zcode, ycode, date}
FK: {vcode} → PROVIDER
FK: {zcode} → PIECE
FK: {ycode} → PROJECT

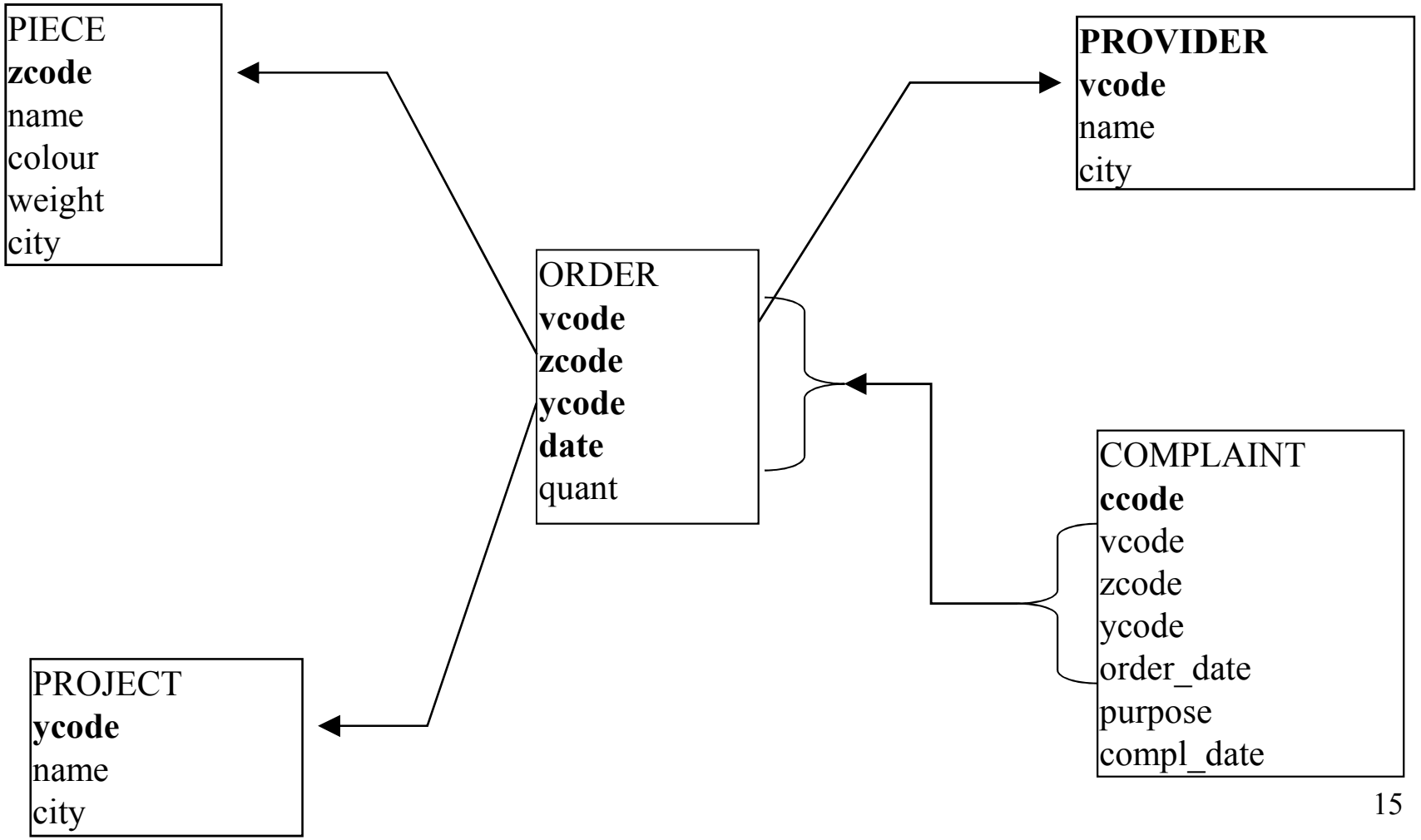
**COMPLAINT** (**ccode**: ccode_d, **vcode**: vcode_d, **zcode**: zcode_d, **ycode**: ycode_d,

**order_date**: date_d, **purpose**: purp_d, **compl_date**: date_d)

PK: {ccode}

FK: {vcode, zcode, ycode, order_date} → ORDER  f(order_date) = date

# 2.4.4.- Example


PIECE
zcode
name
colour
weight
city
PROVIDER
vcode
name
city
ORDER
vcode
zcode
ycode
date
quant
COMPLAINT
ccode
vcode
zcode
ycode
order_date
purpose
compl_date
PROJECT
ycode
name
city

# 2.4.4.- Example

**ORDER**

**referential integrity: total**

| vcode | zcode | ycode | date | quant |
|---|---|---|---|---|
| Ford | wheel | Focus | 1/1/99 | 100 |
| Ford | tyre | Ka | 2/1/99 | 300 |
| ~~Ford~~ | ~~tyre~~ | ~~Ka~~ | ~~?~~ | ~~50~~ |
| ~~Ford~~ | ~~wheel~~ | ~~Focus~~ | ~~1/1/99~~ | ~~500~~ |

**Nulls are not allowed in PK**
**PK must always be unique**

**COMPLAINT**

| ccode | vcode | zcode | ycode | order_date | purpose | compl_date |
|---|---|---|---|---|---|---|
| 1 | Ford | wheel | Focus | 1/1/99 | Square wheel | 5/2/99 |
| 2 | ~~Ford~~ | ~~tyre~~ | ~~?~~ | ~~3/1/99~~ | Punctures | 7/2/99 |
| 3 | ~~Ford~~ | ~~tyre~~ | ~~?~~ | ~~2/1/99~~ | Ovoidal | 7/3/99 |
| 4 | ? | ? | ? | ? | Didn't order this | 13/3/99 |
| 5 | ~~Ford~~ | ~~door~~ | ~~Ka~~ | ~~1/1/99~~ | Cold gets in | 14/3/99 |

# 2.4.5.- Referential integrity restoration: directives to the DBMS

- For an update operation (UPDATE and DELETE in particular) over the database which violates a referential integrity, the DBMS can:
    - Reject the operation
    - Accept the operation but also performing a compensatory action in order to preserve the referential integrity.
        - » Setting some values to *null*
        - » Propagating the action *in cascade*
- Usually, the referential integrity affects and depends on the definition of foreign keys.

# 2.4.5.- Referential integrity restoration: directives to the DBMS

**Example: Cascade propagation(1).**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | GAGA |
| B-000017 | Siempre NO | Theatre | GAGA |
| B-000008 | Doce cuentos peregrinos | Tale | GAGA |
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

| author_id | name |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

# 2.4.5.- Referential integrity restoration: directives to the DBMS

**Example: Cascade propagation(2).**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | GAGA |
| B-000017 | Siempre NO | Theatre | GAGA |
| B-000008 | Doce cuentos peregrinos | Tale | GAGA |
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

| author_id | name |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

# 2.4.5.- Referential integrity restoration: directives to the DBMS

**Example: Cascade propagation(3).**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

| author_id | name |
|---|---|
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

# 2.4.5.- Referential integrity restoration: directives to the DBMS

**Example: Setting values to null (1).**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | GAGA |
| B-000017 | Siempre NO | Theatre | GAGA |
| B-000008 | Doce cuentos peregrinos | Tale | GAGA |
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

| author_id | name |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

# 2.4.5.- Referential integrity restoration: directives to the DBMS

**Example: Setting values to null (2).**

| book_id | title | kind | author_id |
|---|---|---|---|
| B-000016 | Crónica de una muerte anunciada | Novel | ? |
| B-000017 | Siempre NO | Theatre | ? |
| B-000008 | Doce cuentos peregrinos | Tale | ? |
| B-000001 | El club de los suicidas | Novel | ROST |
| B-000004 | Poemas | Poetry | BERU |

| author_id | name |
|---|---|
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

# 2.4.6.- Other mechanisms for representing IC

- *General integrity constraints:* are those which cannot be expressed with the predefined constraints seen before. This can be:
  - *Static integrity constraints* (CREATE ASSERTION ...).
  - *Transition integrity constraints* (Trigger).

- A database is **valid (it is in a consistent state)**, if all the defined integrity constraints are satisfied.

- Checking the general constraints and all the other constraints (non-null, uniqueness, domain constraints, FK, PK, ...) is responsibility of the DBMS, which must ensure that every update in the database generates a new extension which satisfies all the constraints.

# 2.4.7.- Summary

The definition of a relation can be enriched with the following constraints:

- NNV: non-null value constraint,
- Uni: uniqueness constraint,
- PK: primary key,
- FK: foreign key (including restoration directives),
- general integrity constraints.

# 2.4.7.- Summary

**Example:** (generic relational schema)

$S(B_1:E_1, B_2:E_2, \ldots, B_t:E_t)$

PK: $\{B_j, \ldots, B_n\}$

Uni: $\{B_q, \ldots, B_r\}$

NNV: $\{B_s, \ldots, B_t\}$

$R(A_1:D_1, A_2:D_2, \ldots, A_r:D_r)$

PK: $\{A_i, \ldots, A_m\}$

FK: $\{A_o, \ldots, A_p\} \rightarrow S$

f: $A_o \rightarrow B_j$

$\ldots$

$A_p \rightarrow B_n$

on deletion cascade

on update set to null