DB con arepa negra

Nunca imaginé que las bases de datos serían tan raras desde aquella vez que guardamos la carne del almuerzo en ellas. Solo un bodrio explicaría dos páginas en tres meses.

1 Creación de tablas

Una sintaxis de ejemplo para crear tablas:

```
create table slaves (
        id serial primary key,
        username varchar(50) unique,
        email varchar(100) not null,
        created_at timestamp default now(),
        age int default 20
);
```

2 Consultas básicas

[C] Insertar registros:

```
insert into slaves (username, age, email)
        values ("magumba", 23, "magumba@ucaldas.com
            "),
        ("jairo", 71, "jairo@ucaldas.com"),
        ("el␣negro␣del␣whatsapp", 40, "mostro@ucaldas.
            com");
```

[R] Seleccionar registros:

```
select * from slaves;

select * from slaves where age > 15;

select username from slaves order by age desc;
```

[U] Actualizar registros:

```
update slaves
        set age = 25
        where id = 2;
```

[D] Borrar registros:

```
delete from slaves
        where age = 25;
```

3 Restricciones

Hay muchas patico:

- **PRIMARY KEY**: clave primaria, identifica filas de forma única.
- **UNIQUE**: no se repite ese valor.
- **NOT NULL**: obliga a tener valor.
- **CHECK**: condición lógica.
- **DEFAULT**: valor por defecto.
- **REFERENCES**: llave foránea.

```
create table slaves (
        id serial primary key, -- LLave primaria
        username varchar(50) unique, -- Campo único
        email varchar(100) not null, -- No nulo
        age int default 20 check (age > 18) -- No al
            trabajo infantil
        foreign key (camp_id) references camps(id), --
            Llave foranea
        created_at timestamp default now()
);
```

4 Compuestos

Clave primaria compuesta:

```
CREATE TABLE enrollments (
        student_id INT,
        course_id INT,
        grade CHAR(2),
        PRIMARY KEY (student_id, course_id) --
            clave compuesta
);
```

Campos compuestos únicos:

```
CREATE TABLE friendships (
        user_id INT,
        friend_id INT,
        UNIQUE (user_id, friend_id)
);
```

5 Comportamientos de llave foranea

No action: (default) si hay referencias, impide borrar/actualizar.
Restrict: parecido a **No action**, pero se evalúa inmediatamente (no al final de la transacción)

```
...
foreign key (camp_id)
        references camps(id)
        on delete cascade -- Si borra el campo, no
            deje
        on update cascade, -- Si actualiza ID de
            campo, no deje tampoco
...
```

Cascade: se propaga: si borras el padre, se borran los hijos; si actualizas, también se actualiza.

```
...
foreign key (camp_id)
        references camps(id)
        on delete cascade -- Si borra el campo, sus
            esclavos también
        on update cascade, -- Si actualiza ID de
            campo, actualiza la referencia
...
```

Set null: pone el valor en NULL cuando se borra/actualiza el padre.

```
...
foreign key (camp_id)
        references camps(id)
        on delete cascade -- Si borra el campo, setea
            a null
        on update cascade, -- Si actualiza ID de
            campo, setea a null
...
```

Set default: pone el valor por defecto definido en la columna.

...
foreign key (camp_id)
references camps(id)
on delete set default −− *Si borra el campo, setea a default*
on update set default, −− *Si actualiza ID de campo, setea a default*
...

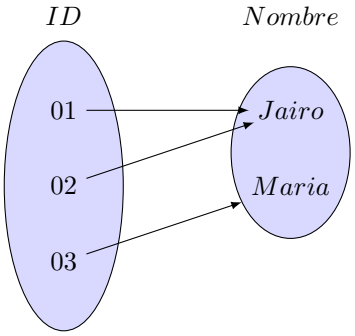
6 Dependencia funcional

En una relación R (una tabla), decimos que existe una dependencia funcional:

$$X \to Y$$

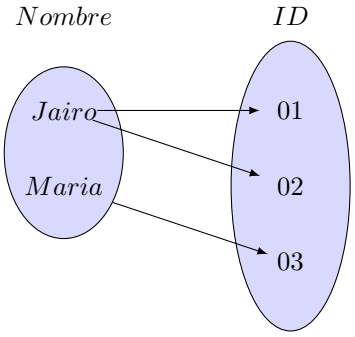
Sin hablar como un matemático subnormal, esto quiere decir que si tengo el valor de un campo (X), puedo determinar un único valor para otro campo (Y) con seguridad. Un ejemplo es una tabla de estudiantes con **id** y **nombre**:

Si me dan la id de un estudiante (X), puedo saber con seguridad su nombre (Y):



$id \to nombre$

En cambio, si me dan el nombre de un estudiante (Y), no puedo llegar a una única id, puede que esxistan Jairo de sistemas con id 01, y Jairo de artes con id 02:



$nombre \to id, no\ se\ cumple$

7 Dependencia parcial

Ocurre cuando se usan campos compuestos y un campo (Y), se puede determinar a partir de solo una parte de otro campo compuesto (X - clave primaria normalmente), por ejemplo:

$$PK(student\_id, course\_id) \to student\_name$$

En este caso sólo student_id bastaría para determinar un student_name específico.

Esto sirve para formas normales y nos puede decir si un campo debería ir en una tabla separada.

8 Dependencia funcional completa

Esto ocurre al usar claves compuestas, cuando un campo (Y) solo puede determinarse usando la totalidad de otro campo compuesto (X - clave primaria normalmente), por ejemplo, en el caso de los estudiantes:

$$PK(student\_id, course\_id) \to student\_grade$$

En este caso necesitamos toda la clave primaria para determinar la calificación de curso ara un estudiante, en los otros casos tendremos problemas:

- Si usamos solo student_id, nos resulta una lista de calificaciónes para todos sus cursos inscritos, imposible saber cuál escoger.
- Si usamos solo course_id, nos resultan calificaciónes para todos los estudiantes inscritos en el curso.

9 Formas normales

9.1 1NF

Una table está en **1NF** si cumple que:

- Todas sus columns tienes valore atómicos (no arrays, ni info desordenada separada por comas).
- Cada fila es única.
- Cada columna tiene un valor único.
- El orden en que se almacenan los datos no importa.

9.2 2NF

Una tabla está en **2NF** si está en **1NF** y no tiene **dependencias parciales**, significando que todo campo no-primo (que no sea llave) debe depender de toda la clave primaria.

9.3 3NF

Una tabla está en **3NF** si cumple la **2NF** y no hay **dependecias transitivas**, es decir, si ningún campo no-llave depende funcionalmente de otro campo no-llave.

10 Lecturas adicionales:

Las 7 formas normales.