Imagen que contiene interior, tabla, computadora, reloj

Descripción generada automáticamente

BASE DE DATOS RELACIONALES

**Informe**

**Hackaton:** 08

**Nombre:** María Paula Tomaylla

**Código:** SV42107644

**Introducción**

A lo largo de los años, la necesidad de almacenar, gestionar y acceder a grandes volúmenes de información ha llevado al desarrollo de sistemas de bases de datos. Durante los primeros años de era informática, la información se almacenaba en archivos planos, lo cual era ineficiente y susceptible a errores. A medida que las organizaciones comenzaron a manejar más datos, se hizo evidente la necesidad de sistemas que permitieran manejar la información de una manera más estructurada y confiable.

En la década de 1960, los primeros sistemas de bases de datos jerárquicas y de red comenzaron a tomar forma, representando un paso significativo respecto a los archivos planos tradicionales. Estos sistemas ofrecían una organización más estructurada y permitían manejar datos de una manera más ordenada. Sin embargo, a pesar de sus ventajas, eran complicados y difíciles de gestionar debido a su rigidez en la administración de relaciones entre los datos.

En los años 1970, se produjo un avance, cuando Edgar F. Codd, un investigador de IBM, revolucionó el ámbito al proponer el modelo relacional. Este modelo introdujo una manera mucho más sencilla y flexible de organizar los datos mediante el uso de tablas. Las tablas en el modelo relacional permiten almacenar datos en filas y columnas, donde cada fila representa un registro único y cada columna un atributo del dato.

La verdadera innovación del modelo relacional fue la capacidad de establecer relaciones entre diferentes tablas utilizando claves primarias y foráneas. Esto facilitaba la gestión, búsqueda y manipulación de la información. El enfoque no solo simplificó la estructura de las bases de datos, sino que también mejoró su eficiencia y accesibilidad, sentando las bases para los sistemas de gestión de bases de datos modernos.

El modelo relacional vino acompañado con el proceso de normalización, cual tiene como objetivo minimizar la redundancia y maximizar la integridad de los datos. El proceso implica la división de una base de datos en tablas más pequeñas y estrechamente relacionadas entre sí, reduciendo la duplicación de datos y simplificando las tareas de actualización y mantenimiento. La normalización se basa en un conjunto de formas normales, cada una con criterios específicos diseñados para asegurar que los datos se mantengan organizados y estructurados de manera eficiente.

Codd también introdujo las 12 Reglas de Codd, que establecen los principios básicos que debe cumplir un sistema de gestión de bases de datos relacional para ser considerado completo y eficiente. Estas reglas aseguran que los datos sean consistentes, accesibles y manipulables de manera coherente.

En este informe, revisaremos las 12 Reglas de Codd con algunos ejemplos para ilustrar su aplicación práctica. También las diversas formas de normalización y finalmente se comentará sobre el modelo relacional en su conjunto y su impacto en el modelado y la gestión de bases de datos.

# **12 Reglas de Cood**

Edgar F. Codd, al formular sus 12 reglas pretendió definir qué características debe tener un sistema de gestión de bases de datos para que se pueda considerar verdaderamente relacional. Estas reglas están diseñadas para garantizar que un sistema de bases de datos pueda manejar la información de manera eficiente y precisa, adhiriéndose al modelo relacional propuesto por Codd. A través de estas reglas, Codd buscaba:

* Definir el concepto de base de datos relacional, estableciendo una serie de principios fundamentales que un sistema de gestión de bases de datos debería seguir para calificar como relacional.
* Establecer estándares para la implementación, proporcionando un marco para evaluar si los sistemas existentes o en desarrollo cumplen con los requisitos de un sistema relacional.
* Garantizar la integridad y consistencia de los datos, lo que asegura que la estructura de datos y las operaciones sobre estos datos sean coherentes y mantengan la integridad.
* Promover la simplicidad en el diseño, para la implementación de bases de datos relacionales sea más intuitiva y menos propensa a errores.
* Fomentar la independencia de los datos, con el fin de que los cambios en la estructura de la base de datos no afecten a las aplicaciones que la utilizan.

A continuación las reglas.

## **Regla de la Información**

Todos los datos en una base de datos relacional deben representarse explícitamente como valores en tablas. Esto significa que cualquier dato accesible debe estar almacenado en una tabla y ser representado por valores que las aplicaciones pueden manipular.

CREATE TABLE tb\_product (id\_product INT PRIMARY KEY, name VARCHAR(90), quantity INT, price FLOAT, id\_category INT)

--Esto creará una tabla con 5 columnas, donde el id\_product es la llave principal:

| id\_product | name | quantity | price | id\_category |

| | | | | |

## **Regla del Acceso Garantizado**

Cada dato en una base de datos relacional debe ser accesible mediante una combinación de la tabla en la que reside, la clave primaria y el nombre de la columna. Esto garantiza que los datos puedan ser localizados de manera precisa y sin ambigüedad.

SELECT quantity, price

FROM tb\_product

WHERE id\_product = 1021;

--Podemos acceder a los datos a través de su llave primaria

## **Regla del Tratamiento Sistemático de Valores Nulos**

Los valores nulos deben ser gestionados de manera consistente para representar información desconocida o inaplicable. Esto incluye su diferenciación de los valores vacíos y el tratamiento adecuado en las operaciones de búsqueda y manipulación de datos.

CREATE TABLE tb\_product (

id\_product INT PRIMARY KEY,

name VARCHAR(90),

quantity INT,

price FLOAT,

id\_category INT,

registration\_date DATE,

modification\_date DATE – Este valor puede ser nulo

);

INSERT INTO tb\_product (

id\_product,

name,

quantity,

price,

id\_category,

registration\_date,

modification\_date

) VALUES

(1, 'Falda', 3, 38.90, 2, GETDATE(), NULL),

(2, 'Pantalón', 4, 69.90, 2, GETDATE(), NULL),

(3, 'Polo', 6, 15.00, 2, GETDATE(), NULL),

-- Fecha de modificación es nulo

## **Catálogo Dinámico en Línea Basado en el Modelo Relacional**

La descripción de la base de datos debe estar almacenada de forma dinámica y accesible para los usuarios, utilizando el mismo lenguaje relacional que se emplea para acceder a los datos. Esto permite a los usuarios consultar y manipular la estructura de la base de datos de manera eficiente.

CREATE TABLE tb\_product (

id\_product INT PRIMARY KEY,

name VARCHAR(90),

quantity INT,

price FLOAT,

id\_category INT,

registration\_date DATE,

modification\_date DATE

);

CREATE TABLE tb\_category (

id\_category INT PRIMARY KEY,

name VARCHAR(50)

);

CREATE TABLE tb\_product\_category (

id\_product INT,

id\_category INT,

FOREIGN KEY (id\_product)

REFERENCES tb\_product(id\_product),

FOREIGN KEY (categoryID)

REFERENCES tb\_category (id\_category),

PRIMARY KEY (id\_product, id\_category)

);

--Acceso a la información

SELECT

p.name AS product\_name,

p.quantity AS product\_quantity,

p.price AS product\_price

FROM

tb\_product p

JOIN

tb\_products\_category pc ON p.id\_product = pc.id\_product

JOIN

tb\_category c ON pc.id\_category = c.id\_category

WHERE

c.name = 'Ropa Mujer';

## **Regla del Sub-Lenguaje de Datos Completo**

Debe existir un lenguaje de datos completo que soporte la definición, manipulación y control de los datos, con capacidades de integridad y control de transacciones. SQL es un ejemplo de este tipo de lenguaje.

SELECT name, id\_category

FROM tb\_product

WHERE id\_category in ( 1, 2, 3) and quantity > 0 ;

--Podemos realizar una consulta compleja que involucre varias operaciones

## **Regla de Actualización de Vistas**

Todas las vistas que son teóricamente actualizables deben ser actualizables por el sistema. Las vistas son tablas virtuales que resultan de una consulta y deben permitir las mismas operaciones de actualización que las tablas físicas.

UPDATE tb\_product

SET discount = price \* 0.5

WHERE id\_category = 1;

--Podemos actualizar una vista y los cambios se reflejan correctamente

## **Inserción, Actualización y Borrado de Alto Nivel**

Un sistema de bases de datos relacional debe soportar operaciones de inserción, actualización y borrado que se apliquen a conjuntos de datos (tuplas) de manera grupal, no solo a nivel individual. Esto aumenta la eficiencia y flexibilidad en la manipulación de datos.

--Add new products

INSERT INTO tb\_product (

id\_product,

name,

quantity,

price,

id\_category,

registration\_date,

modification\_date

) VALUES

(1, 'Falda', 3, 38.90, 2, GETDATE(), NULL),

(2, 'Pantalón', 4, 69.90, 2, GETDATE(), NULL),

(3, 'Polo', 6, 15.00, 2, GETDATE(), NULL),

--Update information

UPDATE tb\_product

SET id\_category = 8

WHERE id\_category = 2;

--Delete user

DELETE FROM tb\_product

WHERE id\_category = 5 and quantity = 0;

## **Independencia Física de los Datos**

Los cambios en la representación física de los datos (cómo y dónde están almacenados) no deben afectar la capacidad de los usuarios para acceder a ellos. Esto permite que las mejoras en el almacenamiento físico no impacten en las aplicaciones existentes.

ALTER TABLE tb\_product

ADD COLUMN description VARCHAR(255);

--Podemos cambiar la forma en que se almacenan los datos sin afectar las consultas

## **Independencia Lógica de los Datos**

Los cambios en la estructura lógica de la base de datos no deben afectar a las aplicaciones que utilizan la base de datos. Esto garantiza que las aplicaciones puedan seguir funcionando sin modificaciones, incluso si la estructura de la base de datos cambia.

ALTER TABLE tb\_product

ADD COLUMN status CHAR(1);

--Podemos cambiar la estructura de los datos sin afectar las aplicaciones.

## **Independencia de la Integridad**

Las restricciones de integridad deben ser definibles por el sistema y almacenadas independientemente de las aplicaciones que acceden a los datos. Esto asegura que las reglas de integridad de los datos se mantengan, independientemente de cómo se acceda a ellos.

ALTER TABLE tb\_product

ADD CONSTRAINT ck\_deleted

CHECK (deleted\_date is not NULL);

--Podemos establecer una regla de integridad sin afectar las aplicaciones

## **Independencia de la Distribución**

La distribución de los datos en diferentes ubicaciones debe ser transparente para los usuarios. Esto significa que los usuarios no necesitan conocer la ubicación física de los datos para poder acceder a ellos.

--Ejemplo de un link server

EXEC master.dbo.sp\_addlinkedserver @server = N'HIST\_SQL', @srvproduct=N'SQL Server'

SELECT \* FROM HIST\_SQL.store.dbo.tb\_hist\_product

--Podemos acceder a datos de otras ubicaciones

## **Regla de No Subversión**

Ningún nivel inferior de acceso a los datos (por ejemplo, mediante lenguajes de bajo nivel o interfaces de programación) debe subvertir las restricciones y reglas establecidas por el sistema de bases de datos relacional. Esto garantiza que todas las reglas y restricciones se apliquen uniformemente, sin importar cómo se accede a los datos.

DELETE FROM tb\_products WHERE deleted\_date < cast('01/01/2024' AS datetime);

--Aun con acceso avanzado, se debe seguir respetando restricciones

# **Normalización de Base de Datos**

La normalización es un proceso que organiza las bases de datos para minimizar la redundancia y asegurar la integridad de los datos. A través de la normalización, los datos se estructuran en formas normales, que son niveles de organización que aumentan en rigor y complejidad.

## **Primera Forma Normal (1NF)**

Establece que la tabla debe estar libre de duplicados y cada campo debe contener solo valores atómicos, es decir, indivisibles. Esto significa que cada columna debe contener un solo valor para cada fila.

Ejemplo:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id\_employee | name | last\_name | jan\_salary | feb\_salary | mar\_salary |
| 1 | Ernesto | Tucto | 3500 | 3500 | 3500 |
| 2 | Miguel | Vargas | 2800 | 2800 | 2800 |
| 3 | Sofía | Silva | 3900 | 3900 | 3900 |

Mejora, modificamos a una sola columna **monthly\_salary** para no repetir datos:

|  |  |  |  |
| --- | --- | --- | --- |
| id\_employee | name | last\_name | monthly\_salary |
| 1 | Ernesto | Tucto | 3500 |
| 2 | Miguel | Vargas | 2800 |
| 3 | Sofía | Silva | 3900 |

## **Segunda Forma Normal (2NF)**

Para cumplir con 2NF, la tabla debe estar en 1NF y todos los atributos no clave deben depender completamente de la clave primaria. Esto elimina dependencias parciales, donde un atributo no clave depende solo de una parte de una clave primaria compuesta.

Ejemplo:

|  |  |  |  |
| --- | --- | --- | --- |
| id\_branch | id\_product | quantity | branch\_address |
| 1 | 1 | 6 | Av. Universitaria 394 |
| 2 | 1 | 30 | Av. Javier Prado 1658 |
| 3 | 1 | 100 | Av. Benavides 547 |
| 1 | 2 | 50 | Av. Universitaria 394 |
| 2 | 2 | 30 | Av. Javier Prado 1658 |
| 3 | 2 | 10 | Av. Benavides 547 |

Mejora, la columna **address** debe estar la tabla **branch** para no repetir datos:

|  |  |  |
| --- | --- | --- |
| id\_branch | id\_product | quantity |
| 1 | 1 | 6 |
| 2 | 1 | 30 |
| 3 | 1 | 100 |

|  |  |
| --- | --- |
| id\_branch | branch\_address |
| 1 | Av. Universitaria 394 |
| 2 | Av. Javier Prado 1658 |
| 3 | Av. Benavides 547 |

## **Tercera Forma Normal (3NF)**

Una tabla está en 3NF si está en 2NF y no existen dependencias transitivas entre atributos no clave. Esto significa que los atributos no clave no deben depender de otros atributos no clave.

Ejemplo, siendo la clave primaria id\_employee y si alguien agrega la columna department\_name:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id\_employee | name | last\_name | monthly\_salary | id\_department | department\_name |
| 1 | Ernesto | Tucto | 3500 | 1 | Sistemas |
| 2 | Miguel | Vargas | 2800 | 2 | Marketing |
| 3 | Sofía | Silva | 3900 | 1 | Sistemas |

Y luego se actualiza el valor de department\_name, con la clave primaria id\_employee se tendría:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| id\_employee | name | last\_name | monthly\_salary | id\_department | department\_name |
| 1 | Ernesto | Tucto | 3500 | 1 | Tecnología |
| 2 | Miguel | Vargas | 2800 | 2 | Marketing |
| 3 | Sofía | Silva | 3900 | 1 | Sistemas |

Mejora, la columna **department\_name** irá en la tabla **department** y se tendrá una tabla **employee\_department**:

|  |  |  |  |
| --- | --- | --- | --- |
| id\_employee | name | last\_name | monthly\_salary |
| 1 | Ernesto | Tucto | 3500 |
| 2 | Miguel | Vargas | 2800 |
| 3 | Sofía | Silva | 3900 |

|  |  |
| --- | --- |
| id\_department | department\_name |
| 1 | Tecnología |
| 2 | Marketing |
| 1 | Sistemas |

|  |  |
| --- | --- |
| id\_employee | id\_department |
| 1 | 1 |
| 2 | 2 |
| 3 | 1 |

## **Forma Normal de Boyce-Codd (BCNF)**

Una versión más estricta de 3NF, BCNF elimina cualquier dependencia no trivial entre atributos. Esencialmente, asegura que las únicas dependencias posibles en la tabla son aquellas donde un atributo depende completamente de la clave.

Ejemplo, si un empleado tiene habilidades y también habla varios idiomas. Existen dos relaciones, una entre empleados y habilidades y otra entre empleados e idiomas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| id\_employee | skill\_code | language\_code | skill\_level | language\_level |
| 1 | 1 | 1 | basic | basic |
| 2 | 1 | 1 | advanced | intermediate |
| 3 | 1 | 1 | intermediate | advanced |

Mejora, se debe evitar esta violación creando dos entidades una para cada relación:

|  |  |  |
| --- | --- | --- |
| id\_employee | skill\_code | skill\_level |
| 1 | 1 | basic |
| 2 | 1 | advanced |
| 3 | 1 | intermediate |

|  |  |  |
| --- | --- | --- |
| id\_employee | language\_code | language\_level |
| 1 | 1 | basic |
| 2 | 1 | intermediate |
| 3 | 1 | advanced |

## **Cuarta y Quinta Formas Normales (4NF y 5NF)**

Estas formas tratan problemas de dependencias múltiples y de proyección-join. La 4NF elimina dependencias multivaluadas, y la 5NF asegura que las dependencias de unión no redundantes están completamente descompuestas.

Ejemplo, empleados a proyectos específicos con ciertos roles:

|  |  |  |
| --- | --- | --- |
| id\_employee | id\_project | id\_role |
| 1 | 101 | 1001 |
| 1 | 102 | 1002 |
| 2 | 101 | 1001 |
| 2 | 103 | 1003 |
| 3 | 102 | 1002 |
| 3 | 103 | 1003 |

Cada fila indica que un empleado está asignado a un proyecto con un rol específico. Esta tabla podría estar en 4NF, pero aún podría tener dependencias de unión redundantes.

Mejora, para asegurarnos de que no existen dependencias de unión que causen redundancia, podríamos dividir la tabla en tres tablas independientes:

employee\_project: Relaciona empleados con proyectos

|  |  |
| --- | --- |
| id\_employee | id\_project |
| 1 | 101 |
| 1 | 102 |
| 2 | 101 |
| 2 | 103 |
| 3 | 102 |
| 3 | 103 |

employee\_role: Relaciona empleados con roles

|  |  |
| --- | --- |
| id\_employee | id\_role |
| 1 | 1001 |
| 1 | 1002 |
| 2 | 1001 |
| 2 | 1003 |
| 3 | 1002 |
| 3 | 1003 |

project\_role: Relaciona proyectos con roles

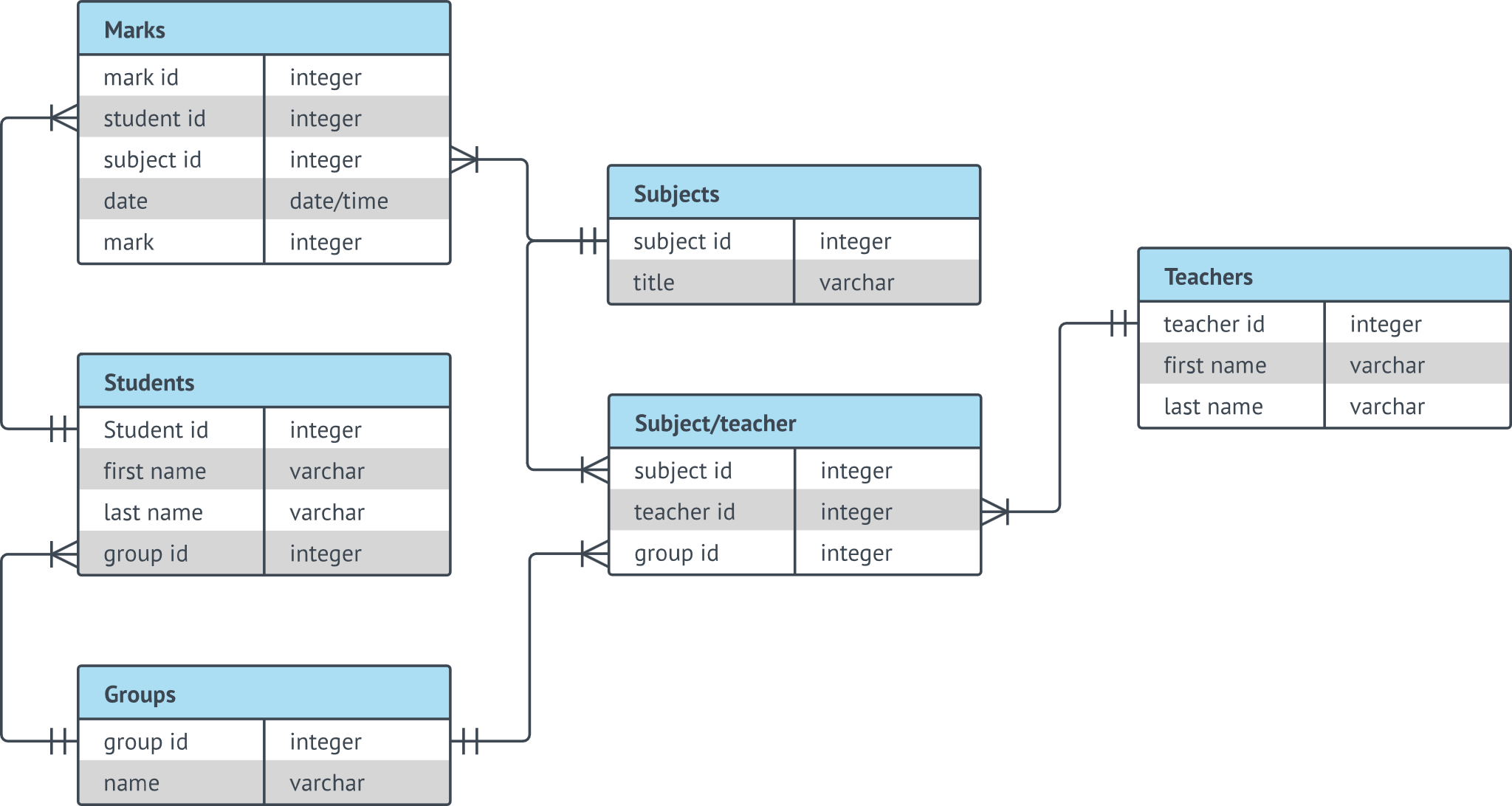
|  |  |
| --- | --- |
| id\_project | id\_role |
| 101 | 1001 |
| 102 | 1002 |
| 101 | 1001 |
| 103 | 1003 |
| 102 | 1002 |
| 103 | 1003 |

# **El Modelo Relacional para el Modelado y la Gestión de Bases de Datos**

El modelo relacional, propuesto por Edgar Frank Codd en 1970 desde los laboratorios IBM, rápidamente se estableció como un paradigma innovador en los modelos de bases de datos.

La idea central de este modelo es la utilización de relaciones, las cuales pueden conceptualizarse lógicamente como conjuntos de datos denominados tuplas. Aunque esta es la teoría de Codd sobre bases de datos relacionales, frecuentemente se representa de manera más accesible, imaginando cada relación como una tabla compuesta por registros, donde cada fila es una tupla, y columnas, también conocidas como campos.

En otras palabras cada tabla está compuesta por filas (tuplas) y columnas (atributos), y las relaciones entre tablas se definen a través de claves primarias y foráneas. Este modelo se basa en el álgebra relacional, un conjunto de operaciones que permite realizar consultas y manipular datos de manera eficiente. Las principales operaciones del álgebra relacional incluyen la selección, proyección, unión, diferencia y producto cartesiano, entre otras.



# **Historia y línea de tiempo**

# **Conclusión**

El desarrollo y la evolución de las bases de datos han sido fundamentales para el manejo eficiente de la información en la era digital. Las 12 reglas de Codd han ayudado a proporcionar una base para la creación y gestión de bases de datos relacionales. Las reglas brindan principios que permiten a asegurar la integridad, la consistencia y la eficiencia en el manejo de la información.

Por otro lado, la normalización ha permitido asegura que las bases de datos no sólo sean funcionales, sino también eficientes. Ayudando a eliminar redundancias y a mantener una estructura lógica, facilitando así la gestión y el mantenimiento de la base de datos.

Asimismo, el modelo relacional sigue siendo una de las metodologías más robustas y utilizadas en la industria, adaptándose y evolucionando con nuevas tecnologías y demandas.

A pesar de la aparición de nuevas tecnologías y metodologías en la gestión de datos, el modelo relacional sigue siendo uno de los enfoques más robustos y ampliamente utilizados en la industria. Su capacidad para adaptarse y evolucionar a la par del constante cambio tecnológico asegura que siga siendo una herramienta importante en la gestión de información en el mundo.

Tanto las 12 reglas de Codd como la normalización han demostrado ser fundamentales para el éxito y la eficacia de las bases de datos relacionales, y el modelo relacional continúa siendo una metodología de referencia en el campo de la gestión de datos.

# **Referencias**

* <https://www.dbadixit.com/12-reglas-de-codd/>
* <https://www.youtube.com/watch?v=HkgVLS8braQ>
* <https://es.wikipedia.org/wiki/12_reglas_de_Codd>
* <https://es.wikipedia.org/wiki/Normalizaci%C3%B3n_de_bases_de_datos>
* <https://www.ibm.com/docs/es/db2-for-zos/13?topic=modeling-normalization-in-database-design>
* <https://deyde.com/blog/normalizacion-datos/tipos-de-normalizacion-datos/>
* <https://www.timetoast.com/timelines/linea-de-tiempo-de-la-historia-y-evolucion-de-las-bases-de-datos>