Bases de Datos SQL

> **Base de Datos**
>
> *"Una colección compartida de datos lógicamente relacionados, junto con una descripción de estos datos (metadatos), que están diseñados para satisfacer las necesidades de información de una organización."* Connolly and Begg, 2005
>
> *"Une base de données est un ensemble de données modélisant les objets d'une partie du monde réel et servant de support à une application informatique"* Gardarin, 2003

Contenido

Modelo Relacional (SQL)

El **modelo relacional** fue propuesto en 1970 por Edgar Frank Codd (Laboratorios IBM). En este modelo los datos en forma de renglones y columnas (*tablas*), y mediante el uso del ***algebra relacional y la teoría de conjuntos***, permite realizar operaciones : **selección, proyección, unión, intersección y diferencia**, para manipular los datos almacenados en las tablas.

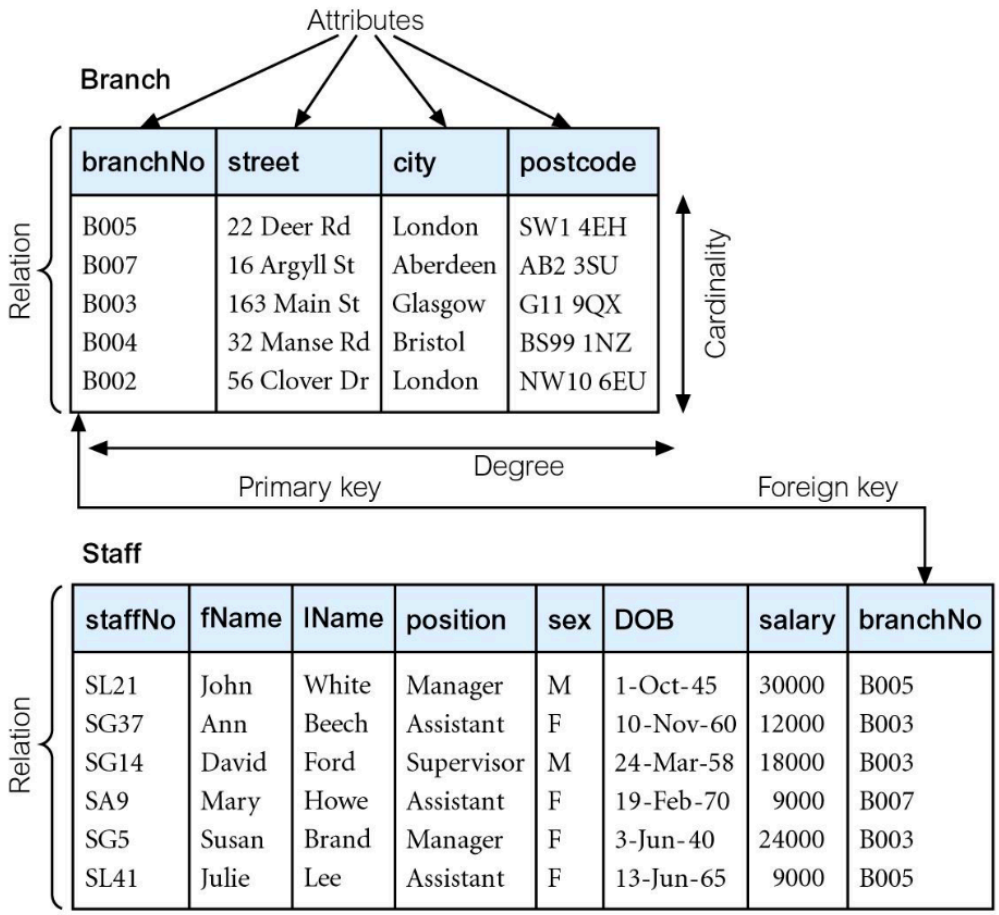
> (E. Codd.)*A Relational Model of Data for Large Shared Data Banks*, Commun. ACM, 13 (6): 377-387 (1970)
>
> Edgar F. Codd en 1970; premio Turing en 1986.

Componentes del ***Modelo Relacional***

- **Relación**, atributo, tupla, identificador, . . .
- **Álgebra relacional**
- **Cálculo relacional**
- **Reglas de normalización**

Relación

Los objetos del mundo real son representadas por las **relaciones**. Una relación es una matriz comúnmente llamada **tabla**.



Propiedades de las relaciones

- **Relación:** Es una tabla con columnas y filas
- **Atributo:** Es una columna en una relación
- **Dominio:** Es un conjunto de valores permitidos para uno o más atributos
- **Tupla:** Es una fila de una relación
- **Grado:** Es el número de atributos que contiene una relación
- **Cardinalidad:** Es el número de tuplas que contiene una relación

Base de Datos Relacional (*RDB, Relational Data Base*)

Es una colección de relaciones **normalizadas** en la que cada relación tiene un nombre distinto

$$RDB = (R_1, R_2, \ldots, R_n)$$

donde es esquema de una relación está dado por un conjunto de atributos y dominios

$$R = (a_1 : d_1, a_2 : d_2, \ldots, a_m : d_m)$$

o simplemente

$$R = (a_1, a_2, \ldots, a_m)$$

Ejemplo del esquema (estructura) de una Base de Datos Relacional

- **Branch** (*branchNo*, street, city, postcode)
- **Staff** (*staffNo*, fName, lName, position, sex, DOB, salary, branchNo)
- **PropertyForRent** (*propertyNo*, street, city, postcode, type, rooms, ownerNo, staffNo, branchNo)
- **PrivateOwner** (*ownerNo*, fName, lName, address, telNo)
- **Client** (*clientNo*, fName, lName, telNo, prefType, maxRent)
- **Viewing** (*clientNo*, propertyNo, viewDate, comment)
- **Registration** (*clientNo*, branchNo, staffNo, dateJoined)

Diagrama Entidad-Relación

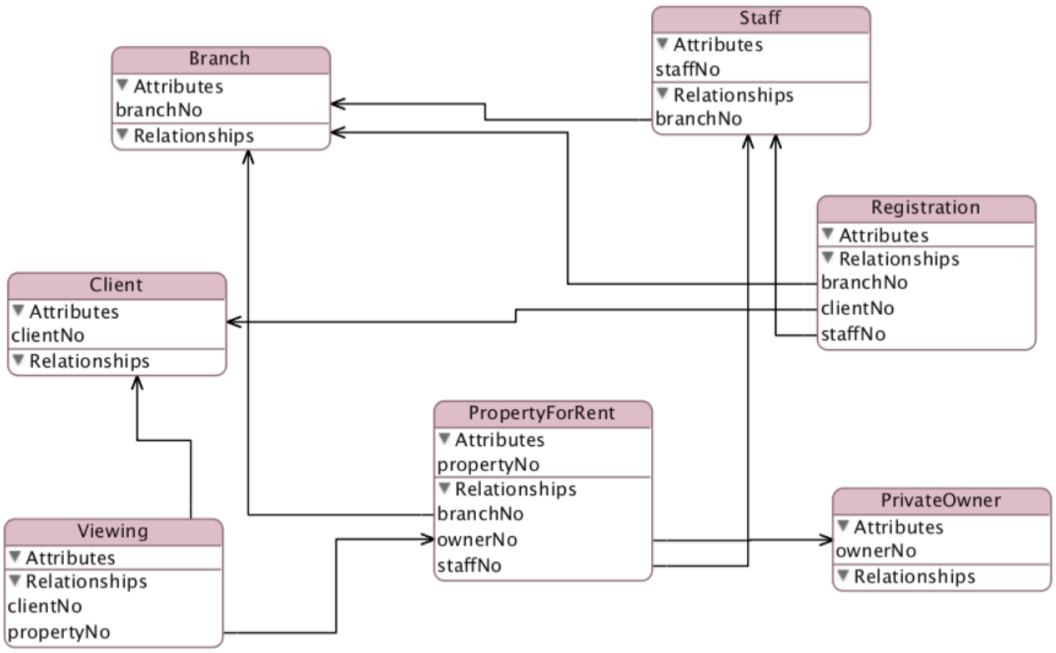
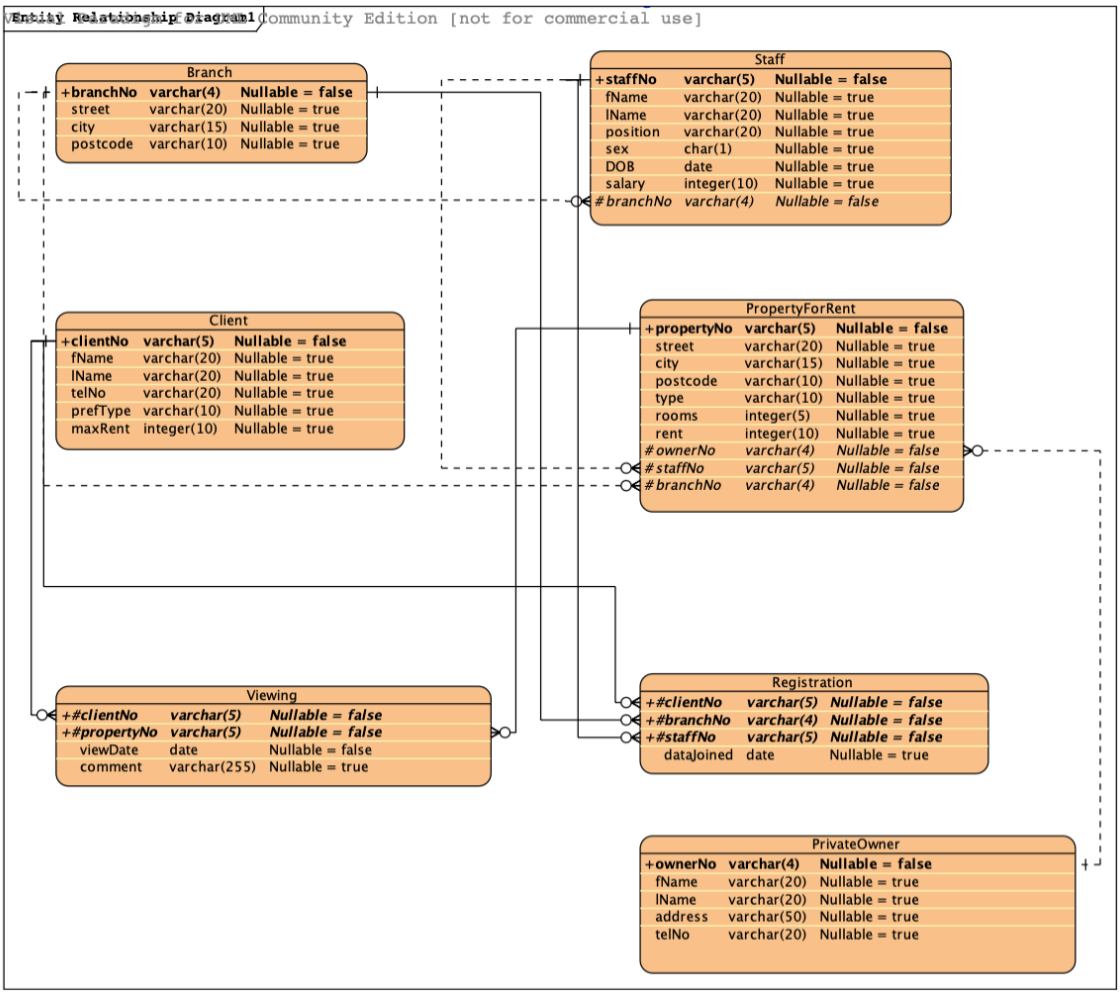


Diagrama Relacional



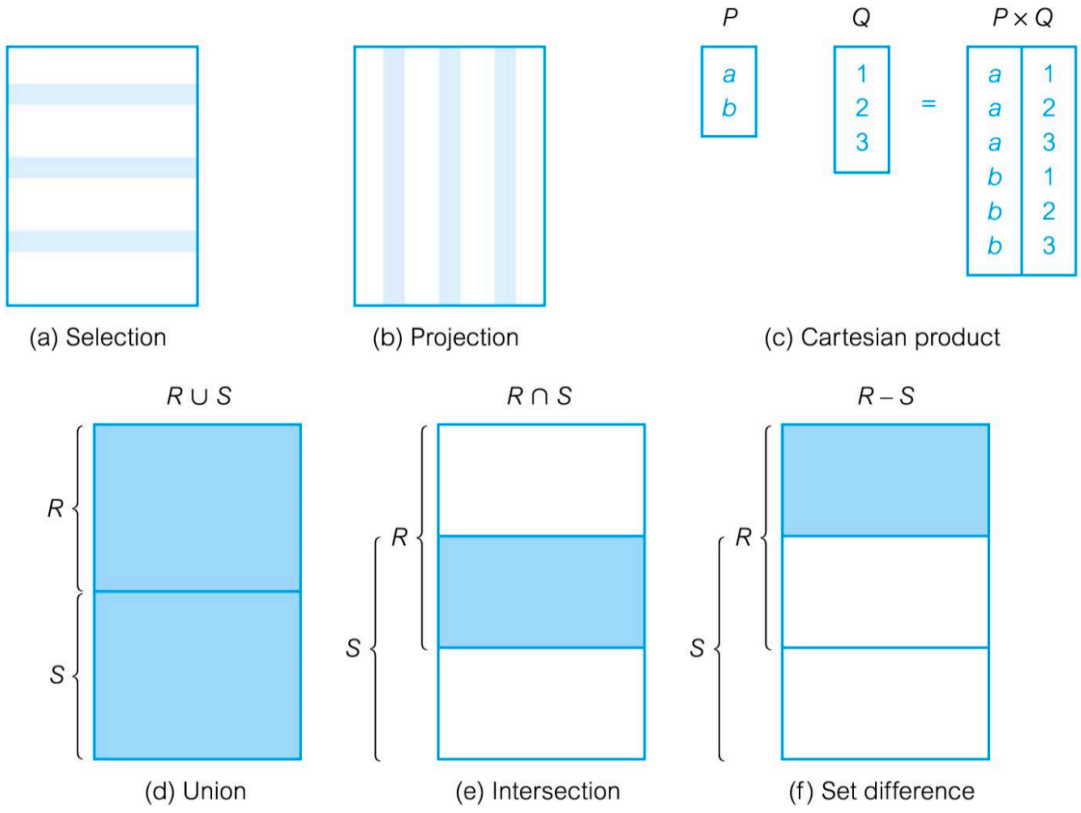
Inicialmente el modelo relacional fue propuesto con dos **lenguajes formales de manipulación de datos**:

- **Álgebra relacional**
- **Cálculo relacional**

Álgebra relacional

Es un conjunto de operadores que, a partir de una o dos relaciones existentes, dan como resultado una nueva relación temporal. La relación resultante tiene exactamente las mismas características que una relación de la base de datos y por lo tanto, también, puede ser manipulada por los operadores del álgebra.

Operaciones Básicas



Álgebra Relacional...
Selección σ . . .

Ejemplo: Listar al personal cuyo salario sea mayor a 10, 000.

staffNo	fName	lName	position	sex	DOB	salary	branchNo
SL21	John	White	Manager	M	1-Oct-45	30000	B005
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000	B003
SG14	David	Ford	Supervisor	M	24- Mar-58	18000	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000	B003

Algebra relacional

$$\sigma_{salary>10000}(Staff)$$

SQL

```
SELECT *
FROM   Staff
WHERE  salary > 1000;
```

Álgebra Relacional...

Proyección Π

Ejemplo: Lista del salarios para todo el personal, mostrando solo: *staffNo, fName, lName y salary*

staffNo	fName	lName	salary
SL21	John	White	30000
SG37	Ann	Beech	12000
SG14	David	Ford	18000
SA9	Mary	Howe	9000
SG5	Susan	Brand	24000
SL41	Julie	Lee	9000

Algebra relacional

$$\Pi_{staffNo,fName,lName,salary}(Staff)$$

SQL

```
SELECT staffNo, fName, lName, salary
FROM   Staff;
```

Álgebra Relacional...
Unión ∪

Ejemplo: Ciudades que tienen una sucursal o una propiedad en renta.

city
London
Aberdeen
Glasgow
Bristol

Algebra relacional

$$\Pi_{city}(Branch) \cup \Pi_{city}(PropertyForRent)$$

SQL

```
SELECT city
FROM   Branch
UNION
SELECT city
FROM   PropertyForRent;
```

Álgebra Relacional...

Diferencia de conjuntos –

Ejemplo: Listado de las ciudades que tienen una sucursal pero **NO** propiedades en renta.

city
Bristol

Algebra relacional

$$\Pi_{city}(Branch) - \Pi_{city}(PropertyForRent)$$

SQL

```
SELECT city
FROM   Branch
MINUS
SELECT city
FROM   PropertyForRent;
```

Álgebra Relacional...

Intersección ∩

Ejemplo: Listado de las ciudades en las que exista tanto una sucursal como al menos una propiedad en renta.

city
Aberdeen
London
Glasgow

Algebra relacional

$$\Pi_{city}(Branch) \cap \Pi_{city}(PropertyForRent)$$

SQL

```
SELECT city
FROM   Branch
INTERSECTION
SELECT city
FROM   PropertyForRent;
```

Álgebra Relacional...

Producto cartesiano *RXS*

Ejemplo: Enumerar los nombres y comentarios de todos los clientes que hayan visitado un inmueble en alquiler.

client.clientNo	fName	lName	Viewing.clientNo	propertyNo	comment
CR76	John	Kay	CR56	PA14	too small
CR76	John	Kay	CR76	PG4	too remote
CR76	John	Kay	CR56	PG4	
CR76	John	Kay	CR62	PA14	no dining room
CR76	John	Kay	CR56	PG36	
CR56	Aline	Stewart	CR56	PA14	too small
CR56	Aline	Stewart	CR76	PG4	too remote
CR56	Aline	Stewart	CR56	PG4	
CR56	Aline	Stewart	CR62	PA14	no dining room
CR56	Aline	Stewart	CR56	PG36	
CR74	Mike	Ritchie	CR56	PA14	too small
CR74	Mike	Ritchie	CR76	PG4	too remote
CR74	Mike	Ritchie	CR56	PG4	
CR74	Mike	Ritchie	CR62	PA14	no dining room
CR74	Mike	Ritchie	CR56	PG36	
CR62	Mary	Tregear	CR56	PA14	too small
CR62	Mary	Tregear	CR76	PG4	too remote
CR62	Mary	Tregear	CR56	PG4	
CR62	Mary	Tregear	CR62	PA14	no dining room
CR62	Mary	Tregear	CR56	PG36	

Algebra relacional

$$\Pi_{clientNo,fName,lName}(Client) X \Pi_{clientNo,propertyNo,comment}(Viewing$$

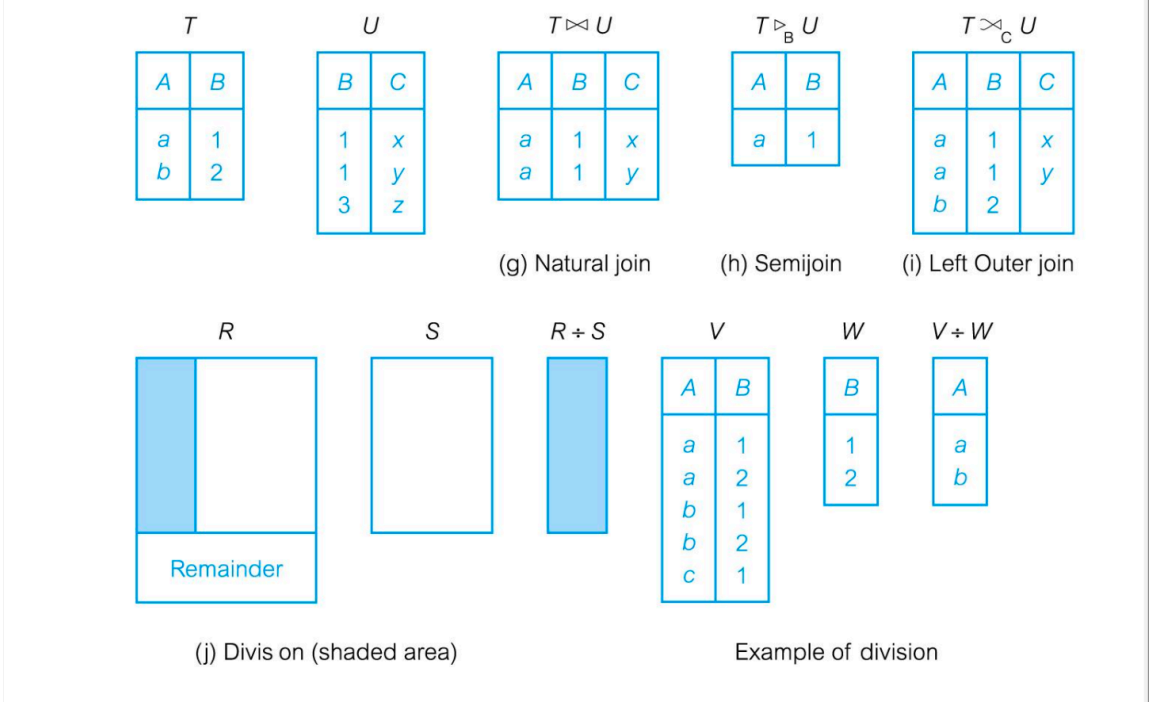
SQL

```
SELECT c.clientNo, c.fName, c.lName,
       v.clientNo, v.propertyNo, v.comments
FROM   Client c, Viewing v;
```

Operaciones Complejas

Álgebra Relacional...

Operaciones complejas



Cálculo relacional

Las consultas de cálculo relacional especifican qué hay que extraer, en lugar de como extraerlo.

Se basa en el cálculo de predicados (lógica de primer orden). Un predicado es una función booleana con argumentos. Cuando se asignan valores a los argumentos, la función nos proporciona una expresión, denominada proposición que puede ser verdadera o falsa.

Cálculo Relacional
de tuplas . . .

e.g. Extraer los valores de **staffNo, fName, lName, position, sex, DOB, salary** y **brachNo** para los empleados que ganes más de 10,000 dls.

$$\{S|Staff(S) \wedge S.salary > 10000\}$$

e.g. Extraer un atributo concretode los empleados que ganes más de 10,000 dls.

$$\{S.salary|Staff(S) \wedge S.salary > 10000\}$$

Cálculo Relacional
de tuplas . . .

Cuantificadores:

Existencial $\exists$ "**existe**".

e.g. Existe una tupla de **branch** que tiene el mismo valor de **banchNo** que el valor de **branchNo** correspondiente a la tupla actual de **Staff** *S* y cuya ciudad correspondiente es "Londres".

$$\{Staff(S) \wedge (\exists B)(Branch(B)) \wedge$$

$$(B.branchNo = S.branchNo) \wedge B.city = \text{''}London\text{''}\}$$

Universal $\forall$ "**para todo**".

e.g. Para todas la tuplas de **Branch**, la dirección no coresponde a "Paris".

$$(\forall B)(B.city \neq \text{''}Paris\text{''})\}$$

Normalización

El modelo de base de datos relacional constituye un ***modelo rígido* (o normalizado)**, dotado de una **estructura predefinida**. En este modelo se aplican las *formas normales*, las cuales son reglas de diseño que tienen por objetivo garantizar la **integridad**: y la eficiencia de la base de datos al **eliminar redundancias y dependencias indeseadas**.

El modelo relacional es soportado por lo general en las 3 primeras formas normales siguientes:

Primera Forma Normal (*1NF, Normal Form*):

- Una relación en la que, la intersección de toda fila y columna contiene un valor y solo un valor.

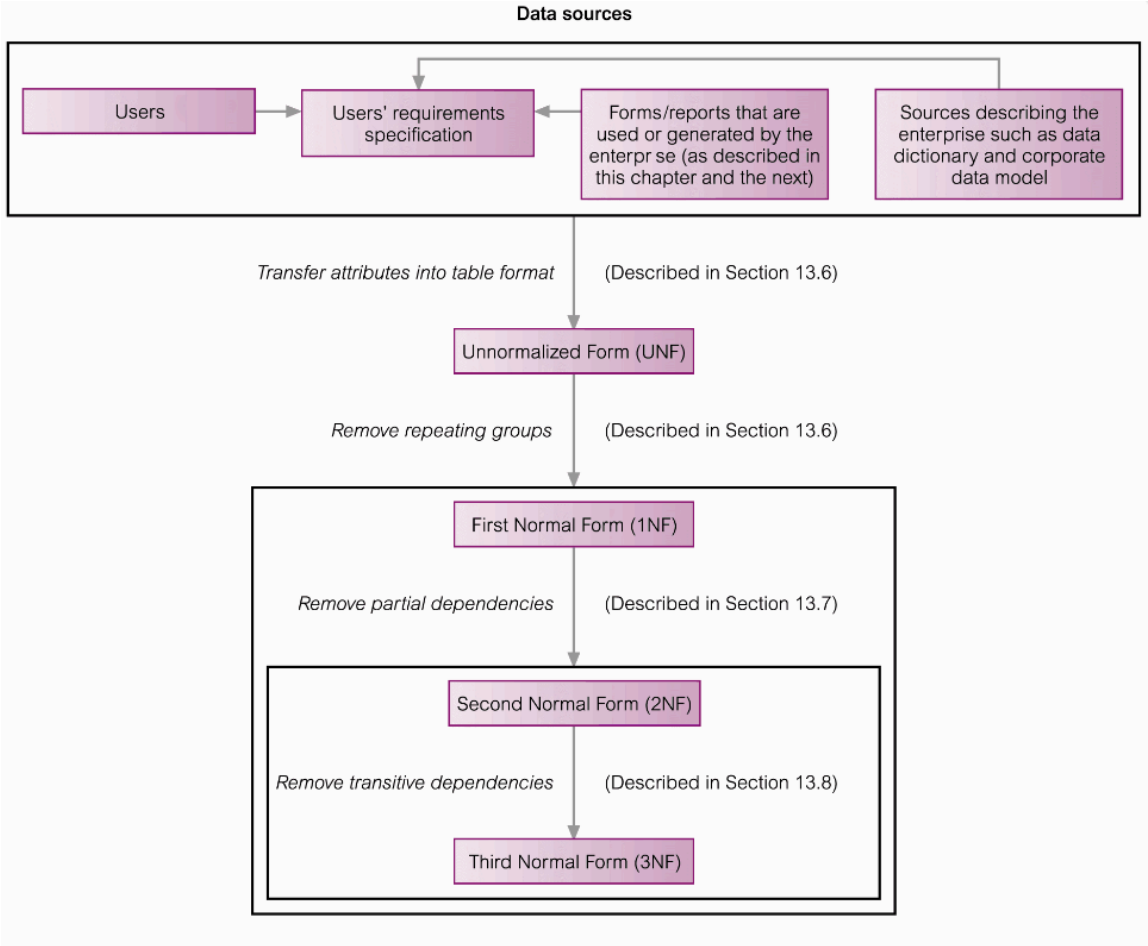
Segunda Forma Normal (2NF):

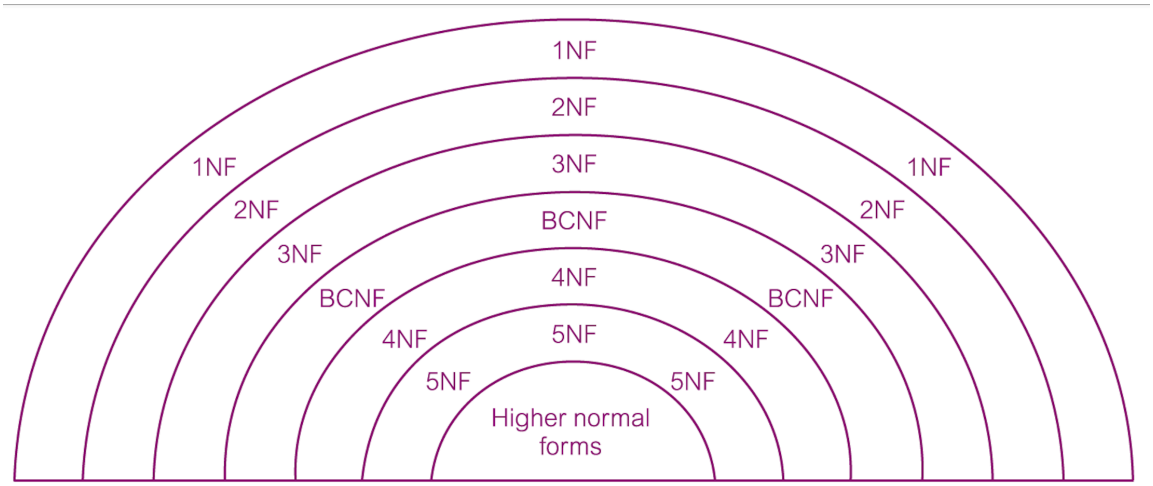
- Debe cumplir con 1NF.
- Todo atributo que no sea llave primaria depende funcionalmente de manera completa de la llave primaria (*PK, Primary Key*).

Tercera Forma Normal (3NF):

- Debe cumplir con 2NF.
- Ningún atributo que no sea PK depende transitivamente de la llave primaria. Lo que significa que, si un atributo depende de la PK, ningún otro atributo debe depender de ese atributo.

Esta forma tiene por objetivo eliminar la redundancia de datos y garantizar que todas las dependencias funcionales de los datos sean directas, con el **propósito de evitar anomalías en la inserción, actualización y eliminación de los datos**.





Ejemplo de normalización

Considera las compras realizadas por los siguientes clientes:

Nombre	Dirección	Compras
Juan Pérez Méndez	Calle A #100	Camiseta 10.00 2, Zapatos 30.00 1
María López Martínez	Calle B #200	Pantalones 20.00 1

Aplicando la **Primera Forma Normal (1NF)** para:

- **Elimina los grupos repetidos:** Cada columna debe contener un solo valor (*atomización de datos*). No se permiten listas o conjuntos de valores en una sola celda.
- **Los datos deben ser tabulares:** Cada fila debe tener el mismo número de columnas.

TABLA DE COMPRAS (1NF)

| Pedido_ID (PK) | Nombre_Cliente | Dirección_Cliente | Nombre_Producto | Precio_Producto | Cantidad | |----------|----------|----------|----------|----------|----------| | 1 | José Pérez Méndez | Calle A #100 | Camiseta | 10.00 | 2 | | 2 | José Pérez Méndez | Calle A #100 | Zapatos | 30.00 | 1 | | 3 | María López Martínez | Calle B #200 | Pantalones | $20.00 | 1 |

Transformación aplicando la **Segunda Forma Normal (2NF)**

Se eliminan las dependencias parciales, creando tablas separadas para cada *entidad* (**Cliente y Producto**) y dejando en la tabla de Pedidos solo las llaves foráneas (*FR, Foreign Key*) y las columnas que dependan completamente de la llave primaria compuesta:

TABLA DE CLIENTES (2NF)

ID (PK)	Nombre	Dirección
101	José Pérez Méndez	Calle A #100
102	María López Martínez	Calle B #200

TABLA DE PRODUCTOS (2NF)

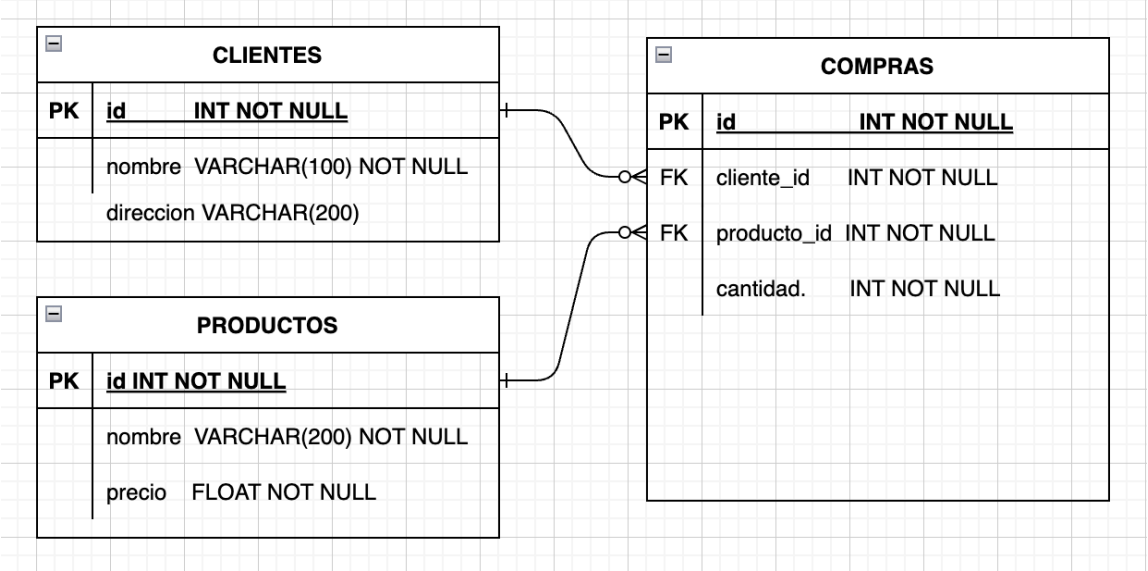
ID (PK)	Nombre	Precio
1001	Camiseta	$10.00
1002	Pantalones	$20.00
1003	Zapatos	$30.00

TABLA DE COMPRAS (2NF)

ID (PK)	Cliente_ID (FK)	Producto_ID (FK)	Cantidad
1	101	1001	2
2	101	1003	1
3	102	1002	1

Al aplicar la **2FN** en este ejemplo, todas las tablas ya cumplen con la **Tercera Forma Normal - 3NF**, ya que todas las columnas dependen directamente de la clave primaria (y foránea),

sin dependencias transitivas.



Conclusiones

EL modelo relacional o estructurado presenta ciertos problemas para representar entidades del mundo real (e.g. datos faltantes o incompletos, atributos multi-valuados o de diferentes tipos -heterogéneos -)

Referencias

- Connolly and Begg, 2005 Connolly, T. and Begg, C. (2005). Sistemas de bases de datos: un enfoque práctico para diseño, implementación y gestión. Pearson Educación.
- Gardarin, 2003 Gardarin, G. (2003). Bases de données. Best of Eyrolles. Eyrolles.

Sistema de Administración de Bases de Datos

En un programa, o conjunto de programas, que se usan para crear, actualizar y administrar bases de datos relacionales.

Los Sistemas Manejadores de Bases de Datos (*RDBMS, Relational Data Base Management System*) deben garantizar la **integridad**, **consistencia** y **confiabilidad** de los datos, mediante el adecuado manejo de **transacciones**.

Ejemplos:

- Oracle

- SQL Server
- PostgreSql
- SQLite, **RDBMS** empleado en el curso
  - Visual Studio Code
    - Extensiones: *SQLite*, *SQLite Viewer*, *SQLTools SQLite*

SQLite

- Descargar e instalar la base de datos SQLite
- Crear una base de datos desde la terminal (`cmd` en Windows) usando el comando `sqlite3` , llamada `compras.db`

```
terminal
❯ sqlite3 compras.db
SQLite version 3.43.2 2023-10-10 13:08:14
Enter ".help" for usage hints.
sqlite>
```

> Puede notar que el *cursor* en la terminal cambio a `sqlite` indicando que se encuentra en la base de datos, listo para manipularla con sentencias **CRUD**.

Consulte la lista de comandos mostrados en la sección 3 de la documentación de SQLite ("Command Line Shell For SQLite"), y practíquelos por ejemplo:

- Para mostrar las bases de datos creadas:

```
terminal
sqlite>. databases
```

- Para mostrar las tablas existentes en la base de datos con la que se encuentra trabajando

```
terminal
sqlite>. tables
```

- Para salir de SQLite

```
terminal
sqlite>. quit
~DB-Compras ❯
```

Transaction

In the context of databases and data storage systems, **a transaction is any operation that is treated as a single unit of work, which either completes fully or does not complete at all, and leaves the storage system in a consistent state**. The classic example of a transaction is what occurs when you withdraw money from your bank account. Either the money has left your bank account, or it has not — there cannot be an in-between state.

Proporties:

ACID is an acronym that refers to the set of 4 key properties that define a transaction: Atomicity, Consistency, Isolation, and Durability. If a database operation has these ACID properties, it can be called an ACID transaction, and data storage systems that apply these operations are called transactional systems. ACID transactions guarantee that each read, write, or modification of a table has the following properties:

- **Atomicity** - each statement in a transaction (to read, write, update or delete data) is treated as a single unit. Either the entire statement is executed, or none of it is executed. This property prevents data loss and corruption from occurring if, for example, if your streaming data source fails mid-stream.
- **Consistency** - ensures that transactions only make changes to tables in predefined, predictable ways. Transactional consistency ensures that corruption or errors in your data do not create unintended consequences for the integrity of your table.
- **Isolation** - when multiple users are reading and writing from the same table all at once, isolation of their transactions ensures that the concurrent transactions don't interfere with or affect one another. Each request can occur as though they were occurring one by one, even though they're actually occurring simultaneously.
- **Durability** - ensures that changes to your data made by successfully executed transactions will be saved, even in the event of system failure.

> A **transaction** is a logical unit of work composed of one or more *SQL-DML* statements (**read, write, update or delete**).
>
> - **COMMIT:** Causes the transaction to end successfully and the changes made to the database will be permanent. After COMMIT a new transaction will be started.
> - **ROLLBACK:** Aborts the transaction, undoing any changes made to the database.

Lenguaje Estructurado de Consulta (*SQL, Structured Query Language*)

> Es un lenguaje diseñado para administrar, y recuperar información de sistemas de administración de bases de datos relacionales.
>
> Originalmente basado en el **álgebra relacional y en el cálculo relacional**. El alcance de SQL incluye la **inserción** de datos, **consultas**, **actualizaciones** y **borrado**, la creación y modificación de esquemas y el control de acceso a los datos. También el SQL a veces se describe como un lenguaje declarativo, también incluye elementos procesales.
>
> SQL fue uno de los primeros lenguajes comerciales para el modelo relacional de Edgar Frank Codd como se describió en su artículo de investigación de 1970 El modelo relacional de datos para grandes bancos de datos compartidos.

SQL se divide en:

Lenguaje de Definición de Datos (*DDL, Data Definition Language*)

Permite crear y destruir objetos de la base de datos (esquemas, tablas, vistas, dominios) y sus principales sentencias son:

```
CREATE / DROP SCHEMA -- Oracle
CREATE / ALTER / DROP DOMAIN
CREATE / ALTER / DROP TABLE
CREATE / REPLACE /DROP VIEW
CREATE / DROP INDEX
```

> **Consultar sintáxis SQLite**

Esquemas

Los esquemas son una colección de objetos de la base de datos que están relacionados entre sí (e.g tablas, vistas, dominios, constraints, . . .); todos esos objetos de un **esquema** perteneciendo al mismo propietario.

Sintáxis

```
CREATE SCHEMA [Name | AUTHORIZATION CreatorId ]
DROP   SCHEMA Name [RESTRICT | CASCADE ]
```

Ejemplo

```
CREATE SCHEMA test AUTHORIZATION smith;
```

Tablas o Relaciones

Crea una tabla con una o más columnas de un tipo de datos especificado.

Sintáxis Oracle

```
CREATE TABLE TableName
{(colName dataType [NOT NULL][UNIQUE]
  [DEFAULT defaultOption]
  [CHECK searchCondition ][ , . . . ]}
  [PRIMARY KEY ( listOfColumns ) , ]
  {[UNIQUE (listOfColumns),][... ,]}
  {[FOREIGN KEY (listOfFKColumns)
    REFERENCES ParentTableName[( listOfCKColumns )],
    [ON UPDATE referentialAction]
    [ON DELETE referentialAction]][ ,...]}
   {[CHECK (searchCondition )] [ ,...] })
```

[*Sintáxis SQLite*][LTE-TABLE]

En las tablas se pueden aplicar las restricciones de integridad siguientes:

Integridad de entidades

- Se logra con la inclusión de la llave primaria (PK) en una tabla, misma que asegura un valor ÚNICO y NO-NULO por cada tupla o renglón.
- El estándar ISO soporta la cláusula `PRIMARY KEY`
- Sólo puede existir una cláusula `PRIMARY KEY` por tabla.
- Puede garantizarse la unicidad para la llaves alternas usando `UNIQUE`

Ejemplo

```
PRIMARY KEY (id)
PRIMARY KEY (id, producto_di)
```

Integridad referencial

- La llave foránea (FK) es una columna o un conjunto de columnas que ligan a un registro de una tabla (hija) conteniendo una FK, hacia un renglón de otra tabla (padre) que contiene la llave primaria (PK) correspondiente.
- La integridad referencial significa que, si FK contiene un valor, dicho valor debe referir a un registro existente en la tabla padre.
- El estándar ISO soporta la definición de FK's con la cláusula `FEREIGN KEY`
- Cualquier `INSERT o UPDATE` que intente *crear o modificar* FK, en una tabla hija, sin que exista una PK en la tabla padre, es rechazada.
- En caso de `DELETE o UPDATE` sobre la PK de una tabla padre se pueden efectuar las siguientes operaciones`
  - `CASCADE, SET NULL, SET DEFAULT, NO ACTION`

Ejemplo en Oracle

```
FOREIGN KEY (cliente_id) REFERENCES clientes(id);

FOREIGN KEY (producto_id) REFERENCES productos(id) ON DELETE SET NULL;
```

Datos requeridos

Ejemplo

```
nombre      VARCHAR(100) NOT NULL
```

Restricciones de dominio

Ejemplo Oracle

```
CREATE DOMAIN SexType AS CHAR
       CHECK(VALUE IN ('H','M'));

sexo     SexType    NOT NULL;
```

Restricciones generales

Son dictadas por aquellas reglas adicionales, especificadas por los usuarios o los administradores de la base de datos para definir o restringir algunos aspectos relacionados con la empresa.

Ejemplo Oracle

```
CREATE or REPLACE TRIGGER compras_AI
AFTER INSERT  ON compras FOR EACH ROW
DECLARE
    ...
BEGIN
    -- Código para actualizar el inventario de productos,
    -- cada vez que se venda una cantidad de ellos
    ...
EXCEPTION
    WHEN ...
    -- Manejo de excepciones
END;
```

Otros objetos: *VIEWS, INDEX, ROLES, SYNONYM, LINKS, ...*

Ejemplo "Compras"

Uso del Lenguaje de Definición de Datos ***DDL***

```
---------------------------------
-- DDL, Data Definition Language

CREATE TABLE clientes
(
    id          INTEGER PRIMARY KEY
   ,nombre      VARCHAR(100) NOT NULL
```

```
   ,direccion  VARCHAR(200)
);

CREATE TABLE productos
(
    id         INTEGER PRIMARY KEY
   ,nombre     VARCHAR(200) NOT NULL
   ,precio     FLOAT
);

CREATE TABLE compras
(
    id           INTEGER PRIMARY KEY
   ,cliente_id   INTEGER NOT NULL
   ,producto_id  INTEGER NOT NULL
   ,cantidad     INTEGER NOT NULL
   ,FOREIGN KEY (cliente_id)  REFERENCES clientes(id)
   ,FOREIGN KEY (producto_id) REFERENCES productos(id)
);
```

Lenguaje de Manipulación de Datos (*DML, Data Manipulation Language*)

> **Consultar sintáxis SQLite**

El **DML** incluye cuatro operaciones para manipular una base de datos: **Crear** (*Create*), **Leer** (*Read*), **Actualizar** (*Update*) y **Eliminar** (*Delete*), operaciones agrupadas con el acrónimo **CRUD**

Particularmente en SQL, las sentencias **CRUD** son las siguientes:

- **INSERT** para insertar nuevas filas o tuplas en una tabla

```
INSERT INTO TableName [(columnList)]
    VALUES (dataValueList)
INSERT INTO TableName [(columnList)]
    SELECT ...
```

- **UPDATE** para modificar los datos existentes en una tabla.

```
UPDATE  TableName
    SET    columnName1 = dataValue1
        [, columnName2 = dataValue2 ...]
[WHERE searchCondition]
```

- **DELETE** para eliminar filas de datos de una tabla.

```
DELETE FROM TableName
[WHERE searchCondition]
```

- **SELECT** para consultar información a partir de una o más tablas.

```
SELECT [DISTINCT | ALL]
       {* | [columnExpression [AS new Name]] [, ...]}
FROM   TableName [alias] [, ...]
[WHERE condition]
[GROUP BY columnList] [HAVING condition]
[ORDER BY columnList]
```

donde:

- **SELECT** indica qué columnas aparecerán en la consulta (*proyección*)
- **FROM** especifica la(s) tabla(s) de dónde se extraerá la información
- **WHERE** filtra los renglones de acuerdo an criterio de búsqueda (*selección*)
- **GROUP BY** forma grupos de renglones
- **HAVING** filtra grupos grupos de acuerdo con una condición
- **ORDER BY** especifica el orden en el que se mostrarán los datos de la consulta

Ejemplo "Compras"

Uso del Lenguaje de Manipulación de Datos **DML** en la tabla `clientes`

```
---------------------------------
-- DML, Data Manipulation Language
---------------------------------

---------------------------------
-- Creando datos con INSERT
INSERT INTO clientes
VALUES (101, "José Pérez Méndez", "Calle A # 100");
INSERT INTO clientes(id, direccion, nombre)
VALUES (102, "María López", "Calle B # 200");

---------------------------------
-- Leyendo datos con SELECT
SELECT *
FROM   clientes;

---------------------------------
-- Actualizando datos con UPDATE
UPDATE clientes SET nombre="María López Martínez"
WHERE  id=102

---------------------------------
-- Borrando datos con UPDATE
DELETE FROM clientes;
```

```sql
---------------------------------
-- Insertando datos en la tabla ```productos```

INSERT INTO productos (id, nombre, precio)
VALUES
    (1001, 'Camiseta', 10.00),
    (1002, 'Pantalón', 20.00),
    (1003, 'Zapatos',  30.00);

SELECT *
FROM   productos;


---------------------------------
-- Insertando datos en la tabla "compras"

INSERT INTO compras (id, cliente_id, producto_id, cantidad)
VALUES
    (1, 101, 1001, 2),
    (2, 101, 1003, 1),
    (3, 102, 1002, 1);

SELECT *
FROM   compras;


---------------------------------
-- Creación de vistas para concentrar información sobre las compras

CREATE VIEW compras_vw AS
    SELECT   cl.id,cl.nombre "Cliente"
            ,p.nombre "producto", p.precio
            ,cp.cantidad, p.precio * cp.cantidad "Subtotal"
    FROM    compras cp, clientes cl, productos p
    WHERE        cp.cliente_id=cl.id
            AND  cp.producto_id=p.id;

CREATE VIEW compras_cliente_vw AS
    SELECT  cliente, SUM(subtotal)
    FROM    compras_vw
    GROUP BY cliente;

SELECT *
FROM   compras_cliente_vw;
```

Lenguaje de Control de Datos (*DCL, Data Control Language*)

Por su parte el **DCL**, son las sentencias o instrucciones que permiten otorgar o revocar permisos de acceso a los usuarios, a los datos almacenados en una base de datos.

Es mediante las siguientes sentencias que, se mantiene la seguridad y control sobre quién tiene acceso a qué partes de la base de datos y qué acciones pueden llevar a cabo.

- **GRANT:** Otorga permisos a un *usuario o rol* para realizar acciones específicas en la base de datos.
- **REVOKE:** Quita permisos previamente otorgados a un *usuario o rol*.

SQL desde Python

Instalar los siguientes módulos en el ambiente `nosql` creado en *MiniConda*

- `sqlite` permite gestionar las bases de datos *sqlite3*
- `pandas` una herramienta de manipulación y análisis de datos
  - How do I read and write tabular data?
- `numpy` un módulo de alto desempeño,para el análisis de datos y cálculo nummérico
- `matplotlib` una librería especializada en la visualización de datos

In [12]:
```
#!conda install -n nosql sqlite
```

In [6]:
```
#!conda install -n nosql pandas numpy matplotlib
```

In [2]:
```
# Verifiying all libraries are ok!
try:
    import sqlite3
    import pandas as np
    import matplotlib.pyplot as plt

    print("Everything is fine, the libraries sqlite3, pandas, and matplotlib are in
except ImportError as e:
    print(f"Error: {e}")
```

```
Everything is fine, the libraries sqlite3, pandas, and matplotlib are installed.
```

In [13]:
```
import sqlite3
import pandas as pd

db  = "./RDB-Compras/compras.db"
```

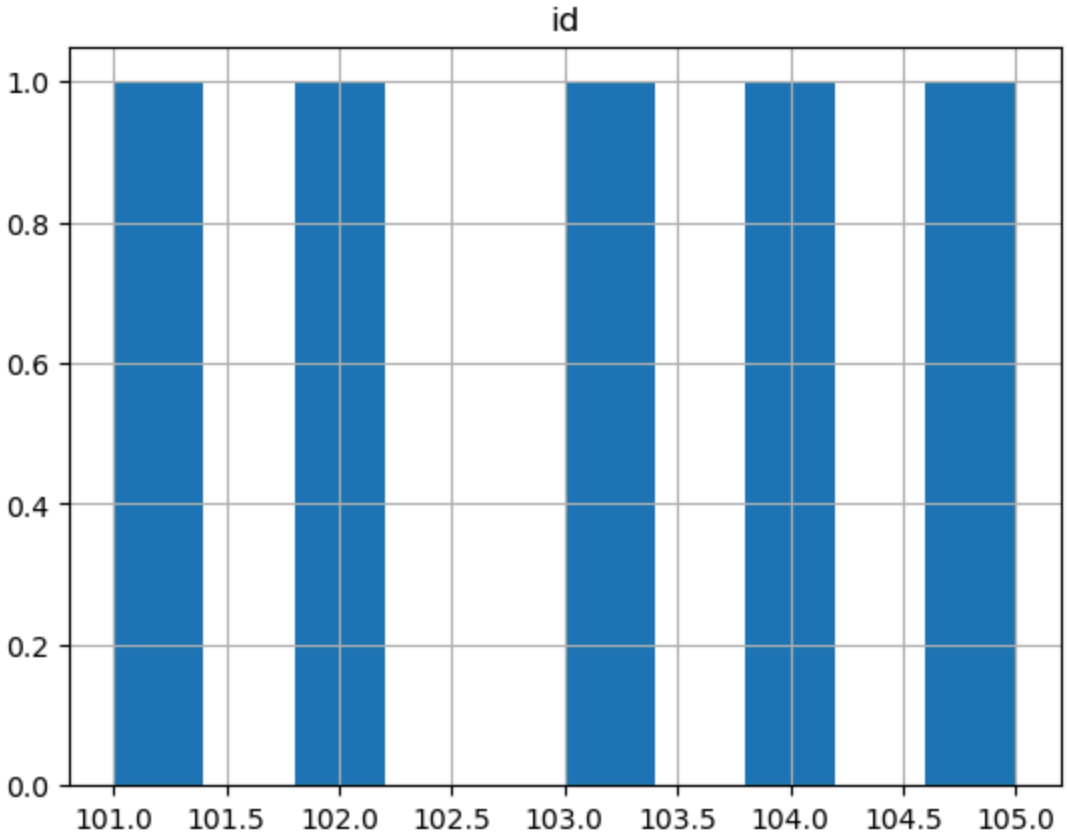
In [14]:
```
# Create a SQL connection to our SQLite3 database
con = sqlite3.connect(db)

# Create a cursor to move over the records in the database
cur = con.cursor()
```

```
# The result of a "cursor.execute" can be iterated over by row
for row in cur.execute('SELECT * FROM clientes;'):
   print(row)

# Be sure to close the connection
con.close()
```

```
(101, 'José Pérez Méndez', 'Calle A # 100')
(102, 'María López Martínez', 'Calle B # 200')
(103, 'Ana Mendoza López', 'Calle C # 103')
(104, 'Juana Carranza Aguilar', 'Calle D # 104')
(105, 'Pedro Infrante Cruz', 'Calle E # 105')
```

In [17]:
```
# Create a SQL connection to our SQLite3 database
con = sqlite3.connect(db)

tabla = "clientes"

#load data into a DataFrame object:
df = pd.read_sql_query(f"SELECT * FROM {tabla}", con)

# Be sure to close the connection
con.close()

#df.head(2)     # Requests that only the top five rows of data
#df.tail()       # Do the same for the bottom five rows of the data
#df.shape        # Dataframe size
#df.columns
#df.info()       # Provides information about both the shape and the columns of the
#df.describe()
#df.plot()
df.hist()
#df.value_counts()
#print(df['nombre'])
```

Out[17]:
```
array([[<Axes: title={'center': 'id'}>]], dtype=object)
```


id
1.0
0.8
0.6
0.4
0.2
0.0
101.0
101.5
102.0
102.5
103.0
103.5
104.0
104.5
105.0