Bases de Datos I
Unidad VIII

Lenguaje de Definición de Datos
SQL/DML

SELECT

https://www.flaticon.com/free-icons/relations

Sentencia SELECT

- **Permite realizar consultas sobre los datos almacenados sobre las distintas tablas de la base de datos.**
- **Adopta conceptos tanto de álgebra relacional, como de cálculo relacional.**
- **Como nosotros hemos utilizado álgebra relacional, veremos la sentencia SELECT en base a este último.**

Sentencia SELECT

- **Cláusulas de la sentencia SELECT:**

SELECT ← Cláusula de proyección

FROM ← Cláusula de origen/reunión

WHERE ← Cláusula de selección

GROUP BY ← Cláusula de agrupamiento

HAVING ← Cláusula de selección de grupos

ORDER BY ← Cláusula de orden

OFFSET FETCH ← Cláusula límite y paginado

Sintaxis básica INSERT

- **Orden de evaluación de las cláusulas:**

5 → SELECT
1 → FROM
2 → WHERE
3 → GROUP BY
4 → HAVING
6 → ORDER BY
7 → OFFSET FETCH

Sentencia SELECT (Proyección → π)

- **Sintaxis mínima obligatoria:**

  SELECT [ALL/DISTINCT] [*/exp1, exp2]

  FROM tabla;

- **DISTINCT permite eliminar los duplicados del resultado. Por defecto ALL.**
- **Podemos proyectar solo algunas columnas, o proyectar todas usando *.**

Ejemplo (Proyección → π)

- **Seleccionar todas las columnas de todas las facturas:**

  SELECT * **FROM** factura**;**

  – **π** numero, tipo, cliente, fecha, hora(factura)

- **Seleccionar el número y la fecha de todas las facturas:**

  SELECT numero, fecha **FROM** factura;

  – **π** numero, fecha (factura)

Ejemplo (Proyección → π)

- **Seleccionar la fecha de todas las facturas, sin duplicados:**

  SELECT DISTINCT fecha **FROM** factura;

  – **π** fecha (factura)

- **Seleccionar el número y la antigüedad de todas las facturas:**

  SELECT numero, fecha – **CURREN_DATE**
  FROM factura;

  – **π** numero, fecha – HOY (factura)

Sentencia SELECT (Selección → σ)

- **Realizamos la selección de las filas que queremos que formen parte del resultado utilizando la clausula WHERE:**

```
SELECT *
FROM tabla
WHERE condición;
```

- **La condición puede ser cualquier expresión lógica válida.**

Expresiones

- **Operadores lógicos**
- **Operadores comparación**
- **Operadores y funciones matemáticas**
- **Operadores y funciones de caracteres**
- **Operadores y de fechas y horas**
- **Expresiones condicionales**
- **Conversión entre tipos de datos**

Operadores lógicos

- **Disponemos de los operadores lógicos habituales: AND, OR y NOT**

a	b	a AND b	a OR b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
TRUE	NULL	UNKNOWN	TRUE
FALSE	FALSE	FALSE	FALSE
FALSE	NULL	FALSE	UNKNOWN
NULL	NULL	UNKNOWN	UNKNOWN

a	NOT a
TRUE	FALSE
FALSE	TRUE
NULL	UNKNOWN

Operadores de comparación

- **Para realizar comparaciones:**
  - <, >, <=, >=, =, <>
  - a [**NOT**] **BETWEEN** [**SYMMETRIC**] x **AND** y

- a **BETWEEN** x **AND** y es equivalente:
  - a >= x **AND** a <= y
- a **BETWEEN SYMMETRIC** x **AND** y es equivalente a:
  - a >= x **AND** a <= y
  - a <= x **AND** a >= y
  - Sin importar si x < y o y < x

Operadores de comparación

- **Para lidiar con nulos:**
  - **a IS [NOT] NULL**
  - **a IS [NOT] TRUE**
  - **a IS [NOT] FALSE**
  - **a IS [NOT] UNKNOWN**
  - **a IS [NOT] DISTINCT FROM b**

Operaciones y funciones matemáticas

- **Algunos operadores y funciones:**
  - **+ y - →** Suma y resta
  - *** y / →** Multiplicación y división
  - **% →** Resto de la división entera
  - **^ →** Potenciación
  - **ABS(a) →** Valor absoluto del parámetro
  - **SIGN(a) →** Signo del parámetro
  - **RANDOM()** → Número aleatorio

Operaciones y funciones de caracteres

- **Algunos operadores y funciones:**
  - **||** → Concatenación de cadenas
  - **CHARACTER_LENGTH(a)** → Tamaño
  - **LOWER(a)** → Todo a minúscula
  - **UPPER(a)** → Todo a mayúscula
  - **POSITION(q IN a)** → Posición
  - **SUBSTRING(a FROM 2 FOR 3)** → Extraer
  - **TRIM([LEADING | TRILING | BOTH] FROM a)** → Elimina espacios

Operaciones y funciones de caracteres

- **Podemos realizar comparación de cadenas de caracteres por patrones:**

  a LIKE 'patrón' [ESCAPE 'escape']

- **Retorna verdadero si la cadena de caracteres coincide con el patrón dado. El patrón puede incluir dos caracteres especiales:**
  - % Significa una secuencia de cero o más caracteres
  - _ Significa un carácter cualquiera
- **El operador SIMILAR TO permite comparar utilizando expresiones regulares.**

Operaciones y funciones de fechas y horas

- **Algunos operadores y funciones:**
  - **+ y -** → Suma y resta
  - **CURRENT_DATE** → Fecha actual
  - **CURRENT_TIME** → Hora actual
  - **CURRENT_TIMESTAMP** → Ambos
  - **AGE(h) o AGE(d, h)** → Intervalo transcurrido
  - **EXTRACT(q FROM f)** → Extrae parte

  q puede ser: **CENTURY, YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, etc**

Expresiones condicionales

- **Algunas expresiones útiles:**
  - **CASE WHEN** condicion **THEN** resultado

    [WHEN …]

    [ELSE resultado**]**

    END → Retorna el primer resultado donde se cumple la condición, o **ELSE**
  - **COALESCE(a1 [, …]) →** Retorna el 1 no nulo
  - **GREATEST(a1 [, …]) →** Retorna el mayor
  - **LEAST(a1 [, …]) →** Retorna el menor

Conversión de tipos

- **Podemos realiza conversiones entre distintos tipos de datos, siempre que la misma sea posible:**

  CAST(a **AS** tipo**)**

- **Por ejemplo podemos convertir un numero a cadena de caracteres:**

  CAST(1 **AS VARCHAR**(50)**)**

- **O viceversa:**

  CAST('1' **AS INTEGER)**

Sentencia SELECT (Renombrar → ρ)

- **Podemos renombrar tanto columnas como tablas utilizando el operador AS:**

  SELECT campo **AS** nombre

  FROM tabla **AS** otra**;**

- **Puede ser utilizado cuando la consulta involucre varias veces la misma tabla o para dar nombre a columnas generadas a partir de expresiones.**

Ejemplo (Renombrar → ρ)

- **Obtenemos el documento y nacimiento de los clientes renombrados:**

SELECT documento **AS** dni**,**
nacimiento **AS** “Fecha de nacimiento”
FROM cliente;

  - ρ dni←documento, fecha←nacimiento (π documento, nacimiento(cliente))

Sentencia SELECT (Ordenamiento)

- **Podemos ordenar las filas del resultado haciendo uso de la cláusula ORDER BY:**

  SELECT * **FROM** tabla

  - **ORDER BY** campo1 **[ASC/DESC] [NULLS LAST/NULLS FIRST];**

- **El orden puede ser tanto ascendente como descendente (por defecto ASC) e incluir varios campos por su nombre o por su posición en el resultado.**

Ejemplo (Ordenamiento)

- **Obtenemos el apellido, nombre y nacimiento de los clientes ordenados por apellido de forma descendente y nombre de forma ascendente:**

```
SELECT apellido, nombre, nacimiento
FROM cliente
ORDER BY apellido DESC, nombre;
```

Limitando y paginando resultados

- **En ocasiones no queremos obtener todas las filas, sino sólo las 10 primeras o las 10 siguientes. Esto podemos realizarlo utilizando:**

  OFFSET [inicio] **[ROW/ROWS]**

  FETCH [FIRST/NEXT] [cantidad] **[ROW/ROWS] ONLY**

- **Con OFFSET establecemos el desplazamiento y con FETCH la cantidad.**

Ejemplo (Limitando)

- **Obtenemos el apellido, nombre y nacimiento de los 10 clientes más longevos que tenemos:**

```
SELECT apellido, nombre, nacimiento
FROM cliente
ORDER BY nacimiento DESC
FETCH FIRST 10 ROWS ONLY;
```

- **Omitir OFFSET implica OFFSET 0.**

Ejemplo (Paginando)

- **Obtenemos los 10 siguientes:**

```
SELECT apellido, nombre, nacimiento
FROM cliente
ORDER BY nacimiento DESC
OFFSET 10
FETCH FIRST 10 ROWS ONLY;
```

- **Omitir la cantidad en FETCH, implica FETCH 1.**

Operadores SQL y álgebra

- **Existe también en SQL los operadores de conjunto del álgebra relacional:**
  - UNION → ∪
  - INTERSECT → ∩
  - EXCEPT → **-**

Sentencia SELECT (Unión → ∪)

- **Podemos realizar la unión de resultados haciendo uso de UNION:**

```
SELECT * FROM tabla1
UNION [DISTINCT/ALL]
SELECT * FROM tabla2;
```

- **Por defecto elimina los duplicados (DISTINCT), pero podemos incluirlos si lo deseamos (ALL).**
- **Los resultados deben ser unión compatibles.**

Ejemplo (Unión → ∪)

- **Obtenemos todos los clientes que nacieron el 09/12/1978 y todos los que nacieron el 20/05/1977:**

```
SELECT * FROM cliente
WHERE nacimiento = '1978-12-09'
UNION
SELECT * FROM cliente
WHERE nacimiento = '1977-05-20'
```

- π *(σ nacimiento=’1978-12-09’(cliente)) $\cup$

  π *(σ nacimiento=’1977-05-20’(cliente))

Sentencia SELECT (Intersección→ ∩)

- **Podemos realizar la intersección de resultados haciendo uso de INTERSECT:**

```
SELECT * FROM tabla1
INTERSECT [DISTINCT/ALL]
SELECT * FROM tabla2;
```

- **Por defecto elimina los duplicados (DISTINCT), pero podemos incluirlos si lo deseamos (ALL).**
- **Los resultados deben ser unión compatibles.**

Ejemplo (Intersección→ ∩)

- **Obtenemos todos los clientes que tiene apellido 'Perez' y también nombre 'Carlos':**

```
SELECT * FROM cliente
WHERE apellido = 'Perez'
INTERSECT
SELECT * FROM cliente
WHERE nombre = 'Carlos'
```

- π *(σ apellido=’Perez’(cliente)) $\cap$

  π *(σ nombre=’Carlos’(cliente))

Sentencia SELECT (Diferencia → -)

- **Podemos realizar la diferencia de resultados haciendo uso de EXCEPT:**

```
SELECT * FROM tabla1
EXCEPT [DISTINCT/ALL]
SELECT * FROM tabla2;
```

- **Por defecto elimina los duplicados (DISTINCT), pero podemos incluirlos si lo deseamos (ALL).**
- **Los resultados deben ser unión compatibles.**

Ejemplo (Diferencia → -)

- **Obtenemos todos los clientes que tiene apellido 'Perez' y no tienen nombre 'Carlos':**

```
SELECT * FROM cliente
WHERE apellido = 'Perez'
EXCEPT
SELECT * FROM cliente
WHERE nombre = 'Carlos'
```

- π *(σ apellido=’Perez’(cliente)) -

  π *(σ nombre=’Carlos’(cliente))

Sentencia SELECT (Reunión)

- **Distintos tipos de reunión:**
  - Producto cartesiano → ×
  - Reunión interna (natural) → ⋈
  - **Reunión externa**
    - Reunión externa izquierda → ⟕
    - Reunión externa derecha → ⟖
    - Reunión externa completa → ⟗

Producto cartesiano → ×

- **Para realizar el producto cartesiano de dos tablas, podemos utilizar dos formas:**

```
SELECT * FROM tabla1, tabla2;
```

```
SELECT * FROM tabla1 CROSS JOIN tabla2;
```

  - tabla1 × tabla2

Ejemplo (Producto cartesiano → ×)

- **Obtenemos el productor cartesiano de clientes y facturas:**

  SELECT * **FROM** cliente, factura;

  SELECT * **FROM** cliente **CROSS JOIN** factura;

  - cliente × factura

Ejemplo (Producto cartesiano → ×)

- **Cabe recordar que la reunión natural en álgebra relacional, puede ser realizada combinando el producto cartesiano y la selección:**

```
SELECT f.*, c.apellido
FROM factura AS f, cliente AS c
WHERE f.cliente= c.documento;
```

  - π f.*, c.apellido(σ f.cliente=c.documento(ρ f(factura) × ρ c(cliente)))

Reuniones

- **Para realizar reuniones, existen distintas variantes y modificadores:**
- **Variante 1:**

  tabla1 [tipo] **JOIN** tabla2 **ON(**comparaciones**)**
- **Variante 2:**
  - tabla1 [tipo] **JOIN** tabla2 **USING**(cols)
- **Variante 3:**
  - tabla1 **NATURAL** [tipo] **JOIN** tabla2
- **Donde tipo puede ser INNER, LEFT OUTER, RIGHT OUTER o FULL OUTER.**

Reuniones (Variante 1)

- **En esta variante realizamos la reunión entre las filas de las tablas en las que se cumple la condición de la cláusula ON:**

```
SELECT *
FROM tabla1 [tipo] JOIN tabla2 ON (tabla1.a = tabla2.a);
```

Reuniones (Variante 2)

- **En esta variante realizamos la reunión entre las filas de las tablas a través de la coincidencia de las columnas indicadas (mismo nombre en ambas tablas) en la cláusula USING:**

```
SELECT *
FROM tabla1 [tipo] JOIN tabla2
USING(columna1, columna2);
```

Reuniones (Variante 3)

- **En esta variante realizamos la reunión entre las filas de las tablas a través de la coincidencia de las columnas que tengan el mismo nombre en ambas tablas:**

```
SELECT *
FROM tabla1 NATURAL [tipo] JOIN tabla2;
```

Reunión interna → ⋈

- **Podemos realizar la reunión natural de dos tablas, utilizando INNER JOIN:**

  SELECT *

  FROM tabla1

  [NATURAL] INNER JOIN tabla2

  [ON, USING]

- **Podemos realizar la reunión interna utilizando NATURAL, ON o USING.**

Ejemplo (Reunión interna → ⋈)

- **Obtenemos los datos de las facturas con el apellido del cliente:**

  SELECT factura.*, cliente.apellido
  FROM factura **INNER JOIN** cliente **ON**
  (factura.cliente=cliente.documento)

  - π factura.*, cliente.apellido(factura ⋈ factura.cliente=cliente.documento cliente)

- **Las facturas que no son a nombre de ningún cliente no las obtenemos.**

Reunión externa izquierda → ⟕

- **Podemos realizar la reunión externa izquierda de dos tablas, utilizando LEFT OUTER JOIN:**

  SELECT *

  FROM tabla1

  [NATURAL] LEFT OUTER JOIN tabla2

  [ON, USING]

- **Podemos realizar la reunión externa izquierda utilizando NATURAL, ON o USING.**

Ejemplo: Reunión externa izquierda → ⟕

- **Obtenemos los datos de las facturas con los datos de los clientes:**

  SELECT *
  FROM factura **LEFT JOIN** cliente **ON**
  (factura.cliente = cliente.documento)

  - π *(factura ⟕ factura.cliente=cliente.documento cliente)

- **Las facturas que no son a nombre de ningún cliente las obtenemos igual, con los datos de cliente nulos.**

Reunión externa derecha → ⟖

- **Podemos realizar la reunión externa derecha de dos tablas, utilizando RIGHT OUTER JOIN:**

  SELECT *

  FROM tabla1

  [NATURAL] RIGHT OUTER JOIN tabla2

  [ON, USING]

- **Podemos realizar la reunión natural utilizando NATURAL, ON o USING.**

Ejemplo: Reunión externa derecha → ⟖

- **Obtenemos los datos de las facturas con los datos de los clientes:**

  SELECT *
  FROM factura **RIGHT JOIN** cliente **ON**
  (factura.cliente = cliente.documento)

  - π *(factura ⟖ factura.cliente=cliente.documento cliente)

- **Los clientes a los que no se le realizó ninguna factura los obtenemos igual con los datos de la factura nulos.**

Reunión externa completa → ⟗

- **Podemos realizar la reunión externa completa de dos tablas, utilizando INNER JOIN:**

  SELECT *

  FROM tabla1

  [NATURAL] FULL OUTER JOIN tabla2

  [ON, USING]

- **Podemos realizar la reunión natural utilizando NATURAL, ON o USING.**

Ejemplo: Reunión externa completa → ⟗

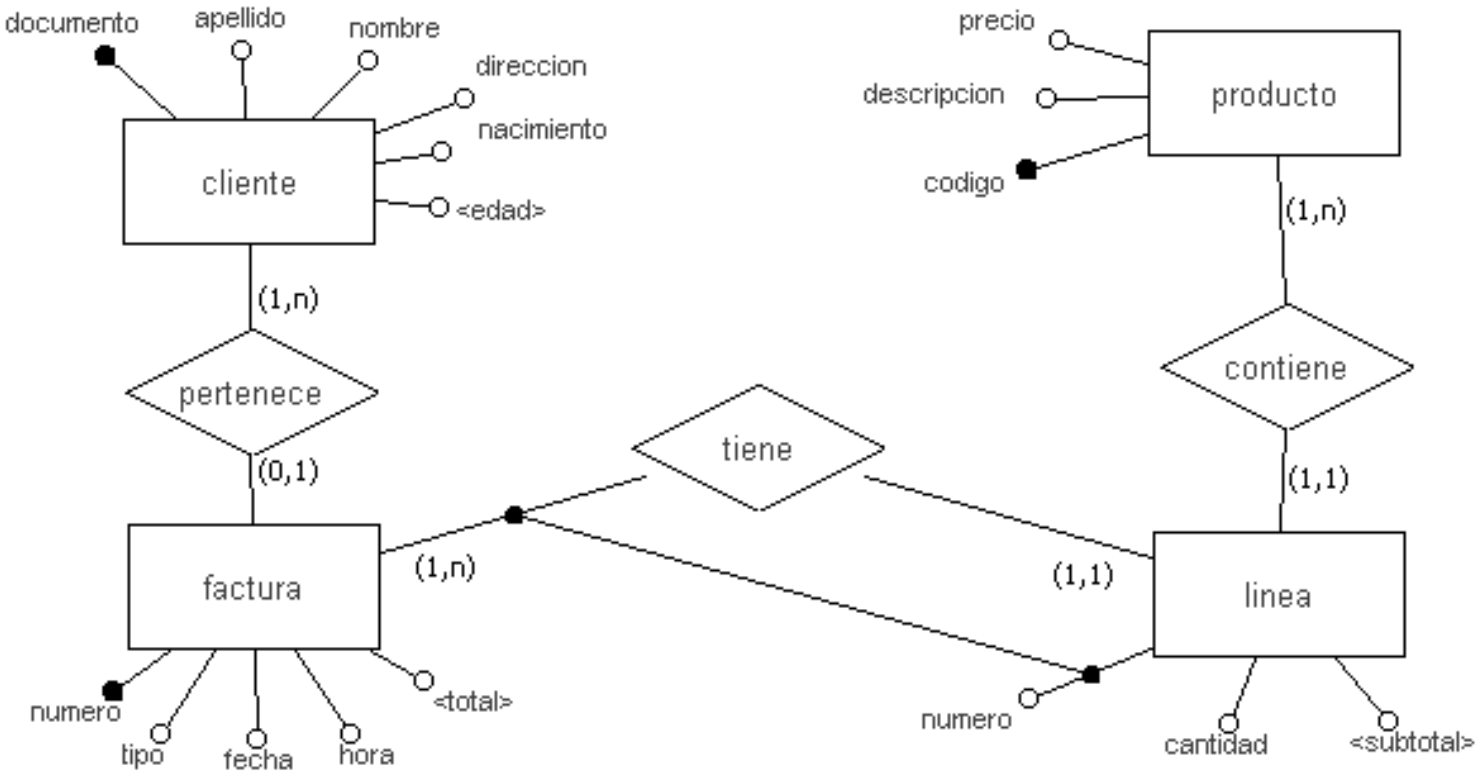
- **Obtenemos los datos de las facturas con los datos de los clientes:**

  SELECT *
  FROM factura **FULL JOIN** cliente **ON**
  (factura.cliente = cliente.documento)

  - π *(factura ⟗ factura.cliente=cliente.documento cliente)

- **Obtenemos incluso las facturas sin cliente y los clientes sin facturas.**

Diseño conceptual para los ejemplos



Base de datos relacional ejemplos

- **Tabla CLIENTE:**

```
CREATE TABLE cliente(
  documento INTEGER PRIMARY KEY,
  apellido      VARCHAR(50) NOT NULL,
  nombre        VARCHAR(50) NOT NULL,
  nacimiento DATE NOT NULL,
  direccion     VARCHAR(100) NOT NULL
);
```

Base de datos relacional ejemplos

- **Tabla PRODUCTO:**

```
CREATE TABLE producto(
  codigo       BIGINT PRIMARY KEY,
  descripcion  VARCHAR(50) NOT NULL,
  precio       NUMERIC(12, 2) NOT NUL DEFAULT 0
);
```

Base de datos relacional ejemplos

- **Tabla FACTURA:**

```
CREATE TABLE factura(
  numero INTEGER PRIMARY KEY,
  tipo       CHAR(1) NOT NULL,
  cliente  INTEGER REFERENCES cliente,
  fecha    DATE NOT NULL,
  hora     TIME NOT NULL
);
```

Base de datos relacional ejemplos

- **Tabla LINEA:**

```
CREATE TABLE linea(
  factura INTEGER NOT NULL REFERENCES factura,
  numero INTEGER NOT NULL,
  producto BIGINT NOT NULL REFERENCES producto,
  cantidad INTEGER NOT NULL,
  PRIMARY KEY(factura, numero)
);
```

Datos de ejemplo tabla cliente

INSERT INTO cliente VALUES

(11222333, 'Perez', 'Pedro', '1977-01-01', 'Los Cauquenes 123'),

(22333444, 'Gonzalez', 'Juana', '1978-01-01', 'Marcos Zar 234'),

(33444555, 'Alvarez', 'Mirta', '1979-01-01', 'Maria Sanchez de Caballero 345'),

(44555666, 'Fernandez', 'Juliana', '1980-01-01', 'Lucas Bridges 456'),

(55666777, 'Argento', 'Mauro', '1977-06-06', 'Marcos Vera 567'),

(66777888, 'Muñoz', 'Juan', '1978-06-06', 'Concejal Rubinos 678'),

(77888999, 'Estevez', 'Mario', '1979-06-06', 'La Pataia 789'),

(88999000, 'Perez', 'Maria', '1980-06-06', 'Los Cauquenes 123');

Datos de ejemplo tabla producto

INSERT INTO producto VALUES

(7790072001037, 'Limpiador de pisos', 20.19),
(7790072001038, 'Lavandina', 25.59),
(7790072001039, 'Cera para pisos', 59.99),
(7790172001019, 'Desodorante en crema', 12),
(7790172001029, 'Tintura para cabello', 61.2),
(7790172001049, 'Crema dental', 11.19),
(7790072002011, 'Galletitas dulces', 10),
(7790072002022, 'Pan dulce con frutas', 60.99),
(7790072002033, 'Turrón de maní', 8.99),
(7790071001000, 'Leche entera', 10),
(7790073001000, 'Yogurt bebible de frutilla', 14.59),
(7790074001000, 'Crema de leche entera', 15);

Datos de ejemplo tabla factura

INSERT INTO factura VALUES

(1, 'A', 11222333, '2012-01-01', CURRENT_TIME), (2, 'B', 22333444, '2012-01-05', CURRENT_TIME),
(3, 'A', 33444555, '2012-02-01', CURRENT_TIME), (4, 'B', 44555666, '2012-02-05', CURRENT_TIME),
(5, 'A', 55666777, '2012-03-01', CURRENT_TIME), (6, 'B', 66777888, '2012-03-05', CURRENT_TIME),
(7, 'A', 77888999, '2012-04-01', CURRENT_TIME), (8, 'B', 88999000, '2012-04-05', CURRENT_TIME),
(9, 'A', 11222333, '2012-05-01', CURRENT_TIME), (10, 'B', 22333444, '2012-05-05', CURRENT_TIME),
(11, 'A', 33444555, '2012-06-01', CURRENT_TIME),(12, 'B', 11222333, '2012-06-05', CURRENT_TIME),
(13, 'A', 22333444, '2012-07-01', CURRENT_TIME),
(14, 'B', 44555666, '2012-07-05', CURRENT_TIME),
(15, 'A', 55666777, '2012-08-01', CURRENT_TIME),
(16, 'B', 77888999, '2012-08-05', CURRENT_TIME),
(17, 'A', 88999000, '2012-09-01', CURRENT_TIME),
(18, 'B', 22333444, '2012-09-05', CURRENT_TIME),
(19, 'A', 33444555, '2012-10-01', CURRENT_TIME),
(20, 'B', 44555666, '2012-10-05', CURRENT_TIME);

Datos de ejemplo tabla linea

INSERT INTO linea VALUES

(1, 1, 7790072001037, 2), (1, 2, 7790172001019, 4),
(2, 1, 7790072001037, 5), (2, 2, 7790172001049, 6),
(3, 1, 7790172001029, 1), (3, 2, 7790172001019, 7),
(4, 1, 7790172001029, 10), (4, 2, 7790072001037, 12),
(5, 1, 7790071001000, 13), (6, 1, 7790072002033, 90),
(7, 1, 7790072002033, 2), (8, 1, 7790074001000, 3),
(9, 1, 7790073001000, 12), (10, 1, 7790072001037, 78),
(11, 1, 7790074001000, 24), (11, 2, 7790172001029, 56),
(12, 1, 7790172001049, 12), (13, 1, 7790072002022, 12),
(13, 2, 7790073001000, 5), (14, 1, 7790072002011, 4),
(15, 1, 7790172001019, 6), (15, 2, 7790172001029, 8),
(16, 1, 7790172001049, 12), (16, 2, 7790071001000, 77),
(16, 3, 7790072002022, 11), (17, 1, 7790072002011, 55),
(17, 2, 7790172001019, 22), (18, 1, 7790172001019, 23),
(19, 1, 7790072001038, 1), (20, 1, 7790172001049, 56),
(20, 2, 7790072001038, 34);

Bibliografía

- **SQL-99 Complete, Really. 1999. Peter Gulutzan y Trudy Pelzer.**
- **PostgreSQL Introduction and Concepts. 2001. Momjian, Bruce.**
- **PostgreSQL 9.6 Documentation. 2016. The PostgreSQL Global Development Group.**
- **A Simpler (and Better) SQL Approach to Relational Division. 1998. Victor M. Matos y Rebecca Grasser.**