BASE DE DATOS PARA INTELIGENCIA ARTIFICIAL

- 04/11/2023

Esquema de hoy

- Repaso
- Normalización
- Data Languages
- SQL Basics
- SQL Advanced
- Postgres (Chinook)
- Prácticas

REPASO

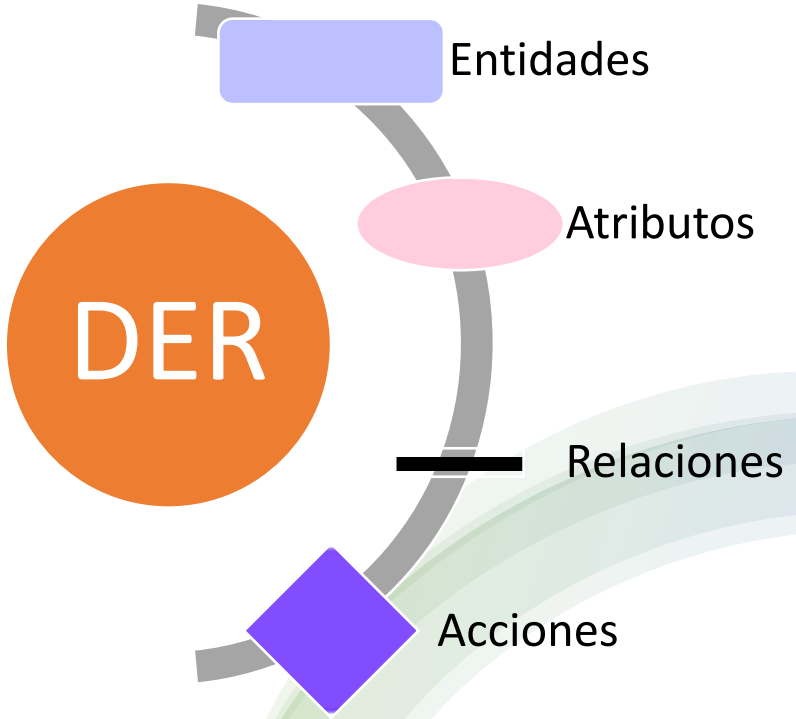
Conceptos básicos

Diagrama Entidad Relación (DER/ERD)

El modelo Entidad-Relación (E-R) es una manera de representar nuestra percepción del sistema que vamos a modelar.

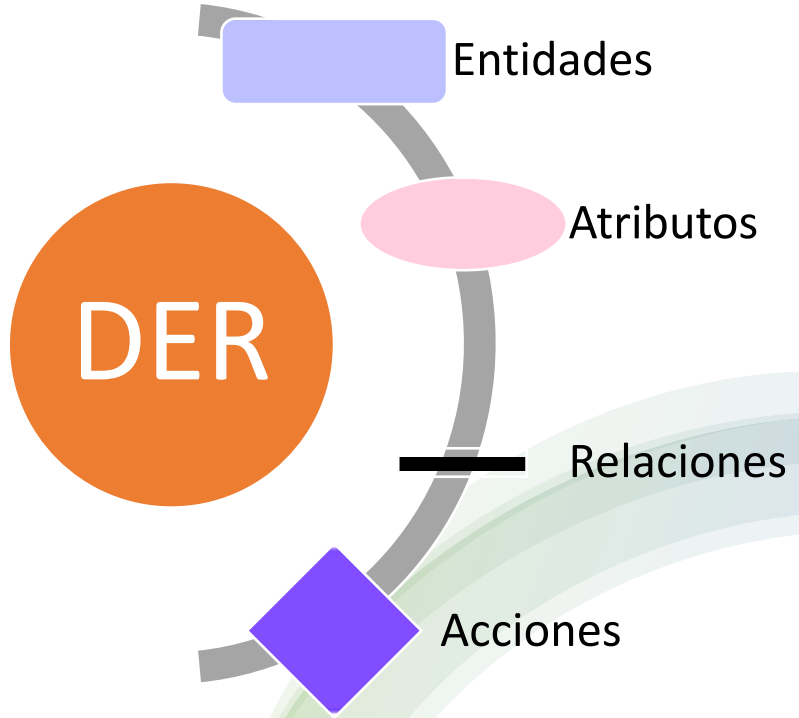
Este consiste en un conjunto de objetos básicos:

- **Entidades**
- **Atributos**
- **Interrelaciones**
- **Acciones**



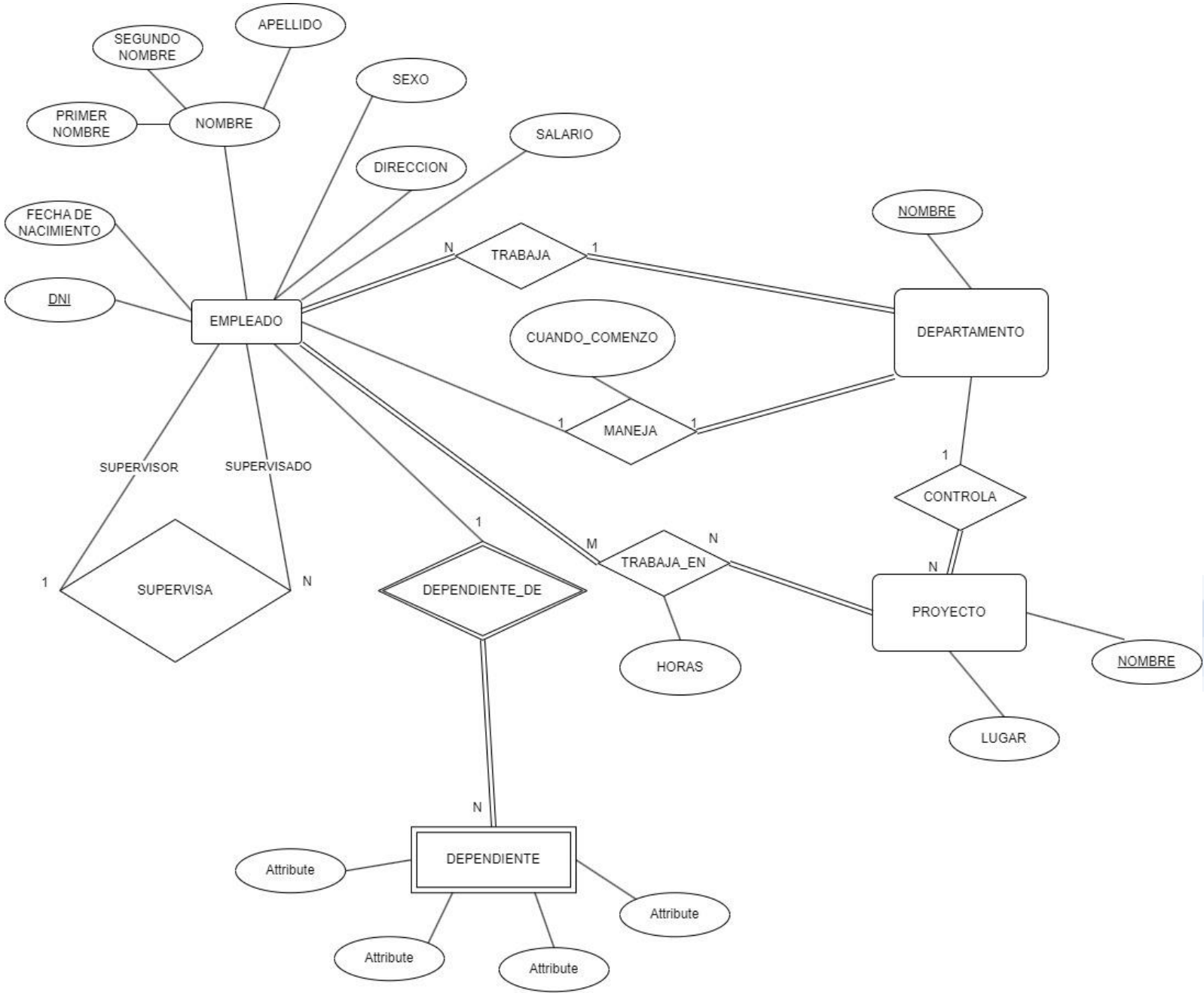
Repaso

- Atributos
  - Simples
  - compuestos
- Interrelaciones
  - **Grado**
    - **Unaria**
    - **Binaria**
  - **Cardinalidad**
    - **1:1**
    - **1:N**
    - **N:M**
  - **Participación**
    - **Total**
    - **Parcial**
- **Entidades**
  - **Fuertes o Debiles**



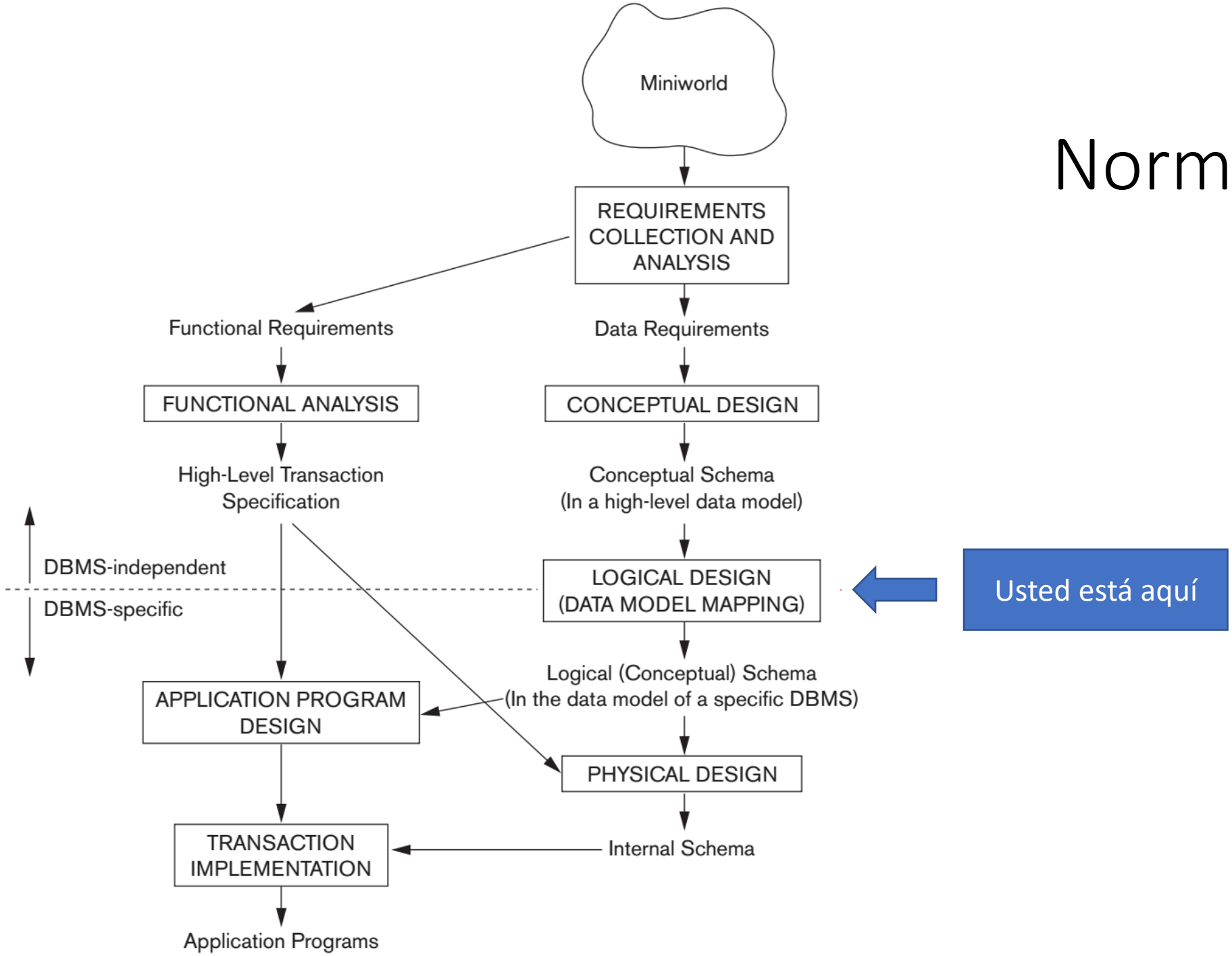
Repaso

- DER
- MER



NORMALIZACION

Normalización



Guía informal de diseño de esquemas

- Asegurarnos que la semántica de los atributos sea la correcta
- Reducir información redundante en las tablas
- Reducir la cantidad de valores NULL en las tuplas
- No permitir la posibilidad de generar tuplas espurias o anomalías
  - Anomalías por Inserción
  - Anomalías por Borrado
  - Anomalías por Modificación

SALES
ITEMS
PRICE
CREATED_AT
QUANTITY
FULL_NAME
ADDRESS
…

Normalización

- Proceso de aplicar una serie de reglas para minimizar (o eliminar) la redundancia de datos
- Evita anomalías sobre los datos
- Separa grandes tablas en tablas mas pequeñas y las une a través de relaciones

Normalización

SALES
ITEMS
PRICE
CREATED_AT
QUANTITY
FULL_NAME
ADDRESS
…

ORDERS
ORDER_ITEMS_ID
CUSTOMER_ID
CREATED_AT
QUANTITY
…

CUSTOMER
EMAIL
ID
FULL_NAME
ADDRESS
…

ORDER ITEMS
ITEM
PRICE
CATEGORY
DESCRIPTION
ID
…

Normalización

1 FN

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

name	address	phone
Thomas Monero	12 Hangover Sq.	3456
Jessica Moos	12 Hangover Sq.	3456
Adam Hardy	Cerrito 333	1243
Maria Anders	Av. del Libertador 900	8906

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

Primary key

customer_id	name	address	phone
101	Thomas Monero	12 Hangover Sq.	3456
102	Jessica Moos	12 Hangover Sq.	3456
103	Adam Hardy	Cerrito 333	1243
104	Maria Anders	Av. del Libertador 900	8906

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

customer_id	name	address	phone
101	Thomas Monero	12 Hangover Sq.	3456
102	Jessica Moos	12 Hangover Sq.	3456
103	Adam Hardy	Cerrito 333	1243
104	Maria Anders	Av. del Libertador 900	8906

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

customer_id	first_name	last_name	address	phone
101	Thomas	Monero	12 Hangover Sq.	3456
102	Jessica	Moos	12 Hangover Sq.	3456
103	Adam	Hardy	Cerrito 333	1243
104	Maria	Anders	Av. del Libertador 900	8906

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

order_detail

order_id	product_id	price	quantity	new_product	new_product_price	new_product_quantity
101	1	45.6	30	2	18	12
102	3	12	61	4	43.9	10
103	3	10	47	1	18	22

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

1FN

order_detail

order_id	product_id	price	quantity
101	1	45.6	30
101	2	18	12
102	3	12	61
102	4	43.9	10
103	3	10	47
103	1	18	22

- PRIMARY KEY
- CADA ATRIBUTO DEBE CONTENER UN SOLO VALOR ATÓMICO
- SIN GRUPOS REPETIDOS.

Normalización

2 FN

- CUMPLE CON 1 FN
- SIN DATOS REDUNDANTES
- LOS ATRIBUTOS QUE NO FORMAN PARTE DE LA CLAVE ESTAN COMPLETAMENTE DEFINIDOS POR LA CLAVE

2FN

order_id	product_id	price	quantity	order_date	shipping_detail
101	1	45.6	30	2019-08-16	31 Hill Haven Drive
101	2	18	12	2019-08-16	31 Hill Haven Drive
102	3	12	61	2020-10-10	12 Breezewood Court
102	4	43.9	10	2020-10-10	12 Breezewood Court
103	3	10	47	2018-03-17	100 West Street
103	1	18	22	2018-03-17	100 West Street

- CUMPLE CON 1 FN
- SIN DATOS REDUNDANTES
- LOS ATRIBUTOS QUE NO FORMAN PARTE DE LA CLAVE ESTAN COMPLETAMENTE DEFINIDOS POR LA CLAVE

2FN

DEPENDENCIA PARCIAL

order_id	product_id	price	quantity	order_date	shipping_detail
101	1	45.6	30	2019-08-16	31 Hill Haven Drive
101	2	18	12	2019-08-16	31 Hill Haven Drive
102	3	12	61	2020-10-10	12 Breezewood Court
102	4	43.9	10	2020-10-10	12 Breezewood Court
103	3	10	47	2018-03-17	100 West Street
103	1	18	22	2018-03-17	100 West Street

PRIMARY KEY COMPUESTA

NON KEY COLUMNS

- CUMPLE CON 1 FN
- SIN DATOS REDUNDANTES
- LOS ATRIBUTOS QUE NO FORMAN PARTE DE LA CLAVE ESTAN COMPLETAMENTE DEFINIDOS POR LA CLAVE

2FN

order_id	product_id	price	quantity
101	1	45.6	30
101	2	18	12
102	3	12	61
102	4	43.9	10
103	3	10	47
103	1	18	22

order_id	order_date	shipping_detail
101	2019-08-16	31 Hill Haven Drive
102	2020-10-10	12 Breezewood Court
103	2018-03-17	100 West Street

- CUMPLE CON 1 FN
- SIN DATOS REDUNDANTES
- LOS ATRIBUTOS QUE NO FORMAN PARTE DE LA CLAVE ESTAN COMPLETAMENTE DEFINIDOS POR LA CLAVE

Normalización

3 FN

- CUMPLE CON 2 FN
- SIN DATOS REDUNDANTES
- NO TENER DEPENDENCIAS TRANSITIVAS (NO INGESTEN COSAS DE OTRAS TABLAS QUE NO SEAN PRIMARY KEYS)

3FN

ProductID	ProductName	Supplier ID	Supplier Name	Supplier Address	UnitPrice
1	Rssle Sauerkraut	3	Plutzer	3200 ParkWay	45.6
2	Chartreuse verte	3	Plutzer	3200 ParkWay	18
3	Spegesild	10	Lyngbysild	131 Stadium Drive	12
4	Vegie-spread	10	Lyngbysild	131 Stadium Drive	43.9
5	Chocolate Bar	43	Cedar Choc	20 Barfield Lane	10
6	Choco Milk	43	Cedar Choc	20 Barfield Lane	18

- CUMPLE CON 2 FN
- SIN DATOS REDUNDANTES
- NO TENER DEPENDENCIAS TRANSITIVAS (NO INGESTEN COSAS DE OTRAS TABLAS QUE NO SEAN PRIMARY KEYS)

3FN

DEPENDENCIA TRANSITIVA

DEPENDENCIA

ProductID	ProductName	Supplier ID	Supplier Name	Supplier Address	UnitPrice
1	Rssle Sauerkraut	3	Plutzer	3200 ParkWay	45.6
2	Chartreuse verte	3	Plutzer	3200 ParkWay	18
3	Spegesild	10	Lyngbysild	131 Stadium Drive	12
4	Vegie-spread	10	Lyngbysild	131 Stadium Drive	43.9
5	Chocolate Bar	43	Cedar Choc	20 Barfield Lane	10
6	Choco Milk	43	Cedar Choc	20 Barfield Lane	18

- CUMPLE CON 2 FN
- SIN DATOS REDUNDANTES
- NO TENER DEPENDENCIAS TRANSITIVAS (NO INGESTEN COSAS DE OTRAS TABLAS QUE NO SEAN PRIMARY KEYS)

3FN

ProductID	ProductName	Supplier ID	Supplier Name	Supplier Address	UnitPrice
1	Rssle Sauerkraut	3	Plutzer	3200 ParkWay	45.6
2	Chartreuse verte	3	Plutzer	3200 ParkWay	18
3	Spegesild	10	Lyngbysild	131 Stadium Drive	12
4	Vegie-spread	10	Lyngbysild	131 Stadium Drive	43.9
5	Chocolate Bar	43	Cedar Choc	20 Barfield Lane	10
6	Choco Milk	43	Cedar Choc	20 Barfield Lane	18

- CUMPLE CON 2 FN
- SIN DATOS REDUNDANTES
- NO TENER DEPENDENCIAS TRANSITIVAS (NO INGESTEN COSAS DE OTRAS TABLAS QUE NO SEAN PRIMARY KEYS)

3FN

ProductID	ProductName	Supplier ID	UnitPrice
1	Rssle Sauerkraut	3	45.6
2	Chartreuse verte	3	18
3	Spegesild	10	12
4	Vegie-spread	10	43.9
5	Chocolate Bar	43	10
6	Choco Milk	43	18

Supplier ID	Supplier Name	Supplier Address
3	Plutzer	3200 ParkWay
10	Lyngbysild	131 Stadium Drive
43	Cedar Choc	20 Barfield Lane

- CUMPLE CON 2 FN
- SIN DATOS REDUNDANTES
- NO TENER DEPENDENCIAS TRANSITIVAS (NO INGESTEN COSAS DE OTRAS TABLAS QUE NO SEAN PRIMARY KEYS)

Lenguaje SQL

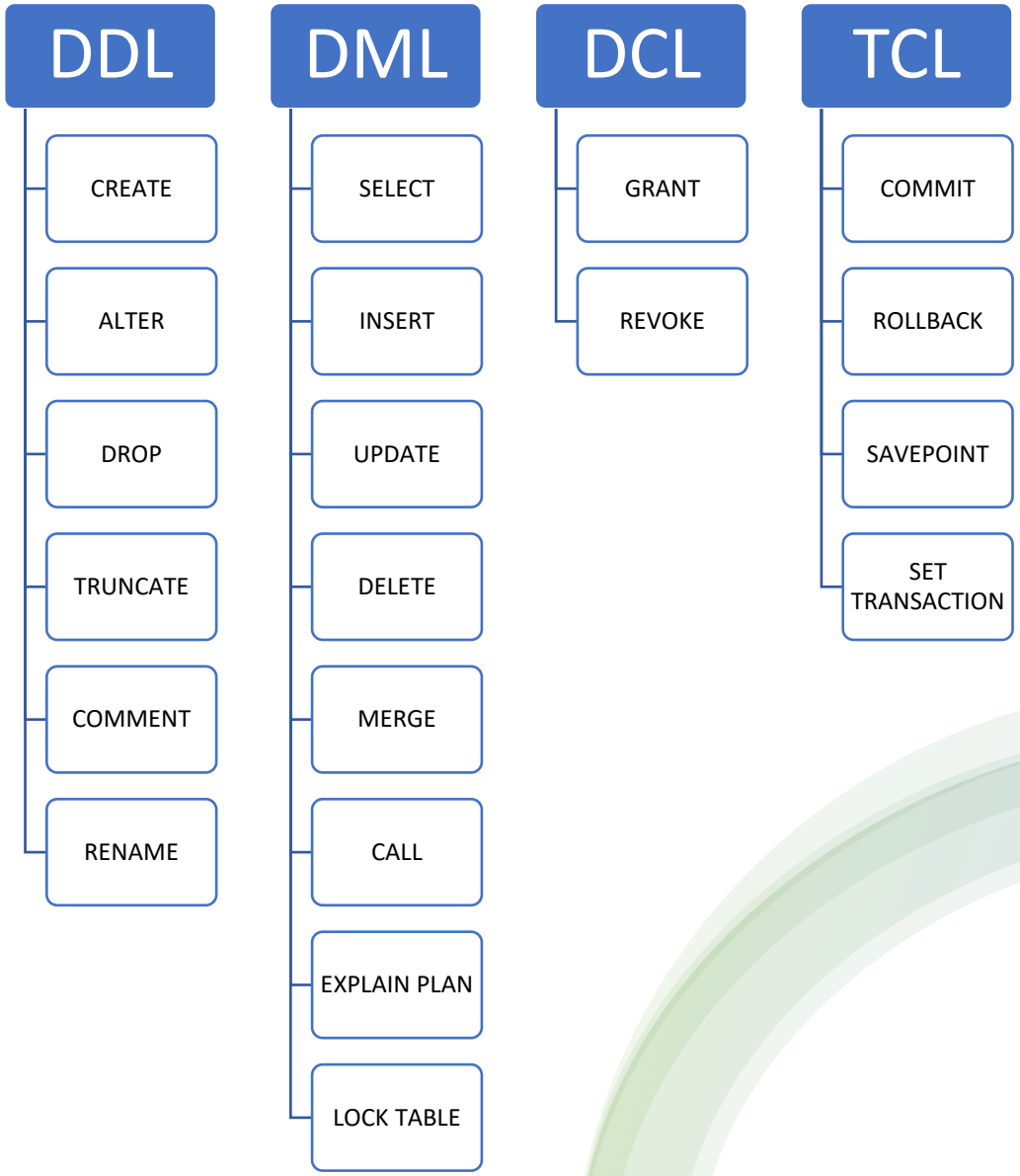
Data * Language

Tipos de SQL

En SQL podemos encontrar 4 categorías importantes:

- **DDL**:Corresponde a los comandos que nos permiten definirla estructura de nuestro MR.

- **DML**: Estos comandos nos permiten manipular nuestro MR



Tipos de SQL

En SQL podemos encontrar 4 categorías importantes:

- **DCL:** Estos comandos rigen la parte de control y gobierno de nuestros datos

- **TCL:** Estos comandos rigen el control sobre las operaciones de DML.

Nosotros nos vamos a concentrar sobre todo en **DML** y un poco de **DDL**

DDL

Data Definition Language

Creación de Elementos

- En PostgreSQL, tenemos que la sentencia **CREATE** nos permite crear distintos elementos en nuestra base de datos. Estos pueden ser:
  - **DATABASE**
  - **TABLE**
  - **INDEX**
  - **ROLE**
  - **USER**
  - …

- `CREATE DATABASE testDB;`

- ```
  CREATE TABLE Persons(
      PersonID int,
      LastName varchar(255),
      FirstName varchar(255),
      Address varchar(255),
      City varchar(255)
  );
  ```

- ```
  CREATE TABLE TestTable AS
      SELECT
          customername,
          contactname
      FROM
          customers;
  ```

- `CREATE USER name [ [ WITH ] option[ ... ] ];`

Name	Aliases	Description
bigint	int8	signed eight-byte integer
bigserial	serial8	autoincrementing eight-byte integer
bit [(*n*)]		fixed-length bit string
bit varying [(*n*)]	varbit [(*n*)]	variable-length bit string
boolean	bool	logical Boolean (true/false)
box		rectangular box on a plane
bytea		binary data ("byte array")
character [(*n*)]	char [(*n*)]	fixed-length character string
character varying [(*n*)]	varchar [(*n*)]	variable-length character string
cidr		IPv4 or IPv6 network address
circle		circle on a plane
date		calendar date (year, month, day)
double precision	float8	double precision floating-point number (8 bytes)
inet		IPv4 or IPv6 host address
integer	int, int4	signed four-byte integer
interval [*fields*] [(*p*)]		time span
json		textual JSON data
jsonb		binary JSON data, decomposed
line		infinite line on a plane
lseg		line segment on a plane
macaddr		MAC (Media Access Control) address
macaddr8		MAC (Media Access Control) address (EUI-64 format)
money		currency amount
numeric [(*p*, *s*)]	decimal [(*p*, *s*)]	exact numeric of selectable precision
path		geometric path on a plane
pg_lsn		PostgreSQL Log Sequence Number
pg_snapshot		user-level transaction ID snapshot
point		geometric point on a plane
polygon		closed geometric path on a plane
real	float4	single precision floating-point number (4 bytes)
smallint	int2	signed two-byte integer
smallserial	serial2	autoincrementing two-byte integer
serial	serial4	autoincrementing four-byte integer
text		variable-length character string
time [(*p*)] [without time zone]		time of day (no time zone)
time [(*p*)] with time zone	timetz	time of day, including time zone
timestamp [(*p*)] [without time zone]		date and time (no time zone)
timestamp [(*p*)] with time zone	timestamptz	date and time, including time zone
tsquery		text search query
tsvector		text search document
txid_snapshot		user-level transaction ID snapshot (deprecated; see pg_snapshot
uuid		universally unique identifier
xml		XML data

Creación de Elementos

- Usualmente, en nuestro rol de DS, MLE, etc. Vamos a estar utilizando creación sobre todo de tablas.
- Una parte importante de las tablas es el tipo de datos que contienen en sus columnas. Esto varia ligeramente entre motor y motor, en el caso de postgresSQL, los mas importantes son:
  - Numéricos (Int, Float, etc.)
  - Caracteres (char, varchar, text, …)
  - Fecha y tiempo (timestamp, date, ….)
  - Objetos (XML,json,…)
  - Sequencias autogeneradas (Serial, …)

Modificación de tablas

- `ALTER TABLE tesTable ADD ( column datatype [DEFAULT expr]);`
- `ALTER TABLE tesTable`
- `ALTER COLUMN columnName datatype;`
- `ALTER TABLE tesTable`

- `ALTER COLUMN columnName SET NOT NULL;`
- `ALTER TABLE tesTable`

- `ALTER CHECK ( col::expression )`
- `ALTER TABLE tesTable`

- `ALTER CONSTRAINT [constraintName];`
- `CHECK ( col::expression )`

```
ALTER TABLE tesTable
ALTER CONSTRAINT [relation_name];
[FOREIGN|PRIMARY] KEY (col.O)
[[REFERENCES] otherTestTable(col.R)]
```

- En PostgreSQL, tenemos que la sentencia ALTER esta nos permite cambiar atributos de nuestro objeto, en particular ALTER TABLE nos permite operar sobre nuestras tablas:
  - Agregar columnas
  - Cambiar la definición de las mismas
  - Agregar o borrar *constraints* (relaciones)

Ejemplo

```
CREATE SEQUENCE "theme_id_seq"
INCREMENT 1
MINVALUE 1
MAXVALUE 2147483647
START 1
CACHE 1;
SELECT setval('"public"."theme_id_seq"', 1, TRUE);

CREATE TABLE theme(
        theme_id int DEFAULT nextval('theme_id_seq'::regclass) NOT NULL,
        namevarchar(256),
        parent_id int,
        CONSTRAINT pk_themes PRIMARY KEY (theme_id)
        );

ALTER TABLE themes ADD CONSTRAINT (fk_themes_id)
FOREIGN KEY (parent_id) REFERENCES themes (theme_id)
```

Ejemplo opción 2

```
CREATE TABLE theme(
        theme_id serial NOT NULL,
        name varchar(256),
        parent_id serial
        CONSTRAINT pk_themes
        PRIMARY KEY (theme_id)
        );

ALTER TABLE themes ADD CONSTRAINT (fk_themes_id)
FOREIGN KEY (parent_id) REFERENCES themes (theme_id)
```

DML

Data Manipulation Language

Insertando valores

- INSERT nos permite agregar filas en una tabla, es la única manera que nos provee SQL directamente para ingresar datos.
- `INSERT INTO tableName [(col1[, col2,…])]`

  `VALUES (val1[, val2, …]);`

- `INSERT INTO tableName`

  `VALUES (val1, val2, … , val3);`

- `INSERT INTO themes(name,parent_id)`

  `VALUES (‘starwarsdeathstar’, 128);`

- `INSERT INTO sets`

  `VALUES (1,‘dome’,1989,128,129);`

ESTRUCTURA DE QUERIES

ESTRUCTURA GENERAL DE UNA QUERY

SELECT

Se seleccionan todos los campos que se desean consultar

FROM

Se seleccionan las tablas de las cuales se extraen esos campos

WHERE

Se filtran los resultados según las condiciones deseadas

GROUP BY

Se agrupan los resultados según todos los campos no calculados

HAVING

Se filtran los resultados según las condiciones deseadas para los campos calculados

ORDER BY

Se ordenan los resultados según las columnas que se elijan con el orden deseado

SELECT AND DEFINE TABLES

CONDITIONALS AND OPERATORS

AGGREGATE FUNCTIONS

JOINS

SELECT AND DEFINE TABLES

CONDITIONALS AND OPERATORS

AGGREGATE FUNCTIONS

JOINS

SELECT

- Select partial table contents by placing restrictions on rows to be included in output.
- Add conditional restrictions to SELECT statement, using WHERE clause.

- Syntax:

```
SELECT *
FROM table_name;

SELECT Col1,Col2,…,Coln
FROM table_name
[ WHERE conditionlist ] ;
```

CREATE, DROP AND DELETE TABLES

CREATE TABLE

- Used to create a new table
- Syntax:

```
CREATE TABLE scheme.table_name (
column1 datatype primary key,
column2 datatype,
column3 datatype,
.... );
CREATE TABLE scheme .table_name AS (
SELECT columnlist
FROM table_name);
```

DELETE TABLE

- Used to drop data in a table
- Syntax:

```
DELETE FROM table_name
[WHERE conditionlist ];
```

DROP TABLE

- Used to drop an entire table
- Syntax:

```
DROP TABLE table_name;
```

ADD, UPDATE AND ALTER TABLE ROWS

ADDING TABLE ROWS

- To enter data into table
- Syntax:

INSERT INTO table_name

[(column1, column2, ...)]

VALUES
(value1,value2,...);

UPDATE

- Modify data in a table
- Syntax:

UPDATE tablename

SET columnname = expression

WHERE conditionlist];

- If more than one attribute is to be updated in row, separate corrections with commas

ALTER

- Alter conditions from a table
- Syntax:

1)ALTER TABLE table_name

ADD COLUMN new_c TYPE;

2)ALTER TABLE table_name

RENAME COLUMN col_1

TO new_name;

SELECT AND DEFINE TABLES

CONDITIONALS AND OPERATORS

AGGREGATE FUNCTIONS

JOINS

CONDITIONALS: COMPARISON OPERATORS

Used in conditional expressions

- Syntax:

SELECT columnlist

FROM tablelist

WHERE <expression>[comparison operator]<expression>;

Operator	Description
=	Equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
<>	Not equal to

CONDITIONALS: LOGICAL OPERATORS

- Combine conditional expressions to return true or false values:
- Syntax:

```
SELECT columnlist
FROM tablelist
WHERE condition1 AND (condition2 OR condition3) ...;
```

```
SELECT columnlist
FROM tablelist
WHERE NOT condition1 ...;
```

CONDITIONALS: SPECIAL OPERATORS

BETWEEN -> BETWEEN DATE '2020-08-01' AND DATE '2020-08-31'

- Used to check whether attribute value is within a range

IS NULL or IS NOT NULL -> SURENAME IS NULL

- Used to check whether attribute value is null

LIKE -> TRACKING_CODE LIKE ‘%DELIVERED%’

- Used to check whether attribute value matches given string pattern

IN or NOT IN -> ID IN (SELECT * FROM table_one)

- Used to check whether attribute value matches any value within a value list

CONDITIONALS: SPECIAL OPERATORS

DISTINCT -> SELECT DISTINCT NAMES FROM CLIENTES;

- Limits values to unique values

AS

- Use to rename columns
- Syntax:

SELECT column1 AS Name, column2 AS Surname

FROM table1;

ORDER BY -> SELECT NAME, QUANTITY FROM CLIENTES ORDER BY 2 DESC;

- Use to sort data in descending or ascending order.

TIPS

- Mantener estructura al escribir
- ; -> al final de cada query
- Distinct -> muestra únicos, usar siempre
- Limit / Top -> limita los resultados = más rápido
- ‘%Alvarez’ -> el % == cualquier caracter

SELECT AND DEFINE TABLES

CONDITIONALS AND OPERATORS

AGGREGATE FUNCTIONS

JOINS

AGGREGATE FUNCTIONS

Common Aggregate functions :

- Avg (expression)
- Count (expression) or Count (*)
- Sum (expression)
- Min/Max (expression)

Statistics Aggregate functions:

- Corr (Y, X)
- Sttdev (expression)
- Variance (expression)

AGGREGATE FUNCTIONS

- Aggregate functions compute a single result from a set of input values.
- Syntax:

SELECT column_name(s), AGGREGATE_FUNCTION() AS column_name

FROM table_name

WHERE condition

GROUP BY column_name(s)

HAVING condition

ORDER BY column_name(s);

* AGGREGATE FUNCTIONS need a GROUP BY clause always !!!!!!!!!!

* HAVING clause is use to conditionate aggregate functions.

SELECT AND DEFINE TABLES

CONDITIONALS AND OPERATORS

AGGREGATE FUNCTIONS

JOINS

JOINS:
UNION CLAUSE AND CARTESIAN JOIN

UNION

- Syntax:

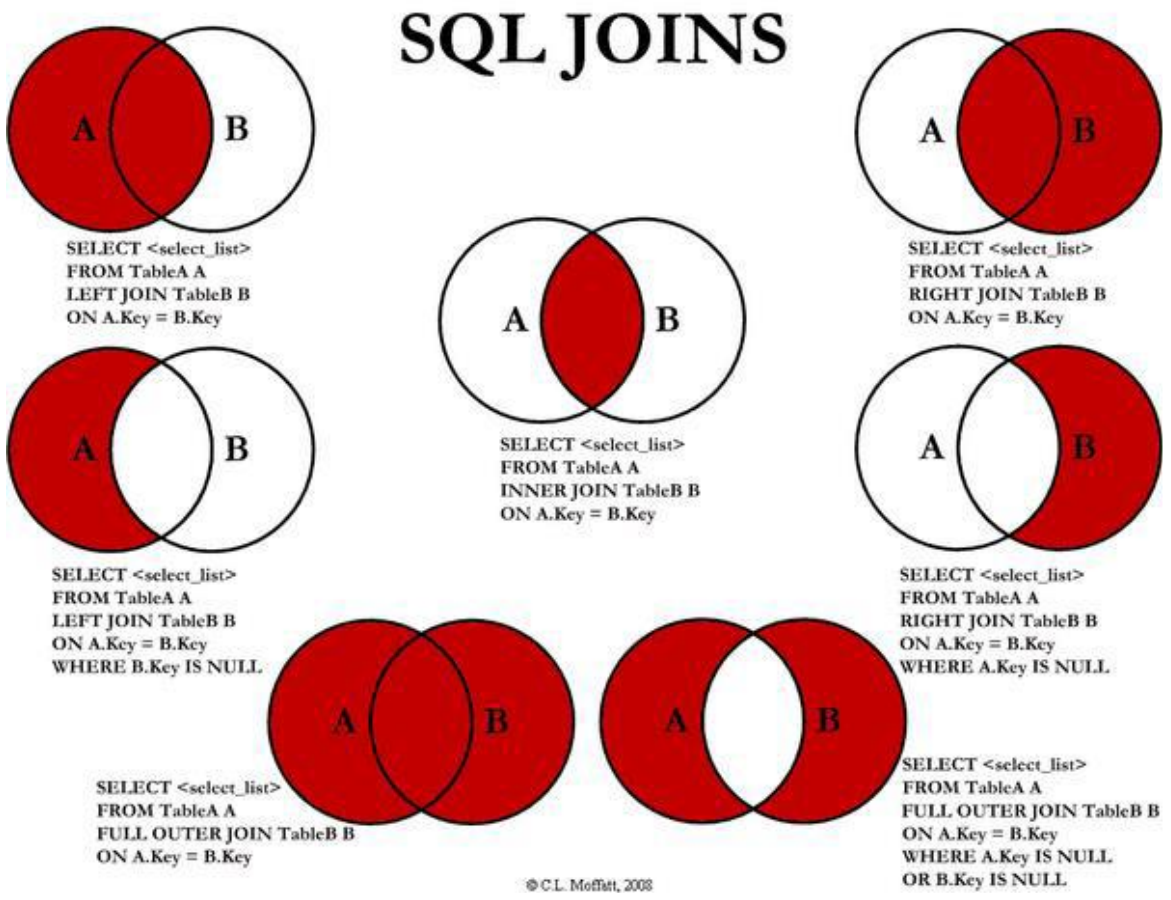
```
SELECT exp1,exp2,exp_n
FROM table_name
[WHERE condition]
UNION ALL
SELECT exp1,exp2,exp_n
FROM table_name
[WHERE condition];
```

CARTESIAN

- Syntax:

```
SELECT t1.a, t1.b, t2.aa, t2.bb
FROM table_name AS t1
[LEFT/INNER/etc ] JOIN table_name AS t2
        ON t1.columnY=t2.colunmX;
```

JOINS


SQL JOINS
A
B
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL
SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL
OR B.Key IS NULL
© C.L. Moffatt, 2008

CARTESIAN JOIN EXAMPLE

Customers

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00
7	Muffy	24	Indore	10000.00

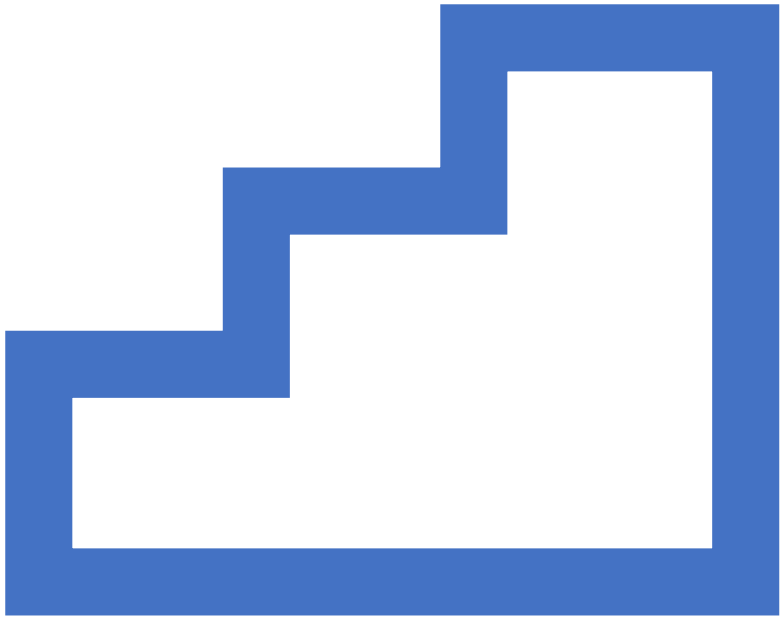
Orders

OID	DATE	CUSTOMER_ID	AMOUNT
102	2009-10-08 00:00:00	3	3000
100	2009-10-08 00:00:00	3	1500
101	2009-11-20 00:00:00	2	1560
103	2008-05-20 00:00:00	4	2060

ID	NAME	AMOUNT	DATE
1	Ramesh	3000	2009-10-08 00:00:00
1	Ramesh	1500	2009-10-08 00:00:00
1	Ramesh	1560	2009-11-20 00:00:00
1	Ramesh	2060	2008-05-20 00:00:00
2	Khilan	3000	2009-10-08 00:00:00
2	Khilan	1500	2009-10-08 00:00:00
2	Khilan	1560	2009-11-20 00:00:00
2	Khilan	2060	2008-05-20 00:00:00
3	kaushik	3000	2009-10-08 00:00:00
3	kaushik	1500	2009-10-08 00:00:00
3	kaushik	1560	2009-11-20 00:00:00
3	kaushik	2060	2008-05-20 00:00:00
4	Chaitali	3000	2009-10-08 00:00:00
4	Chaitali	1500	2009-10-08 00:00:00
4	Chaitali	1560	2009-11-20 00:00:00
4	Chaitali	2060	2008-05-20 00:00:00
5	Hardik	3000	2009-10-08 00:00:00
5	Hardik	1500	2009-10-08 00:00:00
5	Hardik	1560	2009-11-20 00:00:00
5	Hardik	2060	2008-05-20 00:00:00
6	Komal	3000	2009-10-08 00:00:00
6	Komal	1500	2009-10-08 00:00:00
6	Komal	1560	2009-11-20 00:00:00
6	Komal	2060	2008-05-20 00:00:00
7	Muffy	3000	2009-10-08 00:00:00
7	Muffy	1500	2009-10-08 00:00:00
7	Muffy	1560	2009-11-20 00:00:00
7	Muffy	2060	2008-05-20 00:00:00

DUDAS?

SQL
PRIMEROS
PASOS



SUBQUERIES

CASE WHEN

OVER PARTITION BY

LAG

SUBQUERIES

CASE WHEN

OVER PARTITION BY

LAG

NESTED

- Syntax:

```
SELECT columnlist
FROM (SELECT columnlist FROM tablelist);

SELECT columnlist
FROM tablelist
WHERE columnlist in ( SELECT columnlist FROM tablelist);

SELECT product, price, ( SELECT AVG (price) FROM tablelist ) avg_price
FROM tablelist
```

WITH

Permite hacer consultas intermedias (de la misma forma que una subquery) pero de forma

ordenada y más simple cuando tenemos que hacer muchas. La idea es evitar anidar una subquery adentro de la otra y adentro de la otra y adentro de la otra, etc.

- Syntax

```
WITH
        tabla1 AS (SELECT columnlist FROM tablelist),
        tabla2 AS (SELECT columnlist FROM tablelist)
SELECT * FROM tabla1 INNER JOIN tabla2 ON tabla1.campo1 = tabla2.campo2;
```

SUBQUERIES

CASE WHEN

OVER PARTITION BY

LAG

CASE

The SQL CASE expression is a generic conditional expression, similar to if/else statements in other languages:

- Syntax:

SELECT name,

CASE WHEN condition THEN result [WHEN ...] [ELSE result] END column_name

FROM tablelist

SUBQUERIES

CASE WHEN

OVER PARTITION BY

LAG

OVER PARTITION BY ORDER BY

Es una función agregada aplicada a una partición o subconjunto del resultado de una consulta. Se la llama WINDOW FUNCTION

Diferencia con GROUP BY:

- GROUP BY se usa con las funciones de agregación (suma, promedio, etc.) para indicarle a la función todos los campos que debe tener en cuenta para el cálculo.
- OVER clause define una ventana sobre la que hacer el calculo que NO NECESARIAMENTE son todas los campos de la tabla.

OVER PARTITION BY

PARTITION BY: indica qué campos de la tabla mirar para hacer el cálculo solamente sobre los registros que estén dentro de esa Ventana.

- Syntax

```
SELECT depname , empno , salary ,
AVG(salary ) OVER ( PARTITION BY depname )
FROM empsalary ;
```

+ info→diferencia entre aggregate functions y Windows functions

https://learnsql.com/blog/window-functions-vs-aggregate-functions/

depname	empno	salary	avg
develop	11	5200	5020.0000000000000000
develop	7	4200	5020.0000000000000000
develop	9	4500	5020.0000000000000000
develop	8	6000	5020.0000000000000000
develop	10	5200	5020.0000000000000000
personnel	5	3500	3700.0000000000000000
personnel	2	3900	3700.0000000000000000
sales	3	4800	4866.6666666666666667
sales	1	5000	4866.6666666666666667
sales	4	4800	4866.6666666666666667

(10 rows)

ORDER BY

ORDER BY: se usa si estamos haciendo una función que necesita la información de campos anteriores o posteriores. Hay que especificar el orden de los campos para que la Ventana esté correctamente seleccionada.

- Syntax

```
SELECT
        depname , empno , salary ,
        rank () OVER PARTITION BY depname (ORDER BY salary DESC )
FROM empsalary ;
```

depname	empno	salary	rank
develop	8	6000	1
develop	10	5200	2
develop	11	5200	2
develop	9	4500	4
develop	7	4200	5
personnel	2	3900	1
personnel	5	3500	2
sales	1	5000	1
sales	4	4800	2
sales	3	4800	2

(10 rows)

ROWS: PRECEDING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular .

Por ejemplo ROWS BETWEEN 3 PRECEEDING AND CURRENT ROW toma solamente los 3 anteriores, y la fila del cálculo .

- Syntax

```
SELECT Year , DepartmentID , Revenue,
SUM(Revenue) OVER (
        PARTITION BY DepartmentID
        ORDER BY Year
        ROWS BETWEEN 3 PRECEDING AND CURRENT ROW
) AS CurrentAndPrev3
FROM revenue
ORDER BY DepartmentID , Year
```

ROWS: FOLLOWING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular. Por ejemplo ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING toma la fila del cálculo y las 3 posteriores.

- Syntax

```
SELECT Year , DepartmentID , Revenue
        SUM(Revenue ) OVER (
        PARTITION BY DepartmentID ORDER BY Year
        ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING
        ) AS CurrentAndNext3
FROM revenue ORDER BY DepartmentID , Year
```

ROWS:
PRECEDING AND FOLLOWING

ROW or RANGE: Permite que dentro de la ventana establecida por el PARTITION BY se tomen solamente ciertas filas o un rango en particular . Por ejemplo ROWS BETWEEN 1 PRECEEDING AND 1 FOLLOWING toma 1 fila anterior, la fila del cálculo y la posterior.

- Syntax

```
SELECT Year , DepartmentID , Revenue
SUM(Revenue ) OVER (
        PARTITION BY DepartmentID
        ORDER BY [Year]
        ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING)
AS BeforeAndAfter
FROM revenue
ORDER BY DepartmentID , Year
```

	Year	DepartmentID	Revenue	CurrentAndPrev3
1	1998	1	10030	10030
2	1999	1	20000	30030
3	2000	1	40000	70030
4	2001	1	30000	100030
5	2002	1	90000	180000
6	2003	1	10300	170300
7	2004	1	10000	140300
8	2005	1	20000	130300
9	2006	1	40000	80300
10	2007	1	70000	140000
11	2008	1	50000	180000
12	2009	1	20000	180000
13	2010	1	30000	170000

	Year	DepartmentID	Revenue	CurrentAndNext3
1	1998	1	10030	100030
2	1999	1	20000	180000
3	2000	1	40000	170300
4	2001	1	30000	140300
5	2002	1	90000	130300
6	2003	1	10300	80300
7	2004	1	10000	140000
8	2005	1	20000	180000
9	2006	1	40000	180000
10	2007	1	70000	170000
11	2008	1	50000	180000
12	2009	1	20000	140000

	Year	DepartmentID	Revenue	BeforeAndAfter
1	1998	1	10030	30030
2	1999	1	20000	70030
3	2000	1	40000	90000
4	2001	1	30000	160000
5	2002	1	90000	130300
6	2003	1	10300	110300
7	2004	1	10000	40300
8	2005	1	20000	70000
9	2006	1	40000	130000
10	2007	1	70000	160000
11	2008	1	50000	140000
12	2009	1	20000	100000
13	2010	1	30000	130000
14	2011	1	80000	120000

SUBQUERIES

CASE WHEN

OVER PARTITION BY

LAG AND LEAD

LAG AND LEAD FUNCTION

Lag da acceso a un registro previo específico , y Lead al siguiente registro

- Syntax:

```
SELECT
LAG(expression [,offset default_value ]]) OVER (
        PARTITION BY partition_expression , ...
        ORDER BY sort_expression ASC | DESC ]);
```

- Ejemplo

```
SELECT year, amount,
LAG (amount,1) OVER ( ORDER BY year ) prevs_year_sales
FROM table1;
```

	year smallint	amount numeric	previous_year_sales numeric
1	2018	5021.00	[null]
2	2019	4944.00	5021.00
3	2020	5137.00	4944.00

¡TIPS otras funciones

CAST(campo as tipo) -> convierte el formato del dato

campo::FORMATO -> convierte el formato del dato ej : sku ::INT

DATE_TRUNC(‘datepart ’, fecha) -> truncar fechas

To_date -> transforma a fecha

Date(dd,mm,yyyy) -> crea fecha

TRIM() -> quita espacios

EXTRACT(‘datepart ’ from fecha) Extraer una parte dato de la fecha

Group by 1,2,5…. Hacer referencia a los campos segun orden de ocurrencia

FLOOR()

CEILING() -> permite redondear un numero para abajo o para arriba respectivamente