CePETel
Sindicato de los Profesionales de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Formación Profesional en CePETel 2023

Desde la Secretaria Técnica del Sindicato CePETel convocamos a participar del siguiente curso de formación profesional:

Introducción a Bases de Datos y Programación SQL

Clases: 8 de 3 horas c/u de 18:00 a 21:00 hs.
Días que se cursa: los días martes 5, 12, 19 y 26 de septiembre; 3, 10, 17 y 24 de octubre.
Modalidad: a distancia (requiere conectarse a la plataforma Zoom en los días y horarios indicados precedentemente).
Docentes: María Trinidad Aquino y Raúl Alejandro Grassi.

La capacitación es:

- **Sin cargo para afiliados y su grupo familiar directo.**
- **Sin cargo para encuadrados con convenio CePETel.**
- **Con cargo al universo no contemplado en los anteriores.**

Informes: enviar correo a tecnico@cepetel.org.ar

Inscripción (hasta el 4 de septiembre 12:00 hs): ingresar al formulario (se recomienda realizar el registro por medio de una cuenta de correo personal y ***no utilizar dispositivos de la empresa para acceder al link***).

https://forms.gle/R2BjoPCj14BeSgmT9

Temario:

Módulo 1
Conceptos de Bases de Datos y Estructuras

- Introducción a Bases de Datos. Definición
- Bases de Datos Relacionales. Concepto de ACID
- Base de Datos NoSQL. Tipos y cuándo se usa cada una.
- Conceptos de servidor SQL y motor de base de datos
- Fuentes de Datos. Externas e Internas
- Tipos de Datos
- Persistencia Políglota. Teorema de CAP

TP: Instalación de PostgreSQL y crear una base de datos

Módulo 2
Modelado de Datos. Normalización

- Modelado de Datos. Relaciones.
- DER – Diagrama de Entidad Relación
- Características de las Relaciones

Ing. Daniel Herrero – Secretario Técnico – CDC

- Grados de una relación
- Herramientas para Modelado de Datos – CA Erwin
- Normalización/Desnormalización.
- Formas de Normalización

TP: Armar un diagrama de DER en tomando un ejemplo real de base de datos, normalizando o desnormalizando en los casos que crea conveniente

Módulo 3
DDL (Data Definition Lenguage)

- Sublenguajes del SQL: DDL, DML, TCL y DCL
- DDL: principales instrucciones
- Comandos para Crear, utilizar y borrar una base de datos
- Tablas
- Comandos para Crear Tablas
- Constraints
- Comandos para Modificar Tablas (Alter)
- Comando para Borrar Tablas. Diferencias entre Drop y Truncate

TP: Programar los comandos de creación de tablas y constrains (PK, FK, NOT NULL, UNIQUE, CHECK, DEFAULT) del ejercicio del módulo 2

Módulo 4
DML (Data Manipulation Lenguage) - SELECT

- Importación de tablas externas.
- Generar consultas utilizando lenguaje SQL.
- Operador SELECT
- Cláusulas FROM y WHERE (predicado de una consulta).
- Ordenamiento de registros (cláusula ORDER BY)
- Agrupamiento de consultas (cláusulas GROUP BY y HAVING).
- Limitar la cantidad de registros resultantes (Cláusula LIMIT)
- Operadores de comparación.
- Operadores lógicos.

TP: 15 Ejercicios usando comandos y cláusulas de DML sobre la base de datos generada en el módulo 3

Módulo 5
DML (Update, Insert, Delete)

- Consulta de Datos Anexados (INSERT).
- Consulta de actualización (UPDATE)
- Consulta de Eliminación (DELETE / TRUNCATE).
- Uso de cláusula CASCADE
- Transacciones – Consistencia de los Datos
- Comandos BEGIN WORK – COMMIT - ROLLBACK

TP: Utilizando la base de datos creada en los módulos anteriores, realizar comandos para modificar, insertar y borrar registros de tablas. Realizar transacciones y ver resultados

Módulo 6
DML (Secuencias, Vistas, Tablas Temporales)
- Definición de Secuencias
- Implementación de distintos tipos de secuencias en PostgreSQL
- Vistas. Definición. Para que se usan
- Comandos para crear VIEWS (Vistas). Chequeo de Integridad
- Tablas Temporales. Definición
- Tipos de Tablas Temporales. Tipos de Creación. ¿Porque usarlas?
- Tablas Volátiles. Tipos de Creación. Diferencias con Tablas Temporales

TP: Ejercicios en PostreSQL para crear secuencias, vistas y tablas temporales.

Módulo 7
DML (Joins, Subconsultas, y Condicionales)
- Operando con más de una tabla. Uso de JOINS
- Tipo de JOINS (Inner Join, Outer Join, Producto Cartesiano)
- Subconsultas (Subquery)
- Uso del condicional CASE
- Combinación de consultas (UNION, INTERSECT, EXCEPT)
- Consultas relacionadas

TP: Ejercicios grupal en PostgreSQL utilizando distintos tipos de JOINS, y operadores UNION, EXCEPT e INTERSECT

Módulo 8
DML (Funciones, Operadores)
- Operadores SET
- Funciones de Texto (String)
- Conversión de Tipos de Datos
- Funciones de Fechas (DATE)
- Funciones de Hora (TIME, TIMESTAMP)

TP: Ejercicios en PostreSQL para usar las funciones disponibles en SQL

Acerca de los docentes

María Trinidad Aquino: 2007-12–03 al momento Analista Senior Marketing – Región Patagonia Movistar, Neuquén.
* Proyecto Canal Presencial (Agentes y CEC) responsable de la construcción y disposición de los datos (KPI´s) para ver su evolución con aporte analítico, diagnóstico y sugerencia de planes de acción para su mejora.
*Proyecto Educador Digital País, referente de la región Patagonia con seguimiento de los KPI´s y evolución.
* Nuevos proyectos del área con análisis de datos, participación en el diseño, elaboración y difusión de lo implementado.

Formación académica:
- 2020-09 – 2021-10 (finalizada) Big Data, Data Engineer (Diplomatura) ITBA, CABA

- 1999-03 - 2003-12 (finalizada) Ciencias Sociales, Licenciada en Relaciones Públicas Universidad Nacional de Lomas de Zamora, Lomas de Zamora

Raúl Alejandro Grassi: desde 1995 hasta la fecha TELEFONICA DE ARGENTINA S.A. Puesto:Analista Senior - Sector Big Data Comercial.
Responsabilidades: Definición de inversiones anuales en capital (CapEx) en base a análisis de proyección comercial. Gestión de proyectos y seguimiento de inversiones. Diseño e implementación de modelos de aseguramiento de satisfacción de clientes. Planeamiento y ejecución de tableros de control y análisis del negocio basado en datawarehousing (heavy user) en los últimos 10 años, programando en SQL y modelado de datos. Análisis y Evaluación de acciones que impacten en cumplimiento de objetivos del Negocio B2C. Analista Senior BI, desarrollo en herramientas de explotación de BI (Microstrategy; Tibco Spotfire, Power BI, Tableau, etc.) y ecosistema Hadoop (Spark, Hive, SQL, procesos de ingestas ETL, ELT, etc.).
Se desempeñó durante 4 años en el sector Data Driven Comercial, promoviendo la cultura
Data Driven y desarrollando tableros de control predictivos y prescriptivos con herramientas de explotación basadas en modelos relacionales/dimensionales.
Experiencia al menos 7 años como líder de proyectos, Manejo de Metodologías Agiles en
posiciones como Stakeholder, Scrum Master y PO.

Formación académica:

- 2020-2021 Licenciatura en Big Data – especialista en Data Engineer - ITBA (Instituto Tecnológico de Buenos Aires)
- 1999 Posgrado en Gestión Gerencial Avanzada (Managemente Executive Program) Universidad Argentina de la Empresa (UADE)
- 1986-1992 Ingeniero Electrónico Universidad de Buenos Aires

María Trinidad Aquino y Raúl Alejandro Grassi dictaron de manera virtual y para el Sindicato CEPETel Big Data & Analytics – Parte 1 durante el año 2022, y durante el 2023 la Parte 2 de dicha formación y como así también el curso de Visualización y Analítica de datos con Power BI.

7. Introducción al email marketing
- Definición de email marketing
- Ventajas y desventajas del email marketing
- Elección del software adecuado para email marketing
- Diseño y estructura efectiva del correo electrónico
- Segmentación adecuada del público objetivo
- Cómo crear campañas de email marketing efectivas

8. Cómo medir el éxito de tus campañas de marketing digital
- Análisis de las métricas clave para medir el éxito
- Interpretación adecuada de los datos
- Optimización contínua basada en los resultados obtenidos

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 1: Conceptos de Bases de datos y estructuras

Bases de Datos

Base de Datos

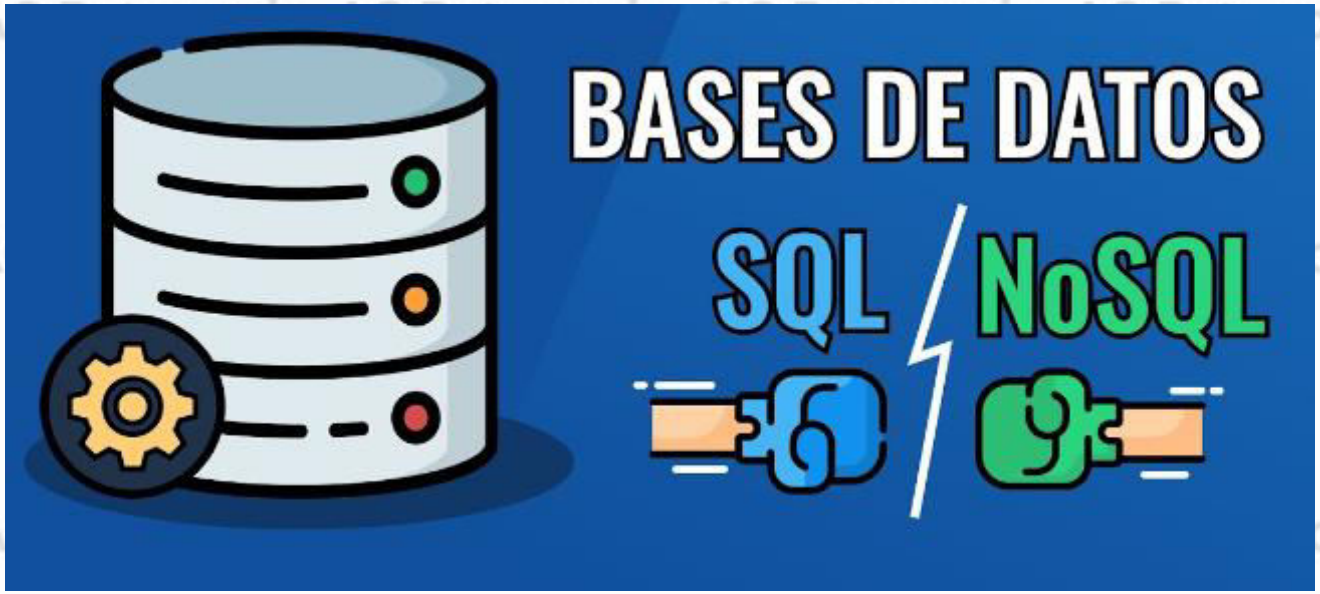
[Video: Historia de las bases de datos]

- Definición
- Sistema de Administración de Bases de Datos
- Bases de Datos Relacionales
- Bases de Datos NoSQL

CePETel

Base de Datos

- Definición
- Sistema de Administración de Bases de Datos
- Bases de Datos Relacionales
- Bases de Datos NoSQL

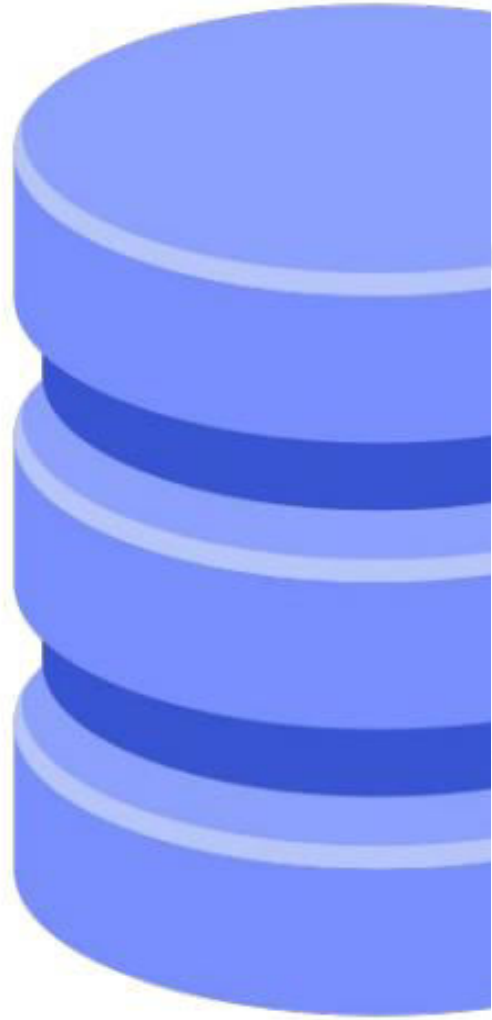


CePETel

Definición de Base de Datos

¿Qué es una Base de Datos?

Una BD es un conjunto de datos persistentes e interrelacionados que es utilizado por los sistemas de aplicación de una empresa, los mismos se encuentran almacenados en un conjunto independiente y sin redundancias o con redundancias mínimas.



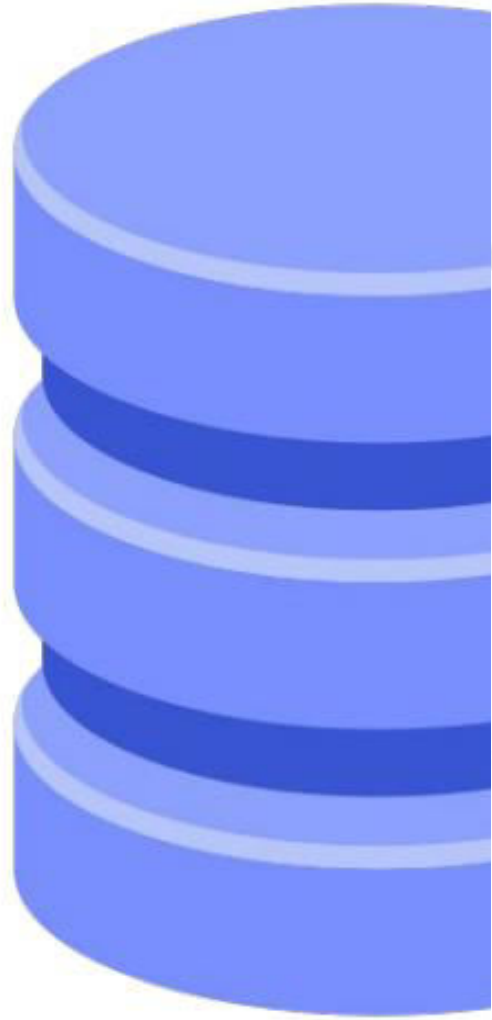
CePETel

Base de Datos

Un poco de Historia

En los últimos 30 de años el mundo de la TI experimento grandes cambios.

- Nuevas arquitecturas de aplicaciones.
- Nuevos paradigmas de programación.
- Nuevas herramientas para desarrollo de software
- Pero algo permaneció constante...



CePETel

Base de Datos

RDMBS – Relational Data Base Management Systems

En los 70´s
Edgar Codd
Modelo Relacional
Algebra Relacional

ORACLE

INFORMIX

IBM DB2

1974
IBM
System R
1er RDBMS
SEQUEL=SQL

1977/79
Lawrence Ellison
SDL luego
Relational Sw Inc.

1980
Relational
Database
Systems

1981
IBM

CePETel

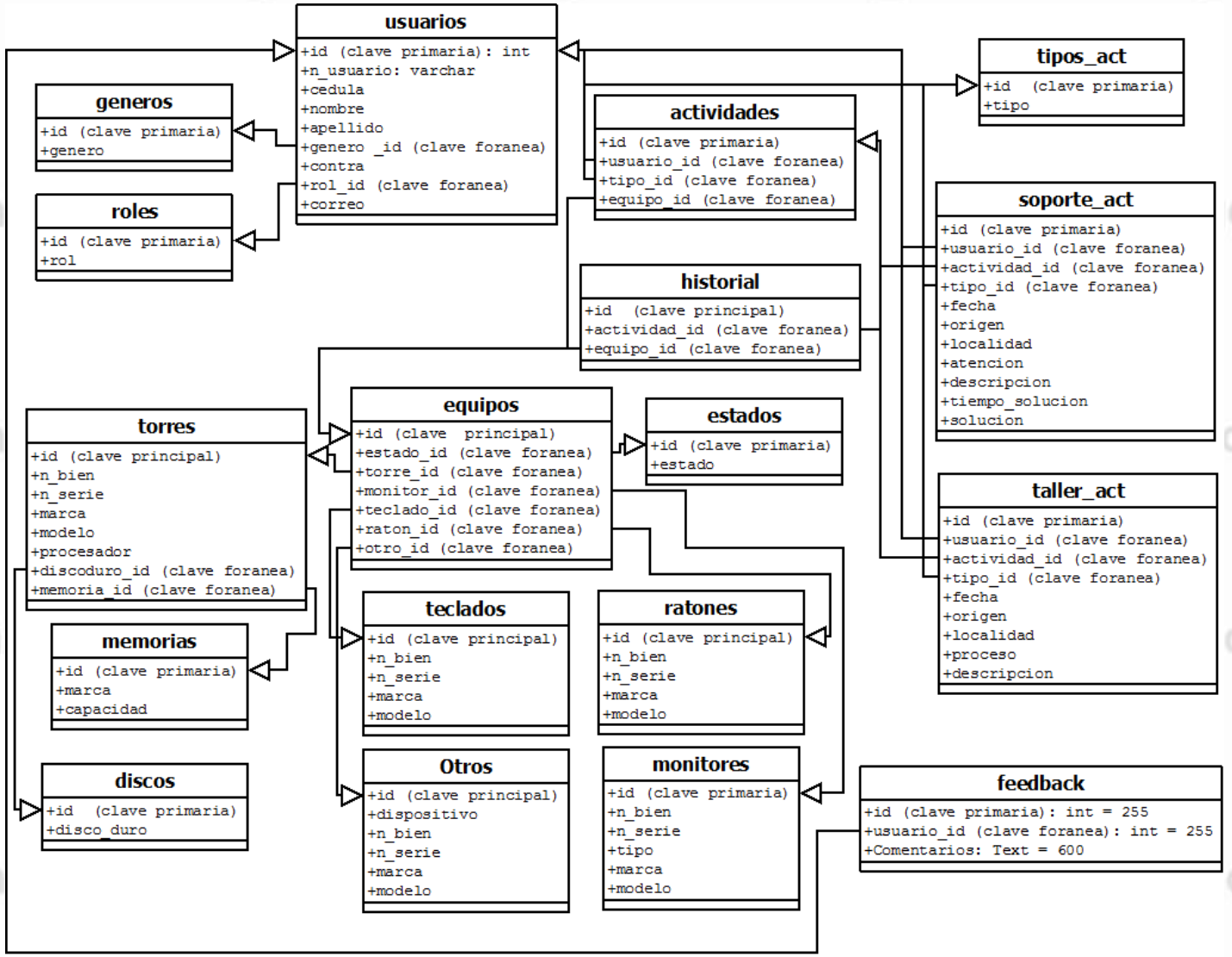
Base de Datos

RDMBS

Estandar de la industria.

Su foco en la ejecución de **transacciones**

- **A**tomicidad
- **C**onsistencia
- **I**solation (Aislamiento)
- **D**urabilidad



CePETel

Base de Datos

ACID

Es el conjunto de propiedades que garantizan que las transacciones de una Base de Datos se procesan de manera fiable.

De forma resumida, las propiedades son:

- ***Atomicidad***: un cambio debe completarse en su totalidad o no modificar nada en absoluto.
- ***Consistencia***: cualquier cambio debe conducir de un estado válido de la base de datos a otro estado válido de acuerdo con las restricciones y el esquema de datos.
- ***Aislamiento (Isolation):*** un cambio no debe afectar a otros cambios que se estén ejecutando al mismo tiempo sobre la base de datos.
- ***Durabilidad:*** una vez completado el cambio, éste debe conservarse, aunque se produzcan fallos en la base de datos o el sistema completo.

CePETel

Base de Datos

OO (Orientado a objetos)

- A mediados de los 90 se hizo más visible el paradigma de desarrollo Orientado a Objetos.
- Era necesario una traducción de objetos a relaciones.
- Se pensó como solución en Bases de Datos específicas para resolver la complejidad.

Bases de Datos Orientadas a Objetos

Se trató de estandarizar el OQL – Object Query Language.

Definición de RDBMS

¿Qué es un sistema de administración de base de datos? (DBMS – Data Base Manager System)

Es un programa que permite administrar los contenidos de una/s base/s de datos almacenadas en disco. También llamado motor de base de datos.

El DBMS ofrece a los usuarios una percepción de la base de datos que está en cierto modo por encima del nivel del hardware y que maneja las operaciones del usuario expresadas en el nivel más alto de percepción.

El DBMS también interpreta y ejecuta todos los comandos SQL que le son enviados.

Entre los motores de base de datos más utilizados podemos nombrar a los siguientes: Oracle, MS SQL Server, MySQL, PostgreSQL, DB2, Informix, SyBase, SQL Lite, entre otros.

Base de Datos

Historia de las BD NoSQL

- Se inicia en 1966 con el surgimiento de las Bases de Datos Jerárquicas – IBM IMS para el programa espacial Apollo.
- En la historia más reciente, Amazon y Google se posicionan cómo líderes en buscar mecanismos de almacenamiento y recuperación para volúmenes de datos enormes.

CePETel

Base de Datos

Historia de las BD NoSQL

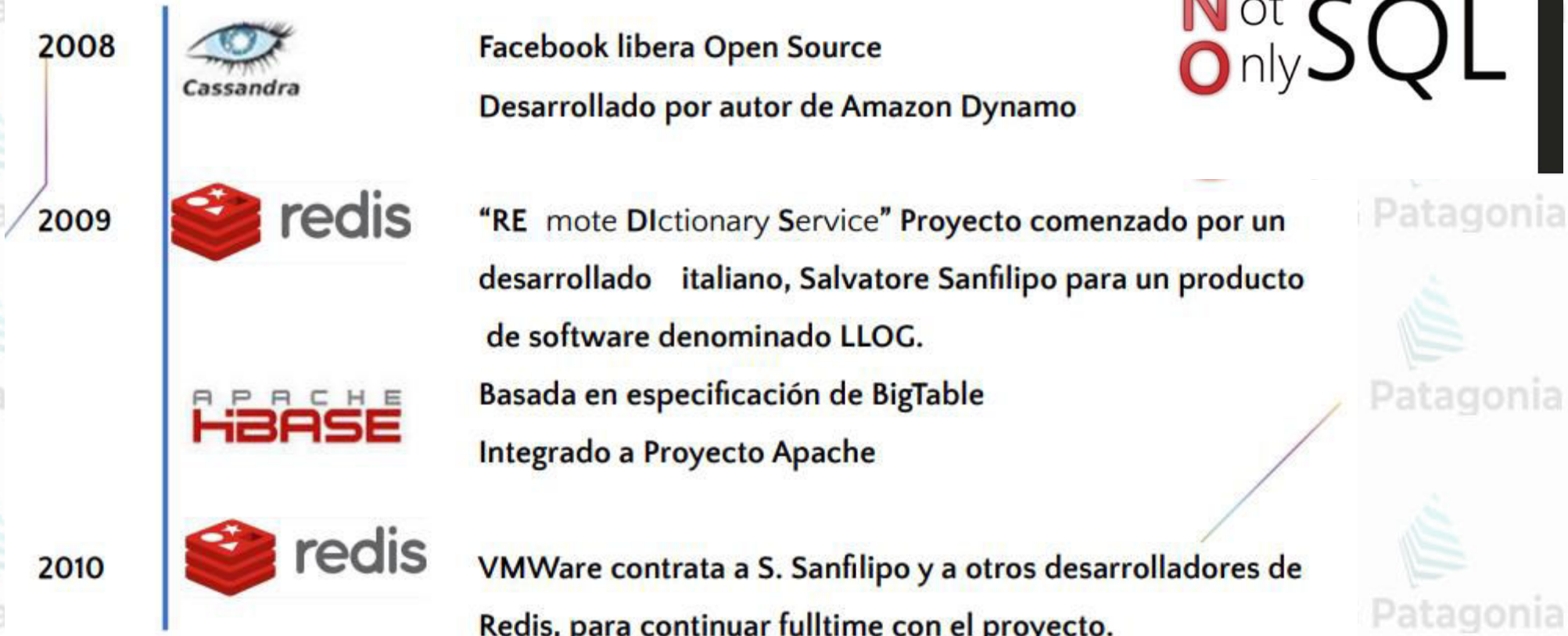
NotOnlySQL

2000	Neo4j the graph database	Comienzo proyecto Desarrollada en JAVA. Basada en Estructuras de Grafos.
2005	CouchDB	Comienzo proyecto Inspirada en Lotus Notes. JavaScript como leng. De consulta. BD basada en documentos.
2006	Google	Proyecto BigTable Primera especificación BD en forma columnar. Escalamiento Horizontal (Pb) Google Reader, G. Maps, G. Earth, Blogger.com
2007	Amazon DynamoDB	Primera especificación BD basadas en clave-valor
	mongoDB	Similar a CouchDB Documentos basados en JSON

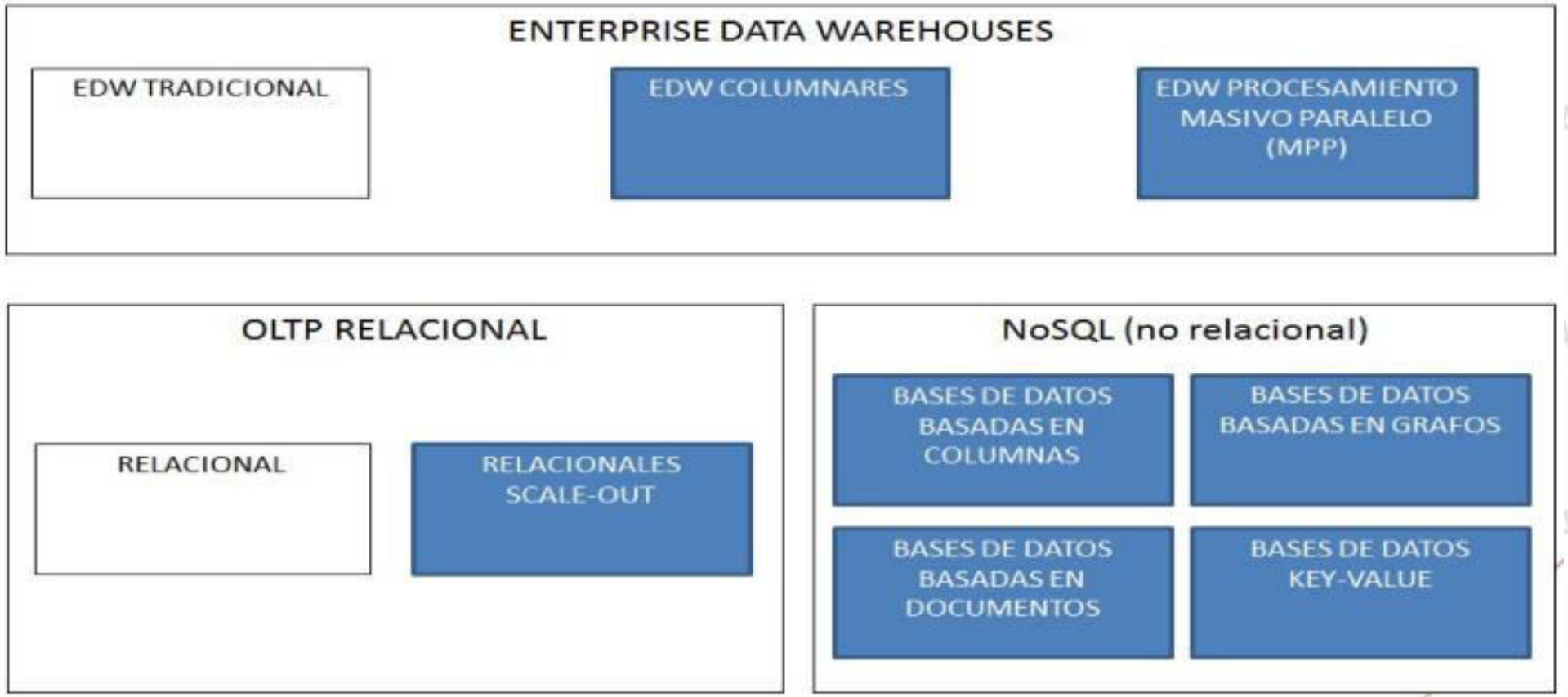
CePETel

Base de Datos

Historia de las BD NoSQL



Mapa de Bases de Datos



CePETel

SQL – DB Engines.com

419 systems in ranking, July 2023

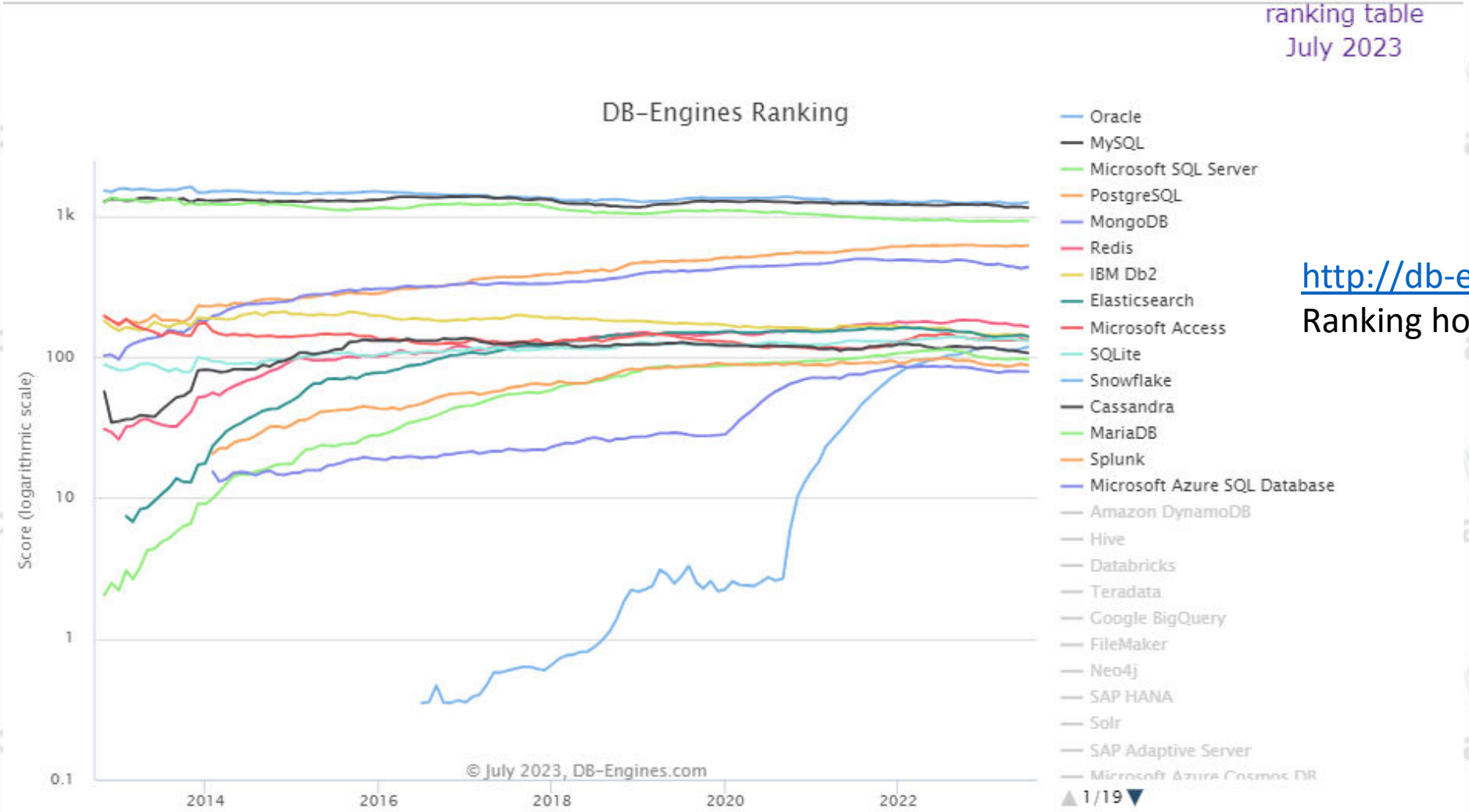
Rank Jul 2023	Rank Jun 2023	Rank Jul 2022	DBMS	Database Model	Score Jul 2023	Score Jun 2023	Score Jul 2022
1.	1.	1.	Oracle	Relational, Multi-model	1256.01	+24.54	-24.28
2.	2.	2.	MySQL	Relational, Multi-model	1150.35	-13.59	-44.53
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	921.60	-8.47	-20.53
4.	4.	4.	PostgreSQL	Relational, Multi-model	617.83	+5.01	+1.96
5.	5.	5.	MongoDB	Document, Multi-model	435.49	+10.13	-37.49
6.	6.	6.	Redis	Key-value, Multi-model	163.76	-3.59	-9.86
7.	7.	7.	IBM Db2	Relational, Multi-model	139.81	-5.07	-21.40
8.	8.	8.	Elasticsearch	Search engine, Multi-model	139.59	-4.16	-14.74
9.	9.	9.	Microsoft Access	Relational	130.72	-3.73	-14.37
10.	10.	10.	SQLite	Relational	130.20	-1.02	-6.48
11.	11.	↑ 13.	Snowflake	Relational	117.69	+3.55	+18.53
12.	12.	↓ 11.	Cassandra	Wide column	106.53	-2.03	-7.88
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	96.10	-1.21	-16.42
14.	14.	14.	Splunk	Search engine	87.12	-2.34	-11.09
15.	↑ 16.	15.	Microsoft Azure SQL Database	Relational, Multi-model	78.96	-0.01	-5.94
16.	↓ 15.	16.	Amazon DynamoDB	Multi-model	78.81	-1.10	-5.13
17.	17.	17.	Hive	Relational	72.87	-2.65	-6.61
18.	18.	↑ 22.	Databricks	Multi-model	68.47	+2.65	+17.25
19.	19.	↓ 18.	Teradata	Relational, Multi-model	60.25	-2.39	-10.67
20.	20.	↑ 24.	Google BigQuery	Relational	55.42	+0.78	+6.53

http://db-engines.com/en/ranking

Ranking hoy

CePETel

SQL – DB Engines.com



http://db-engines.com/en/ranking
Ranking hoy

CePETel

NoSQL – Leaders

CePETel

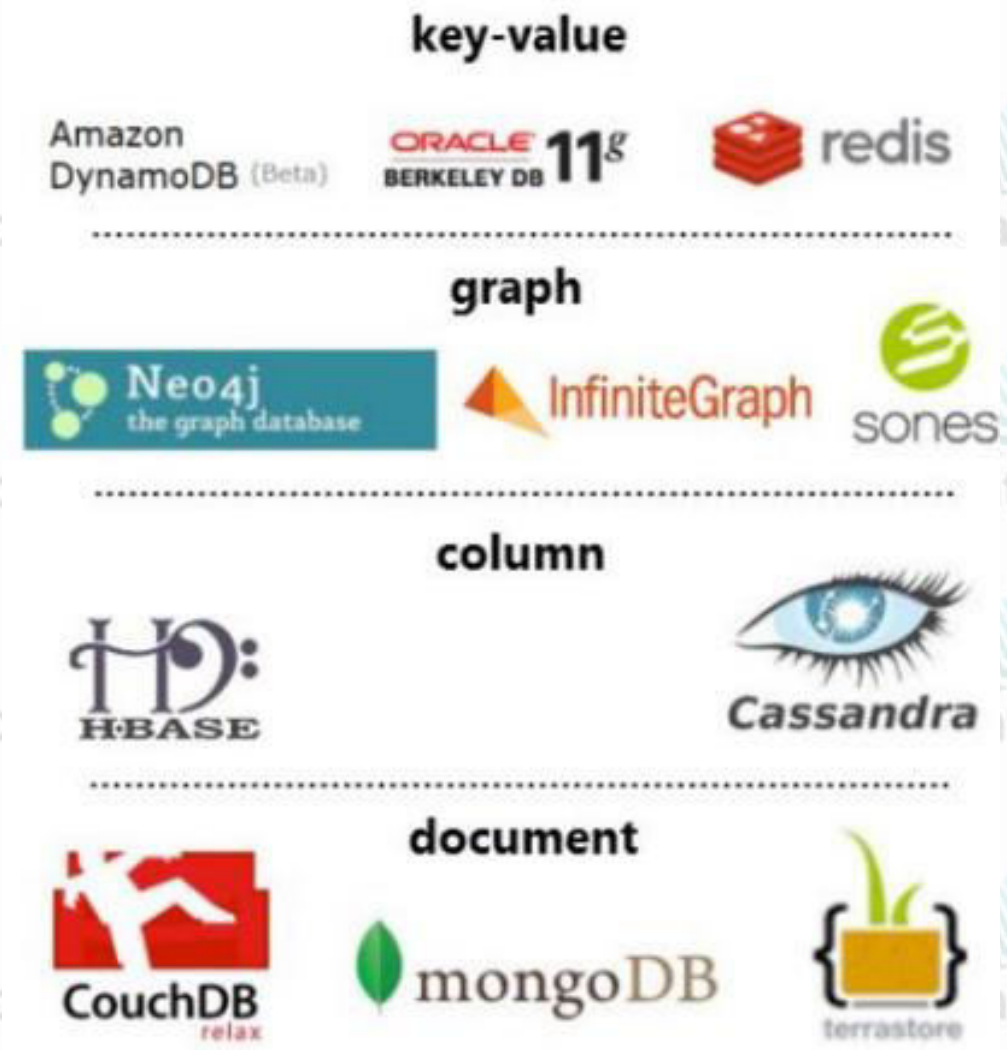
Bases de Datos NoSQL

¿Qué es NoSQL?

Sistemas de gestión de bases de datos que difieren del modelo clásico de bases de datos relacionales: no sólo usan SQL como lenguaje de consulta, los datos almacenados no requieren estructuras fijas como tablas, no garantizan consistencia plena y escalan horizontalmente.

Not Only SQL

Surgieron para complementar a las bases de datos tradicionales, no para reemplazarlas



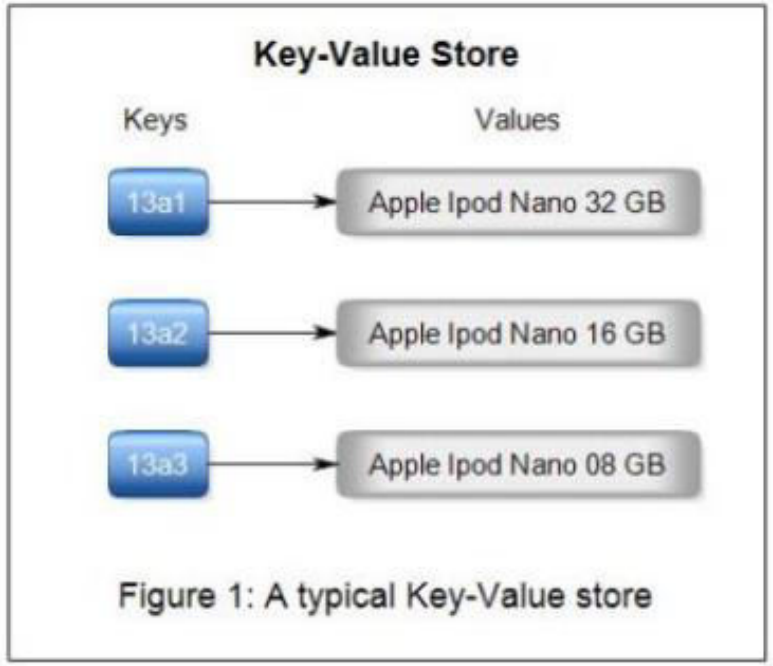
CePETel

Bases de Datos NoSQL

NoSQL – Key Value DB

¿ Cuándo se Usan ?

- Almacenar información de sesiones
- Perfiles de Usuarios
- Información de carros de compras



Pinterest

CePETel

Bases de Datos NoSQL

NoSQL – Columnar DB

¿ Cuándo se Usan ?

CMS, blogging

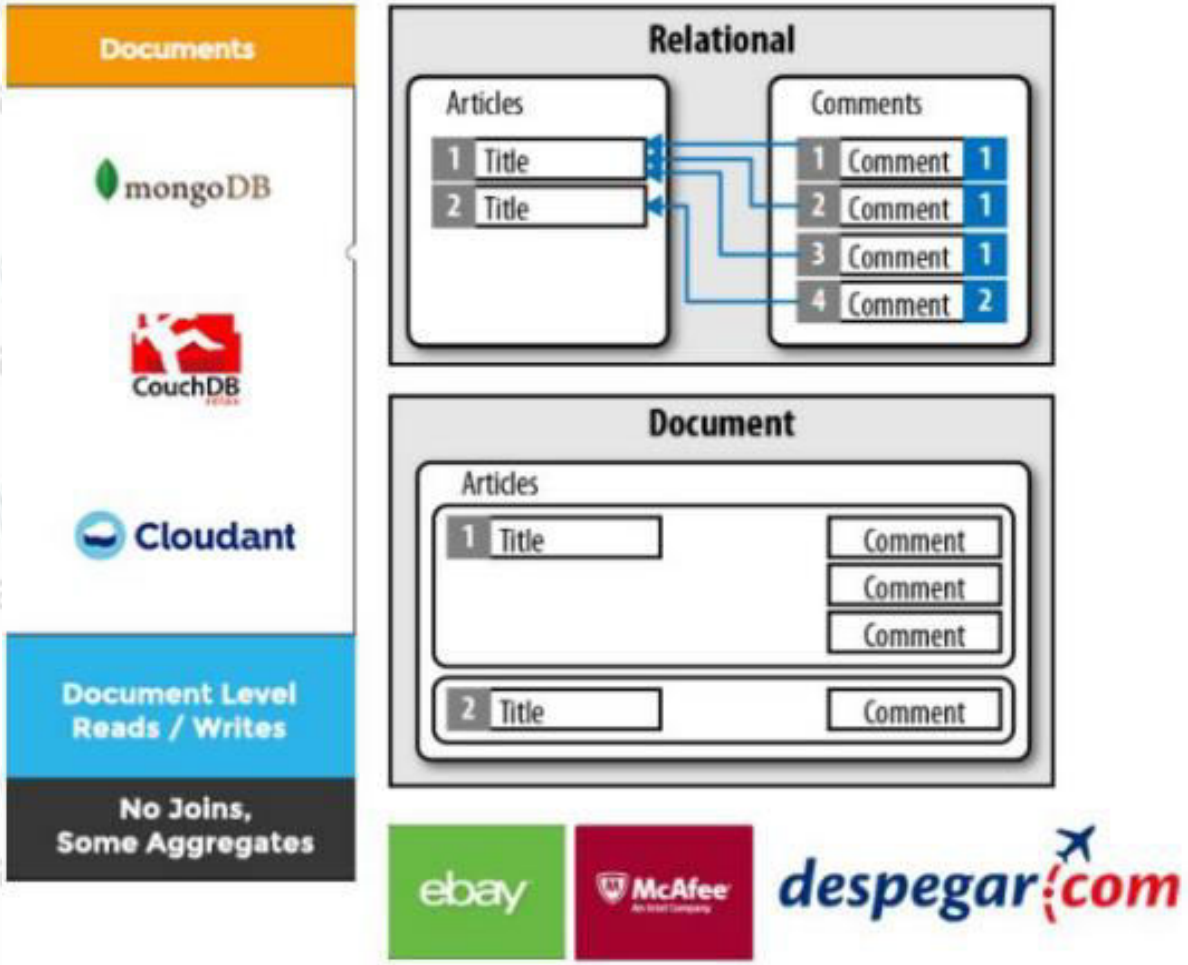
Web-analytics / Real-Time analytics

Expiring

Time series

IoT Metrics

Bases de Datos NoSQL



NoSQL – Document Base DB

¿ Cuándo se Usan ?

- Logging de Eventos
- CMS, blogging
- Web-analytics / Real-Time analytics
- E-Commerce
- Startups/WebApps

CePETel

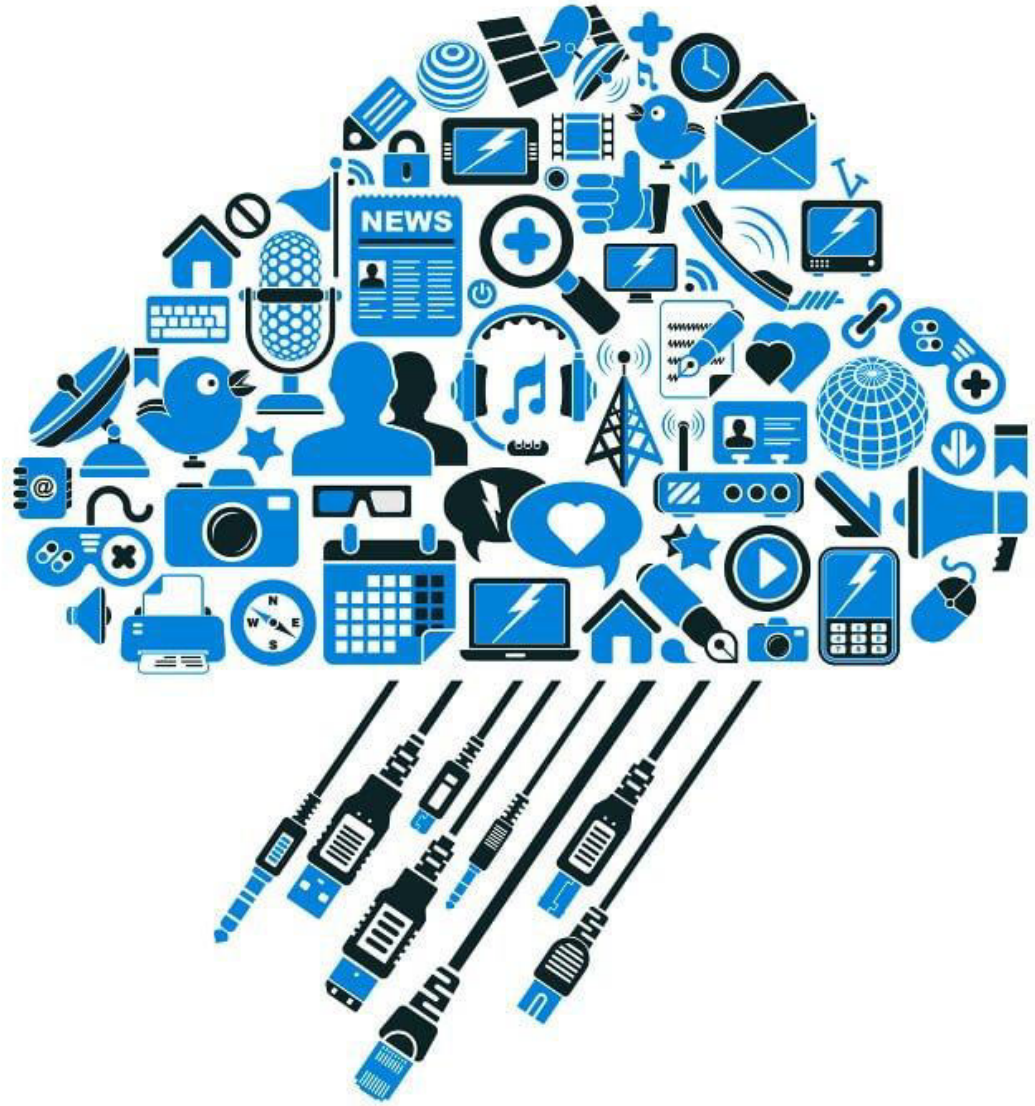
Bases de Datos NoSQL

NoSQL – GRAFH DB

¿ Cuándo se Usan ?

- Datos interconectados
- Servicios de Ruteo / Despachos
- Motores de recomendaciones

CePETel

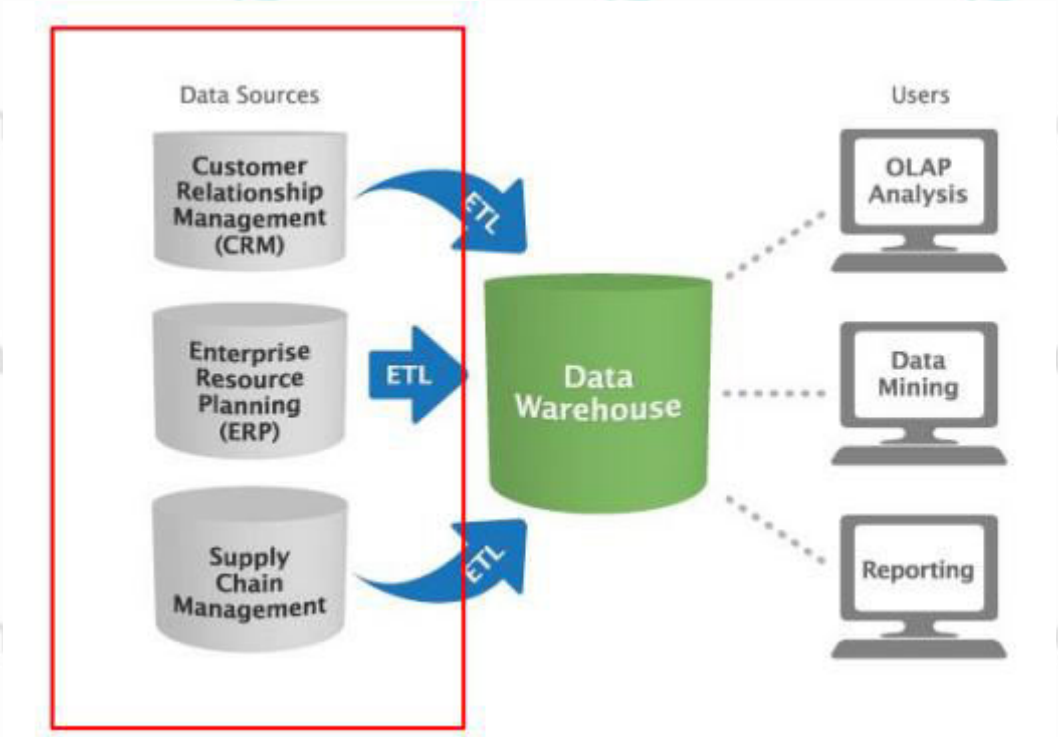


Fuentes de Datos

CePETel

Data Warehouse (DW)

"Es un repositorio de datos integrado, no volátil, variable en el tiempo, orientado al negocio, organizado de forma tal que facilita el análisis de grandes volúmenes de datos para la toma de decisiones"



CePETel

Fuentes de Datos

Fuentes Internas

- ERP
- CRM
- SCM
- MES
- BPM
- Sistemas Legacy
- Planillas de Cálculo
- Stream Events (Sensores, medidores)

Fuentes Externas a la Organización

- Redes Sociales
- Datos de Cámaras empresariales
- Open Data - Gobierno
- Web Crawling Data

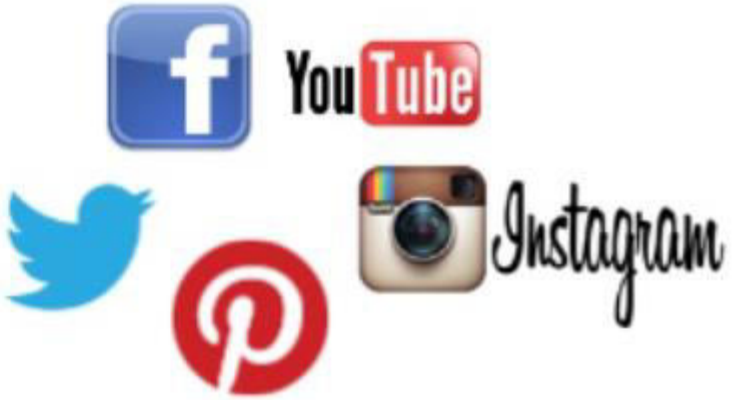
Formatos Posibles

- Bases de Datos Relacionales
- Bases de Datos NoSQL
- Archivos (CSV, AVRO, JSON, XML, XLS, VSAM, comprimidos -zip, zlib, bzip-)

CePETel

Fuentes de Datos

Fuentes Externas



CePETel

Fuentes de Datos

Fuentes Internas

ERP - Enterprise Resource Planning

- Significa “sistema de planificación de recursos empresariales”.
- Estos programas se hacen cargo de distintas operaciones internas de una empresa, desde producción a distribución o incluso recursos humanos.

Las principales ventajas de estos sistemas son:

- Automatización de procesos de la empresa.
- Disponibilidad de la información de la empresa en una misma plataforma.
- Integración de las distintas bases de datos de una compañía en un solo programa.
- Ahorro de tiempo y costes.

- Algunos de los ERP más reconocidos: SAP, JD EDwards, Sage, Microsoft Dynamics ERP.

CePETel

Fuentes de Datos

Fuentes Internas

CRM - Customer Relationship Management

- Las siglas CRM o Customer Relationship Management hacen referencia a un software que permite a las empresas rastrear cada interacción con los clientes, tanto actuales como futuros.

- El objetivo de implementar un CRM es crear un sistema que sus empresas (por lo general, los equipos de ventas y de marketing) puedan usar para interactuar de manera más eficaz y efectiva con los clientes potenciales y actuales.

- Entre los CRMs más conocidos encontramos, T3 CRM, Salesforce, Microsoft Dynamics CRM, Oracle CRM On Demand.

CePETel

Fuentes de Datos

Fuentes Internas

SCM - Supply Chain Management

- Las siglas **SCM** (gestión de la cadena de suministro, del inglés **Supply Chain Management**) se refiere a las herramientas y métodos cuyo propósito es mejorar y automatizar el suministro a través de la reducción de los stocks y los plazos de entrega.

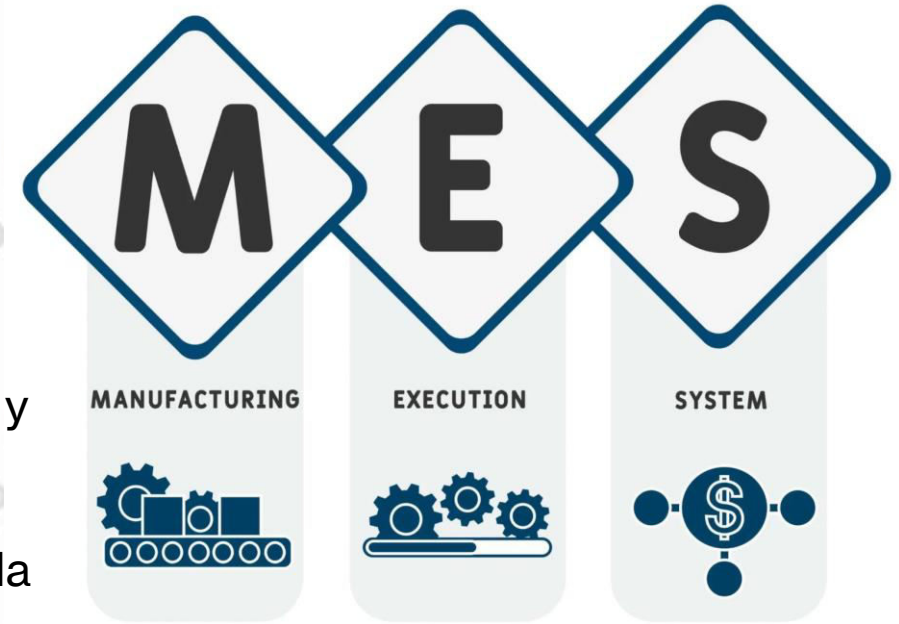
- Incluye la planificación de las actividades de **suministro, fabricación y distribución de cada producto.** Incluye la oferta y demanda dentro y fuera de la empresa.

- Algunos de los SCM más reconocidos, Oracle Logistics Solution, SAP SCM, Microsoft Supply Chain Management.

Fuentes de Datos

Fuentes Internas

MES - Manufacturing Execution System

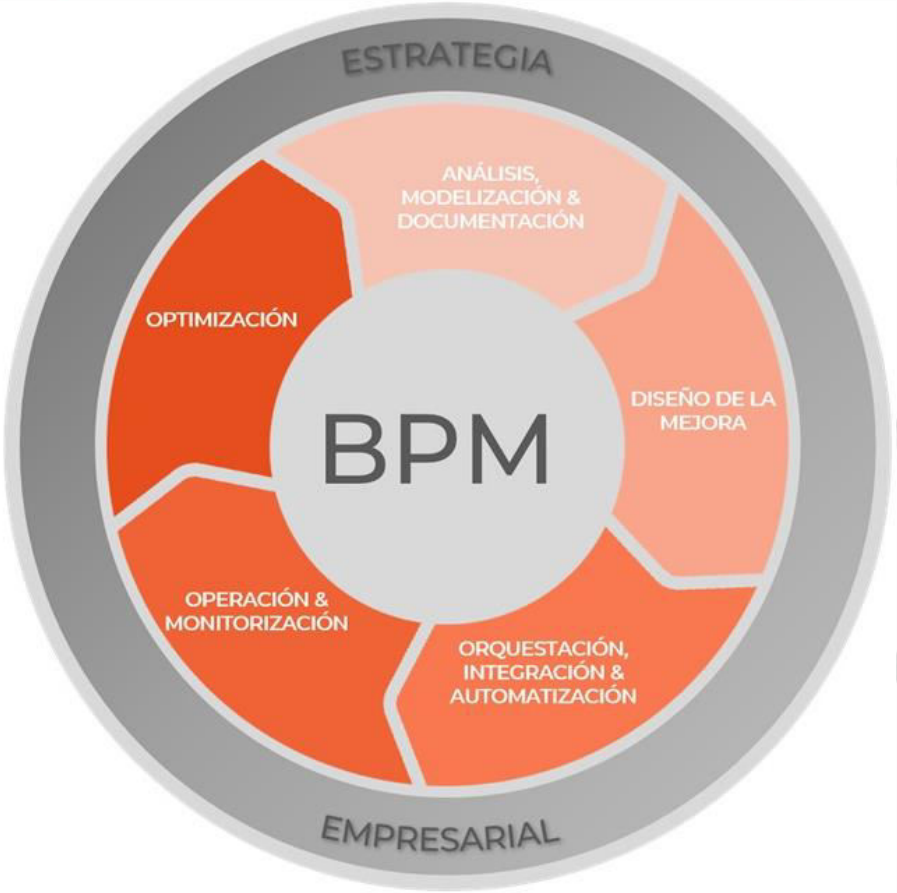


- Es un sistema enfocado al Control de la Producción, que monitoriza y documenta la gestión de la planta.
- El propósito último de un Sistema Mes es aumentar la Eficiencia de la Planta de Producción:
    - Reduciendo Costes
    - Mejorando la Productividad
    - Aumentando la Trazabilidad y la Calidad entregada a tu cliente.
- Algunos de los MES más reconocidos, SAP Manufacturing Execution, Oracle Manufacturing, Microsoft Dynamics Inventory Management.

Fuentes de Datos

Fuentes Internas

BPM - Business Process Management



- Los BPMs son un software empresarial que permite a las empresas modelizar, implementar y ejecutar conjuntos de actividades interrelacionadas –es decir, Procesos– de cualquier naturaleza, ya sea dentro de un departamento o a través de toda la organización.

- Cuentan con extensiones para incluir a los clientes, proveedores y otros agentes como participantes en las tareas de los procesos.

- Algunos BPMs de Mercado, JBPM, Microsoft BPM, Oracle BPM, SAP BPM, RedHat JBoss BPM.

CePETel

Fuentes de Datos

Formatos

Texto / CSV

- Comúnmente usados para **intercambiar** datos **Sistemas**.
- Legibles y parseables.
- **No soportan compresión** de bloques.
- No almacenar header ni footer (**no metadata**). Se debe saber que es cada campo.
- La estructura depende del orden de los campos. Nuevos campos deben ser agregados al final y los existentes no pueden borrarse. **Soporte limitado para evolución de esquema.**

JSON

- **JSON** - Java Script Object Notation.
- Generalmente utilizados como entradas o salidas para **API Rest**.
- Un JSON por cada línea.
- Almacena la **metadata junto con los datos, permitiendo la evolución del esquema.**
- **No soportan compresión** de bloques.
- Muy utilizado por Bases de Datos NoSQL como **MongoDB**.

CePETel

Fuentes de Datos

Formatos

AVRO

- **Formato de almacenamiento multipropósito**.
- Almacena la **metadata junto con los datos**. También **permite** especificar un **esquema independiente en la lectura** del archivo. Esto lo hace el ejemplo perfecto de evolución de esquema, ya que se pueden agregar, renombrar, eliminar y cambiar el tipo de dato de los campos del archivo definiendo un nuevo esquema independiente.
- **Soportan compresión** de bloques.

XML

- XML - **Extensible Markup Language**.
- Almacena **la metadata junto con los datos**.
- Es un **lenguaje multiplataforma** diseñado para almacenar varios tipos de datos.
- Son fácilmente modificables.
- **No soportan compresión de bloques**.
- En las Api/Rest están siendo reemplazados por archivos **JSON** cómo estándar.

Tipos de Datos

Tipo Numérico

- **Enteros**
  - Smallint
  - Integer
  - Bigint
- **Punto Flotante**
  - real
  - double precision

- **Precisión arbitraria**
  - Numeric(p,s)
  - Numeric(p)
  - Numeric
- **Serial / Autoincrementales**
  - Smallserial
  - Serial
  - Bigserial

CePETel

Tipos de Datos

Tipo Numérico

Nombre	Tamaño	Descripción	Rango
smallint	2 bytes	small-range integer	-32768 to +32767
integer	4 bytes	typical choice for integer	-2147483648 to +2147483647
bigint	8 bytes	large-range integer	-9223372036854775808 to +9223372036854775807
decimal	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
numeric	variable	user-specified precision, exact	up to 131072 digits before the decimal point; up to 16383 digits after the decimal point
real	4 bytes	variable-precision, inexact	6 decimal digits precision
double precision	8 bytes	variable-precision, inexact	15 decimal digits precision
smallserial	2 bytes	small autoincrementing integer	1 to 32767
serial	4 bytes	autoincrementing integer	1 to 2147483647
bigserial	8 bytes	large autoincrementing integer	1 to 9223372036854775807

CePETel

Tipos de Datos

Tipo Caracter

- **Longitud Fija**
  - character(n)
  - char(n)
- **Longitud Variable con límite**
  - character varying(n)
  - varchar(n)
- **Longitud variable sin límite**
  - text

Tipo Monetario

- money(n)

Nombre	Tamaño	Descripción	Rango
money	8 bytes	currency amount	-92233720368547758.08 to +92233720368547758.07

CePETel

Tipos de Datos

Tipo Fecha/Tiempo

- **Timestamp**
  - timestamp (sin time zone)
  - timestamp (con time zone)
- **Fecha**
  - date
- **Tiempo**
  - time (sin time zone)
  - time (con time zone)
- **Intervalo**
  - Interval

Nombre	Tamaño	Descripción	Valor Min.	Valor Max.	Resolución
timestamp [(*p*)] [without time zone]	8 bytes	both date and time (no time zone)	4713 BC	294276 AD	1 microsecond / 14 digits
timestamp [(*p*)] with time zone	8 bytes	both date and time, with time zone	4713 BC	294276 AD	1 microsecond / 14 digits
date	4 bytes	date (no time of day)	4713 BC	5874897 AD	1 day
time [(*p*)] [without time zone]	8 bytes	time of day (no date)	00:00:00	24:00:00	1 microsecond / 14 digits
time [(*p*)] with time zone	12 bytes	times of day only, with time zone	00:00:00+1559	24:00:00-1559	1 microsecond / 14 digits
interval [*fields*] [(*p*)]	16 bytes	time interval	-178000000 years	178000000 years	1 microsecond / 14 digits

CePETel

Tipos de Datos

Tipo Booleano

- Booleano
  - boolean

Nombre	Tamaño	Descripción
boolean	1 byte	state of true or false

Tipo Binario

- Binarios
  - bytea

Nombre	Tamaño	Descripción
bytea	1 or 4 bytes plus the actual binary string	variable-length binary string

CePETel

Persistencia Políglota y Teorema de Cap

Bases de Datos NoSQL

Persistencia Políglota

Diferentes tecnologías de bases de datos para resolver diferentes problemas desde una misma aplicación.

Búsquedas performantes sobre Catálogo de productos

Información distribuida Geográficamente
Profile de Usuarios y Documentos de Productos con Info No Estructurada

Caché de Sesiones
Lockeos Distribuidos

Transacciones Económicas

CePETel

Bases de Datos NoSQL

¿Qué es el teorema CAP y cómo elegir la base de datos para tu proyecto?

Cuando hablamos de almacenar y gestionar información hay varios conceptos que se van haciendo cada vez más relevantes y que es indispensable que tengan en cuenta a la hora de **seleccionar el tipo de base de datos que van a usar,** estos conceptos son parte de lo que conocemos como **Requisitos de Calidad o requerimientos no funcionales.**

Los requerimientos no funcionales son aquellos atributos inherentes a la operación del sistema y de su comportamiento. Este tipo de atributos es importante definirlos desde el comienzo del desarrollo del proyecto porque es a partir de ellos que se toman las decisiones más costosas del proyecto.

> Por ejemplo: ¿vas a desarrollar un proyecto en el que la tasa de crecimiento de usuarios es del 50% mensual o es del 5% anual?, necesitas que tu aplicación soporte transaccionalidad o es solo un sitio de consulta?, este tipo de preguntas definen el rumbo de la arquitectura de tu proyecto.

CePETel

Bases de Datos NoSQL

Ahora...*¿qué tiene esto que ver con la selección de la base de datos que vas a usar en tu proyecto?*

Vamos a hablar de tres atributos de calidad que servirán de base para la selección de la base de datos que van a elegir:

- **Consistencia:** La consistencia nos garantiza que una lectura nos retornará la escritura más reciente de un registro dado. Lo que esto implica es que siempre que se haga alguna modificación a un dato dicho cambio debe reflejarse en todos los nodos de la base de datos, esto garantiza que siempre que se acceda a la información cualquiera de los nodos puede responder y la información siempre será la misma.

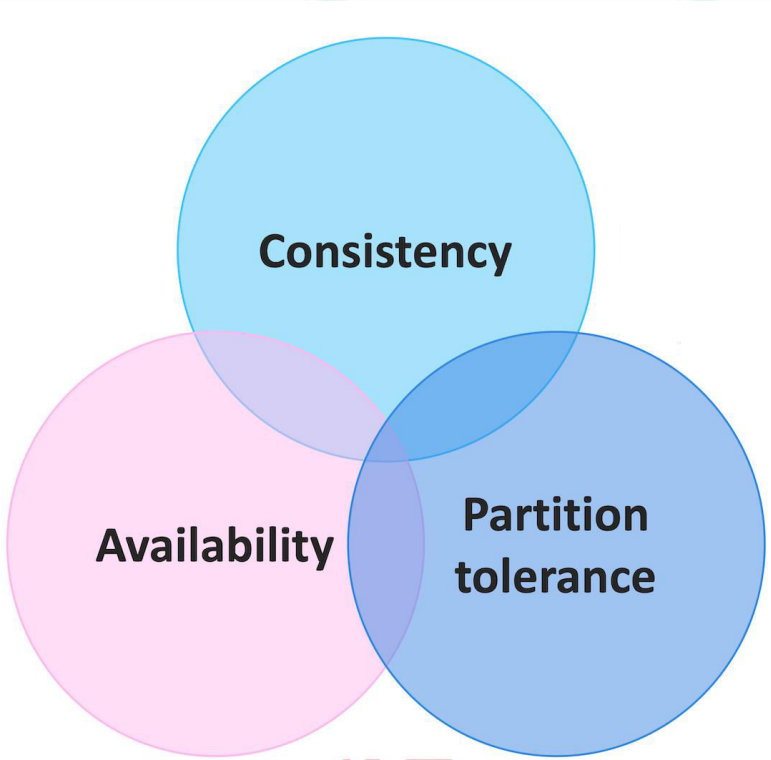
- **Disponibilidad:** Un nodo en funcionamiento nos debe retornar una respuesta razonable en un periodo razonable de tiempo (Ni error ni timeout).

- **Tolerancia al Particionamiento:** El sistema nos debe seguir funcionando, aunque algunos nodos no se encuentren disponibles ya que la información es consistente en todos los nodos.

Bases de Datos NoSQL

Teorema CAP

- Fue desarrollado más como una conjetura que como un teorema por el computador científico **Eric Brewer** de la Universidad de California, Berkeley.
- En el año 2002, **Seth Gilbert y Nancy Lynch** del MIT publicaron una **prueba formal** de la conjetura de Brewer, transformándolo en un teorema.
- **Brewer** indicaba que es imposible en un sistema computacional distribuido, proveer simultáneamente las tres propiedades expresadas:
  - *Consistency*,
  - *Availability* y
  - *Partition Tolerance*.

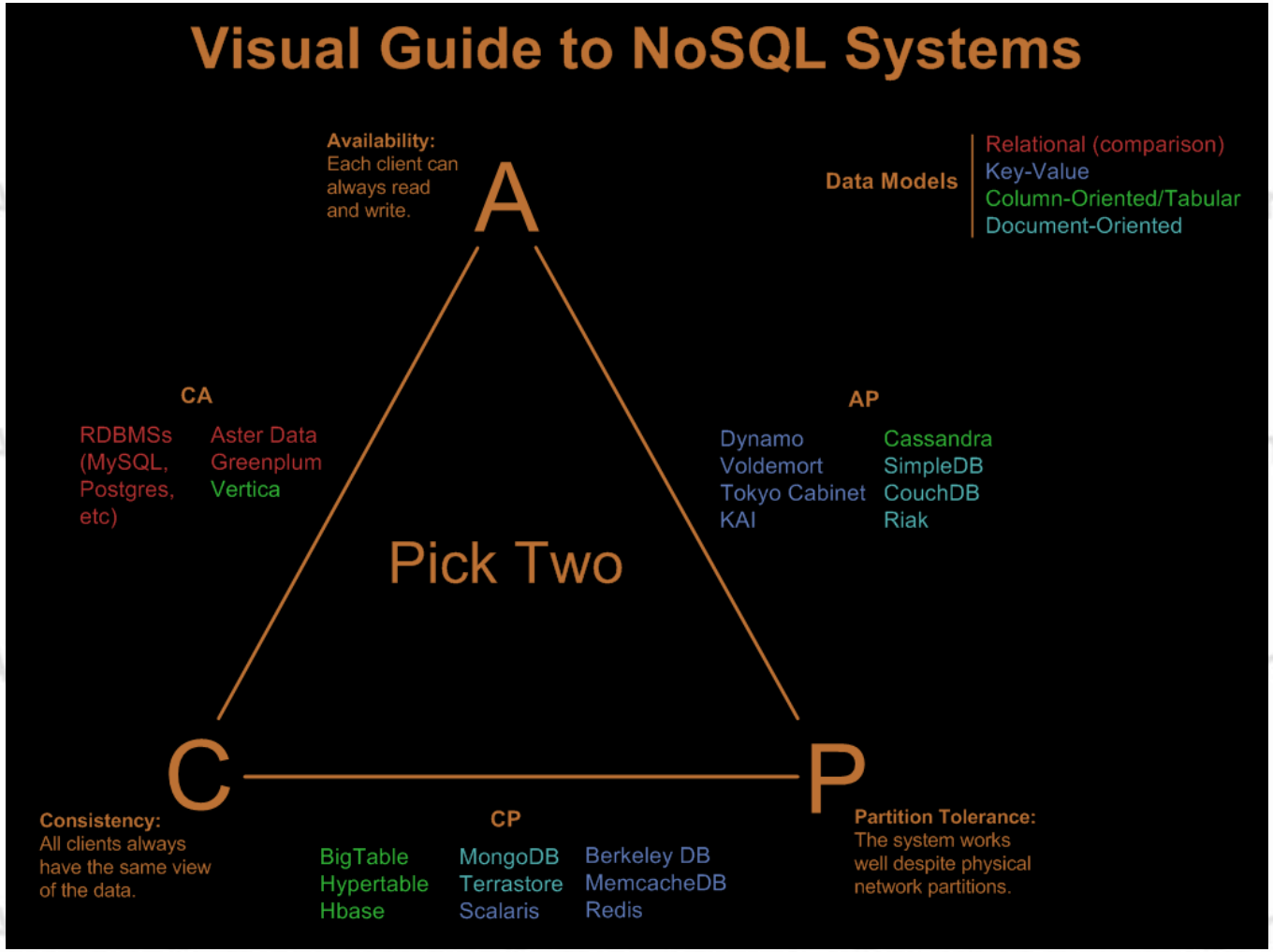


Bases de Datos NoSQL

Teorema CAP

En Ciencias de la Computación se enunció un teorema que nos dice que en un sistema distribuido de almacenamiento de datos no podemos garantizar consistencia y disponiblidad (para actualizaciones), al tiempo cuando el sistema sufre una partición (queda separado en dos o más islas). Este es el Teorema CAP, por ***C**onsistency* (Consistencia), ***A**vailability* (Disponibilidad) y ***P**artition Tolerance* (Tolerancia al Particionamiento).

Debemos tener en cuenta las exigencias de nuestro proyecto para saber qué atributos de calidad necesitamos y así elegir el tipo de base de datos que necesitaremos.



Bases de Datos NoSQL

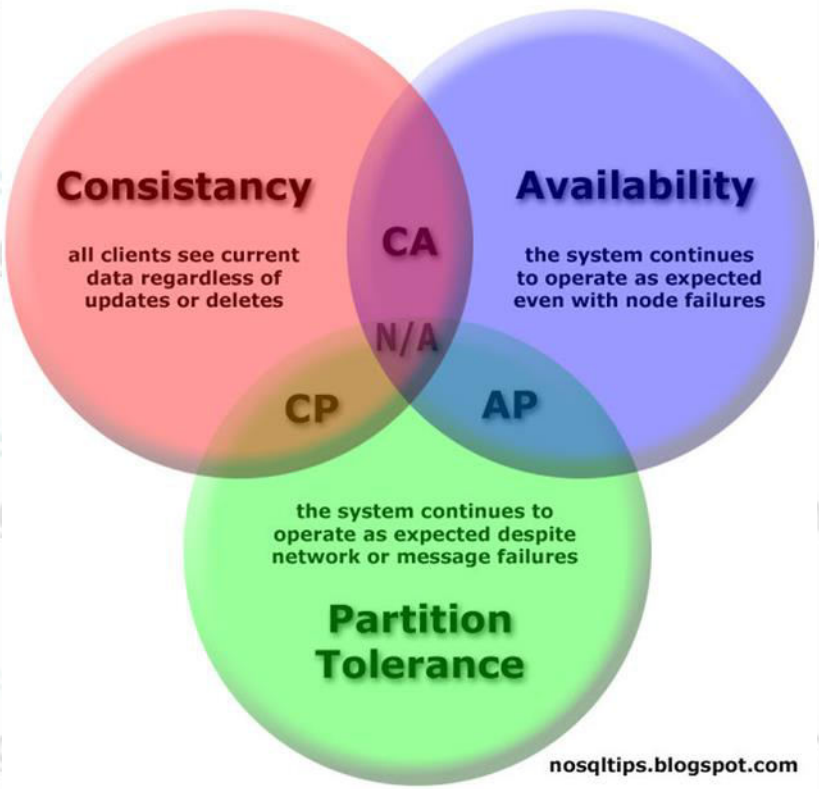
Teorema CAP

Solo puedes garantizar dos de estos tres atributos,

CP (Consistencia y Tolerancia al particionamiento): Esto quiere decir que no se garantiza la disponibilidad, hay clientes que por ejemplo requieren que el sistema esté disponible 100% del tiempo o muy cerca, con bases de datos que cumplan con **CP** no es posible garantizar esto, no quiero decir que no se pueda lograr en cierto nivel, pero el sistema está enfocado en aplicar los cambios de forma consistente, aunque se pierda comunicación con algunos nodos.

AP (Disponibilidad y Tolerancia al particionamiento): En este caso no se garantiza que los datos sean iguales en todos los nodos todo el tiempo, en este caso el sistema siempre estará disponible para las peticiones, aunque se pierda la comunicación entre los nodos.

CA (Consistencia y disponibilidad): En este caso no se puede permitir el particionado de los datos, porque se garantiza que los datos siempre son iguales y el sistema estará disponible respondiendo todas las peticiones.

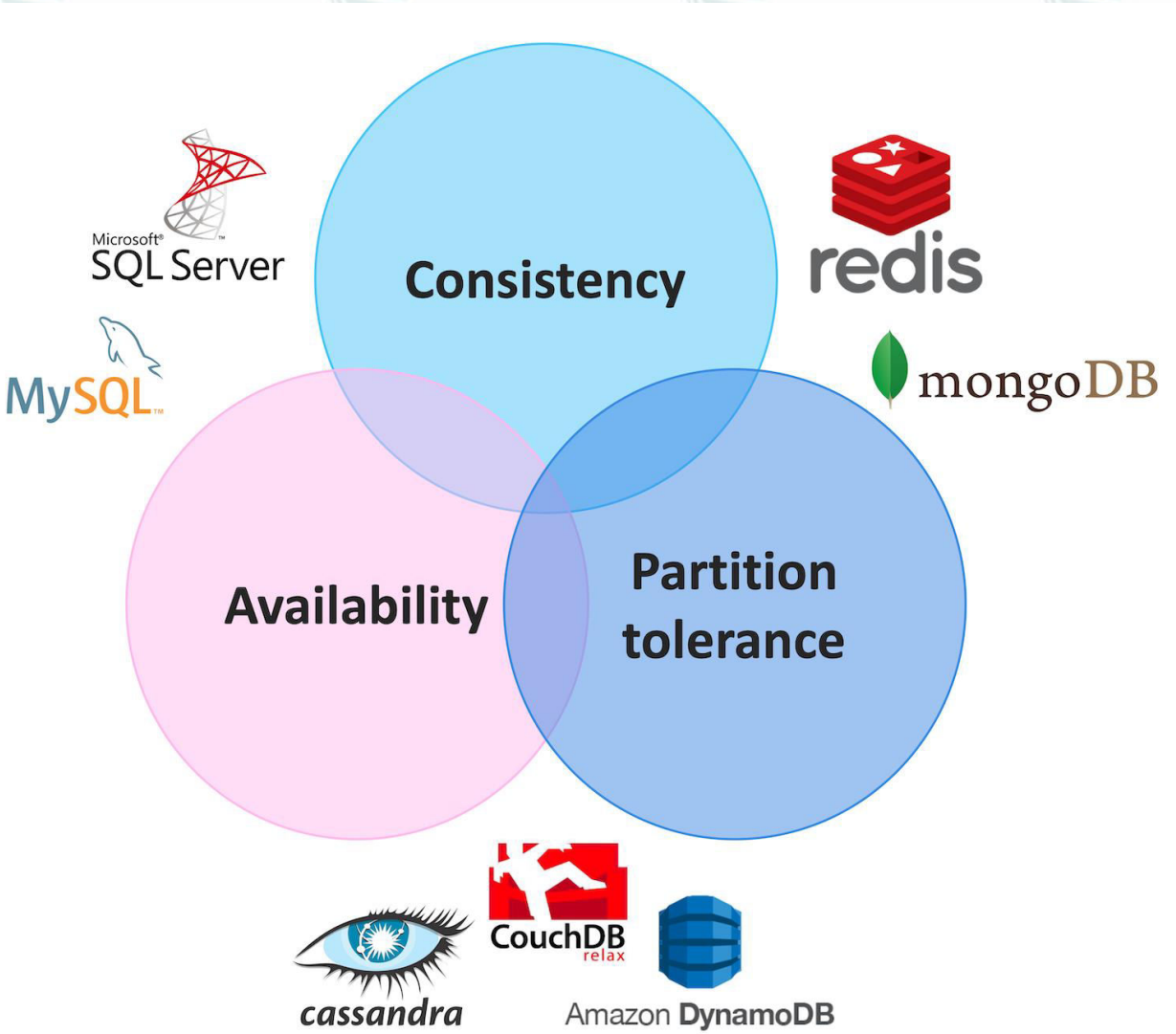


Bases de Datos NoSQL

Teorema CAP

Por ejemplo, los sistemas de bases de datos relacionales (SQL de toda la vida) son CA porque todas las escrituras y lecturas se hacen sobre la misma copia de los datos.

Si en sus proyectos lo que requieren es que todos sus clientes tengan acceso a la misma vista de los datos en cualquier momento que sean requeridos (Consistencia) y además desde el comienzo sabes que se va a requerir disponibilidad de los datos en cercana al 100% del tiempo (Tolerancia al Particionamiento), el tipo de base de datos que deberían usar es una como MongoDB o Redis.



TP N° 1:
Conociendo el entorno Postgres

Para descargar e instalar Postgres SQL Link

Video Explicativo de instalación Postgres

Para acceder a la página de Postgres LINK con información completa.

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

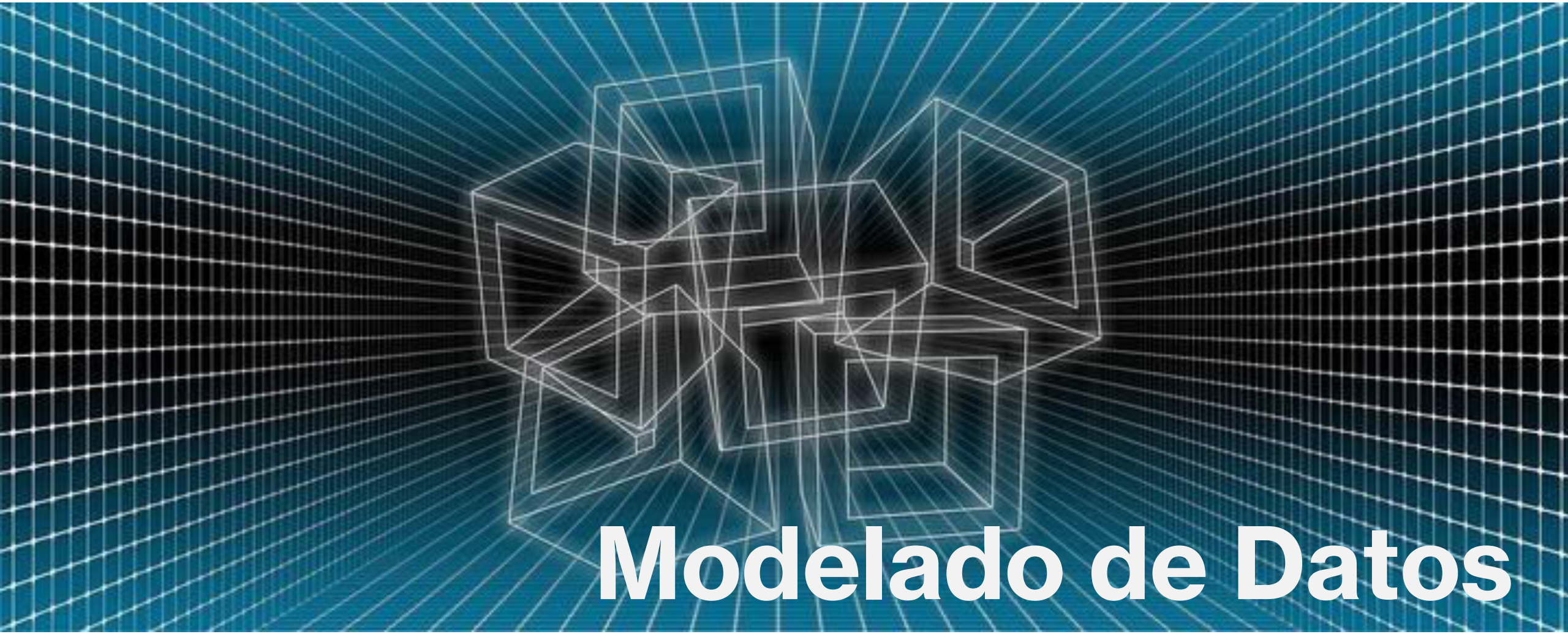
TEMARIO

Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 2: Modelado de Datos y Normalización

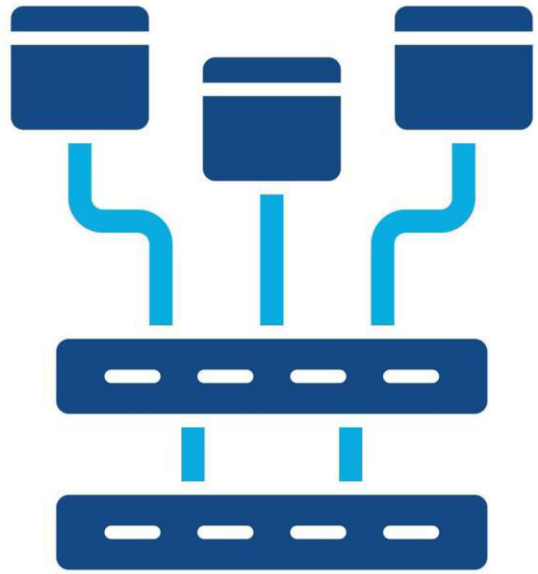


Modelado de Datos

Modelado de Datos

En la mayoría de los paradigmas se separan

- las abstracciones de datos que representan **información** y
- los **procesos**: abstracciones de control del flujo que manipulan esos datos.



En el diseño lógico de datos definimos:

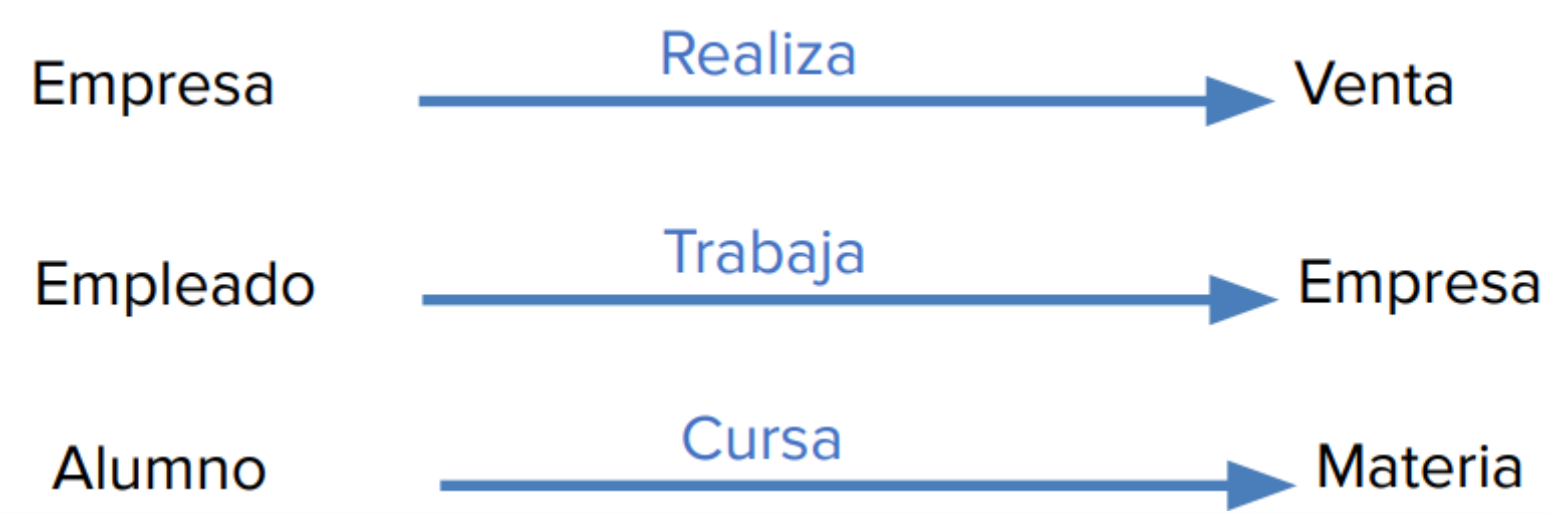
- ✓ las **entidades** que representan partes de un sistema y
- ✓ sus **relaciones**.

CePETel

Modelado de Datos

¿Qué son las Relaciones?

- Representan asociaciones del mundo real entre entidades.
- Se puede representar con un verbo o preposición que conecta dos entidades.

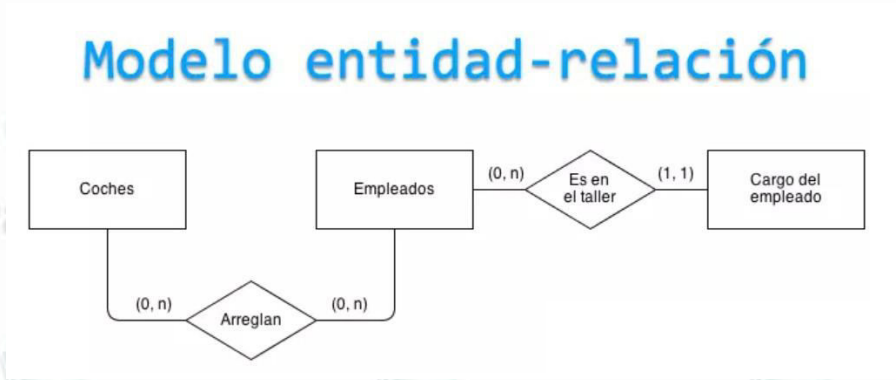


CePETel

Modelado de Datos

¿Cómo podemos **comunicar** nuestro Modelo de Datos?

- Prosa, poca y muy puntual.
- Diagrama de Entidad Relación DER.
- Código SQL – DDL Data Definition Language. (Metadata)(*)



CePETel

Modelado de Datos

Modelo conceptual

El modelo conceptual define las entidades y sus relaciones en base al negocio (es fácil de validar por el usuario). Es independiente de la tecnología.

Ejemplos

- un cliente tiene muchas sucursales,
- cada sucursal está ubicada en un domicilio.
- una materia de una facultad puede tener 0, 1 o muchas correlativas.

Diseño lógico

Aquí definimos el esquema en el cual vamos a trabajar sin depender de una implementación física determinada. Ej: trabajamos en un esquema relacional sin decidir si lo implementaremos en Oracle, MySQL o SQLServer.

CePETel

Modelado de Datos

Modelo conceptual

El modelo conceptual define las entidades y sus relaciones en base al negocio (es fácil de validar por el usuario). Es independiente de la tecnología.

Diseño físico

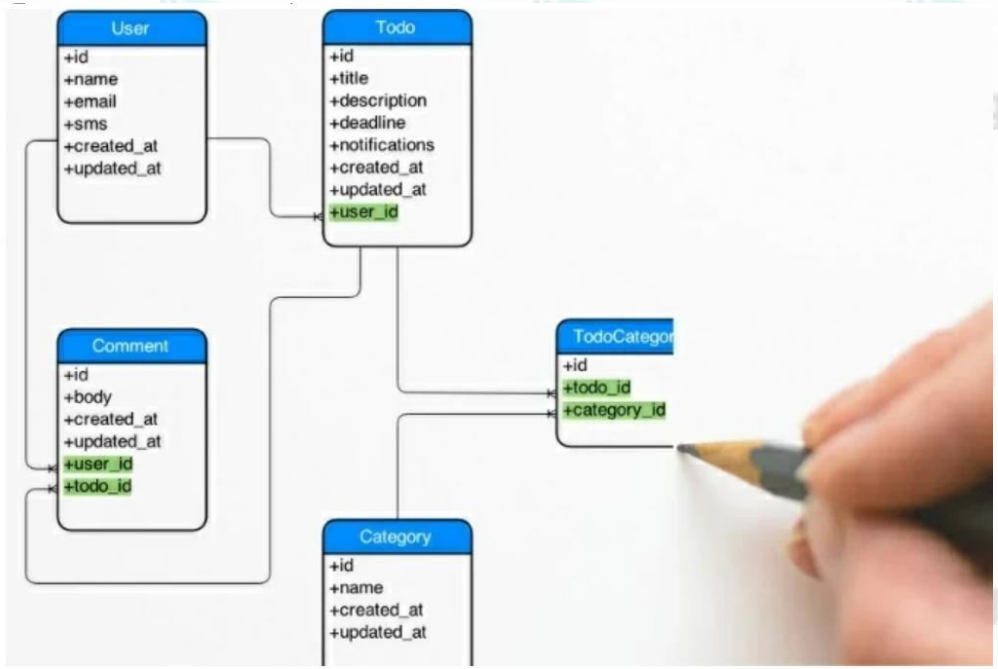
El diseño físico se implementa en un motor de base de datos que, además de soportar un determinado esquema, impone ciertas restricciones de la implementación, como el lenguaje de manipulación de datos (DML Data Manipulation Language) o el que nos permite crear las entidades (DDL - Data Definition Language).

El estándar para el modelo relacional es el SQL aunque cada DBMS implementa su propio subconjunto de instrucciones.

Modelado de Datos

DER – Diagrama de Entidad Relación

- **Entidad**: cualquier cosa que tenga relevancia para nuestro sistema. Los proyectos, las tareas, los impuestos, las complejidades...
- **Atributos**: son las propiedades o características que describen una entidad. Un cliente tiene nombre y cuit, un auto tiene modelo, marca y color, etc.
- **Atributos multivaluados**: son atributos que pueden tomar más de un valor (direcciones de mail de un empleado, subordinados de un jefe, entre otros)



CePETel

Modelado de Datos

DER – Diagrama de Entidad Relación

- **Instancia de una entidad:** Ocurrencia particular de una entidad.

- **Relaciones entre entidades:**
  - Representan asociaciones del mundo real entre entidades
  - Se puede representar con un verbo o preposición que conecta dos entidades.

Modelado de Datos

Características de las Relaciones

- **Grado**
  - Unarias o Recursivas: Son relaciones que asocian a una misma entidad.
  - Binarias: Relaciones que asocian a dos entidades.
  - N -arias: Relaciones que asocian a N entidades.

TIPO	RELACIÓN	REPRESENTACIÓN
1:1	Uno a uno: La cardinalidad máxima en ambas direcciones es 1.	1 ◇ 1
1:N	Uno a muchos: La cardinalidad máxima en una dirección es 1 y en la otra muchos.	1 ◇ N
N:M	Muchos a muchos: La cardinalidad máxima en ambas direcciones en muchos.	N ◇ M

- **Cardinalidad**
  - Describe la cantidad de instancias de entidad permitidas en una relación entre dos entidades
  - 1 a 1: un cliente tiene un domicilio y cada domicilio pertenece a un cliente.
  - 1 a n: una empresa tiene muchas sucursales, cada sucursal pertenece a una empresa.
  - n a n: un profesor da clases a muchos alumnos, cada alumno tiene muchos profesores

CePETel

Modelado de Datos

Características de las Relaciones

- **Modalidad**
  - *Relación Mandatoria (Obligatoria):* Si la instancia de una entidad debe existir en la relación. Ej.: Una Factura debe contener a lo sumo un Ítem.
  - *Relación Opcional:* Si la instancia de una entidad no necesita existir en la relación. Ej.: Un cliente puede tener facturas asociadas.

- **Especialización o Generalización**
  - *Entidad Supertipo*:
    - Entidad padre
  - *Entidad Subtipo:*
    - Son las entidades hijas.
    - Son mutuamente excluyentes
    - Contienen distintos atributos

Modelado de Datos

Algunas Notaciones

Notación	Uno a Uno	Uno a Muchos	Muchos a Muchos
Ross	Empleado — Empleado_ext	Cliente → Pedido	Estudiante ↔ Curso
Bachman	Empleado ↔ Empleado_ext	Cliente ↔ Pedido	Estudiante ↔ Curso
Martin (*)	Empleado | — | Empleado_ext	Cliente | — ◇ Pedido	Estudiante ◇ — ◇ Curso
Chen	Empleado 1 — ◇ — 1 Empleado_ext	Cliente 1 — ◇ — N Pedido	Estudiante N — ◇ — M Curso
IDEFX1	Empleado — Empleado_ext	Cliente — ● Pedido	Estudiante ● — ● Curso

Notación	Mandatoria	Opcional
Ross	Pedido → Item	Cliente - - - → Pedido
Bachman	Pedido ↔ Item	Cliente - - - ↔ Pedido
Martin (*)	Pedido | — ◇ Item	Cliente | - - - ◇ Pedido
Chen	Pedido — ◇ — Item	N/A
IDEFX1	Pedido - - ● Item	Cliente - - ● Pedido

(*) Es la notación que utilizaremos. También llamada Crow's foot notation.

CePETel

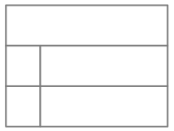
Modelado de Datos

Crow's foot notation

Entity
(with no attributes)

Entity
(with attributes field)

Entity
(attributes field with columns)



Entity
(attributes field with columns and variable number of rows)

Relationships
(Cardinality and Modality)

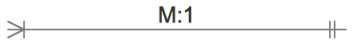
Zero or More

One or More

One and only One

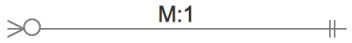
Zero or One

Many - to - One



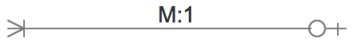
M:1

a one through many notation on one side of a relationship and a one and only one on the other



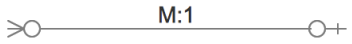
M:1

a zero through many notation on one side of a relationship and a one and only one on the other



M:1

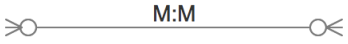
a one through many notation on one side of a relationship and a zero or one notation on the other



M:1

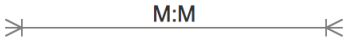
a zero through many notation on one side of a relationship and a zero or one notation on the other

Many-to-Many



M:M

a zero through many on both sides of a relationship



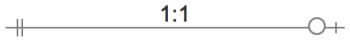
M:M

a one through many on both sides of a relationship

M:M

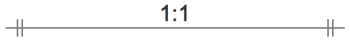
a zero through many on one side and a one through many on the other

Many-to-Many



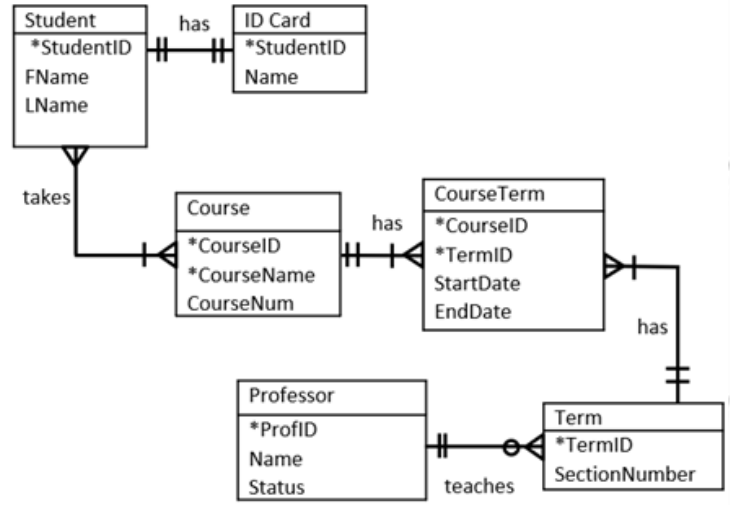
1:1

a one and only one notation on one side of a relationship and a zero or one on the other



1:1

a one and only one notation on both sides



CePETel

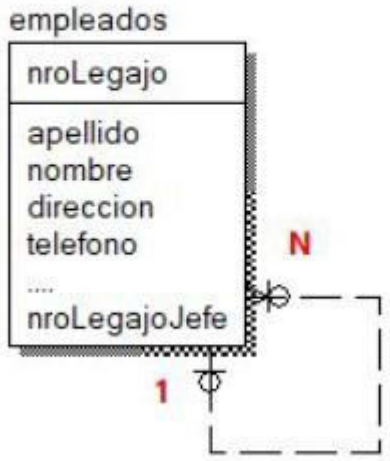
Algunos Ejemplos

Grado de una Relación

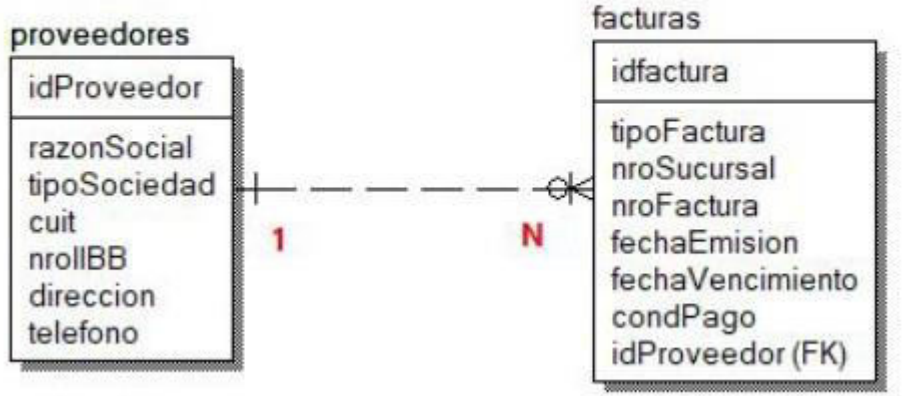
Algunos Ejemplos

Grado de una Relación

Unaria



Binaria

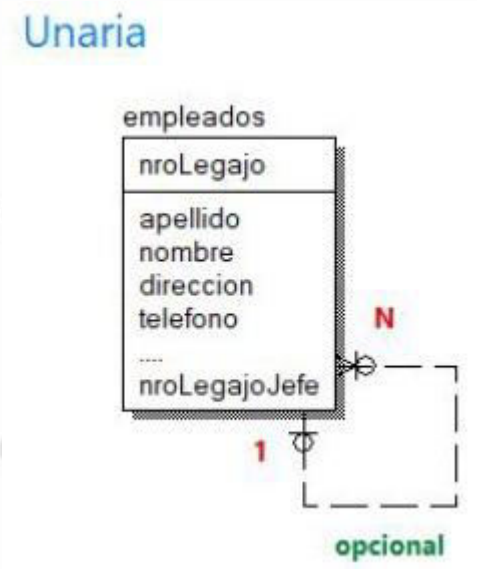


Cardinalidad de una relación

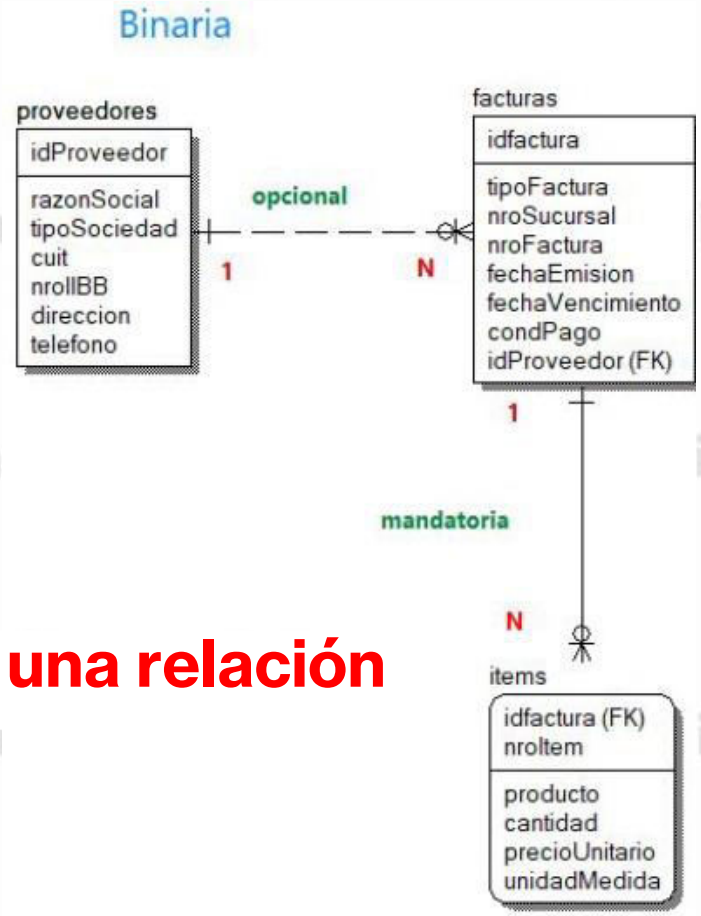
CePETel

Algunos Ejemplos

Grado de una Relación



Modalidad de una relación

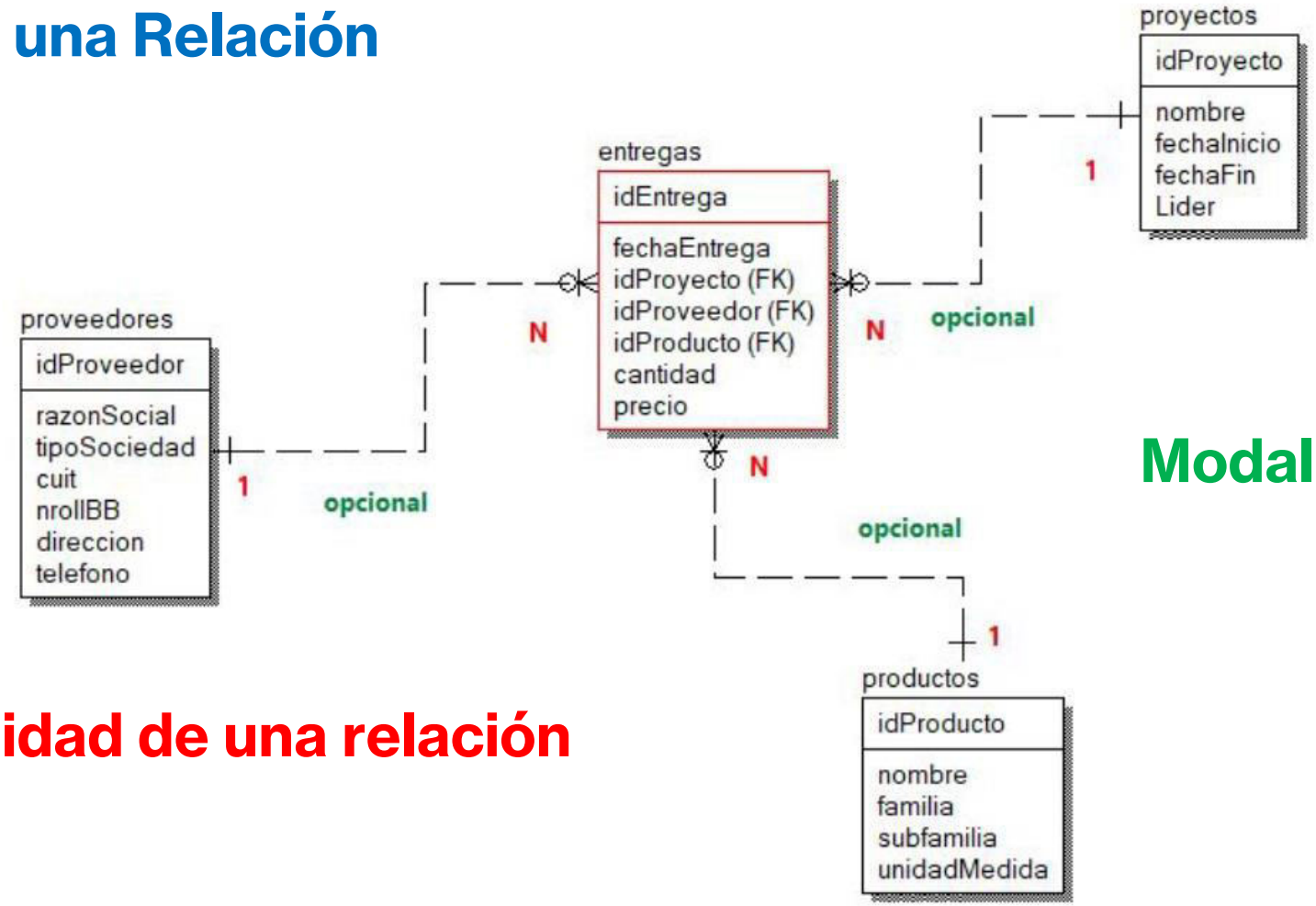


Cardinalidad de una relación

CePETel

Algunos Ejemplos

Grado de una Relación



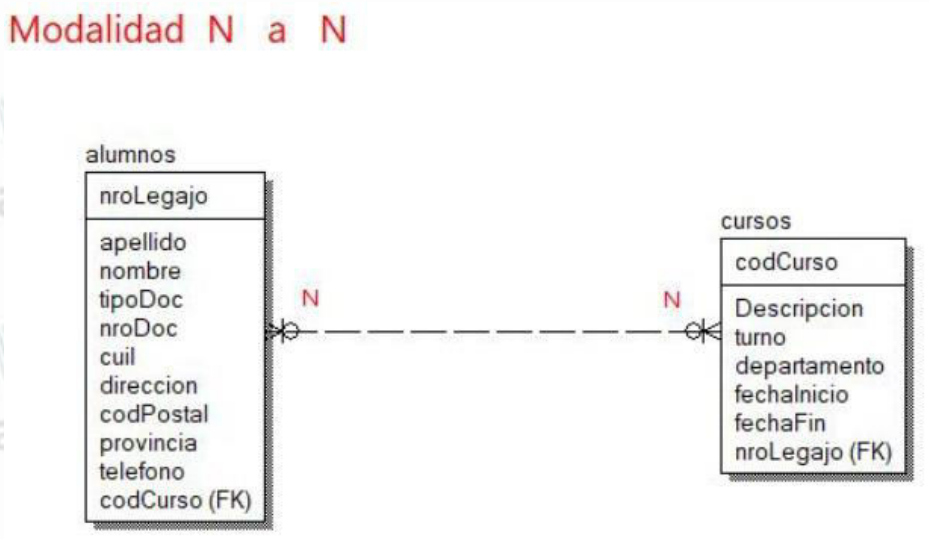
Modalidad de una relación

Cardinalidad de una relación

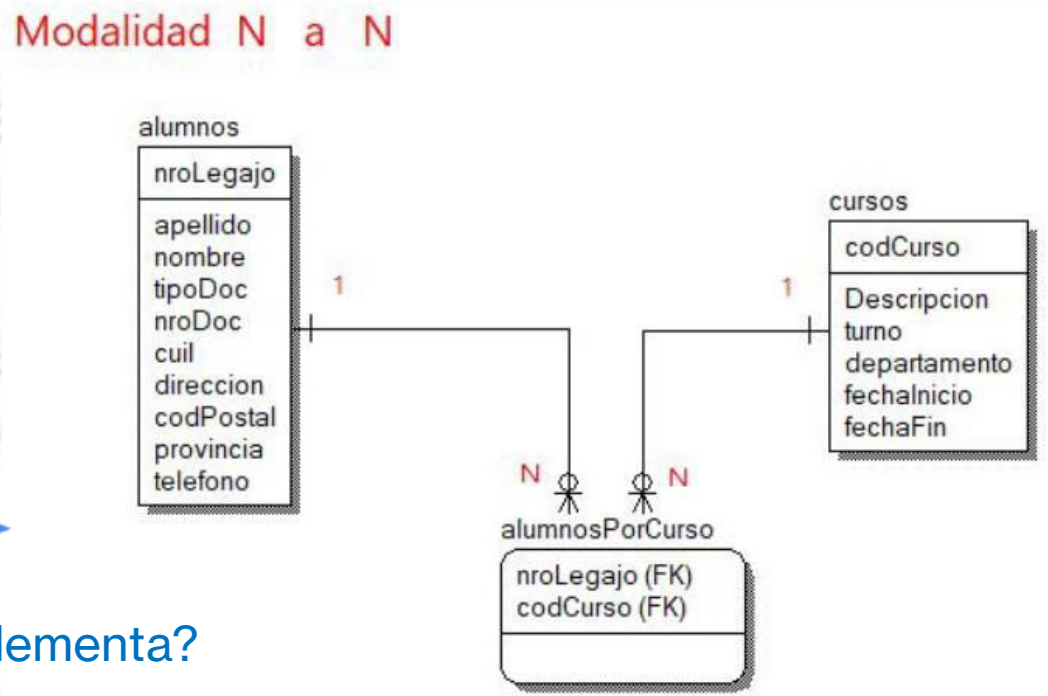
CePETel

Relación N a N

Modelo Lógico



Modelo Físico

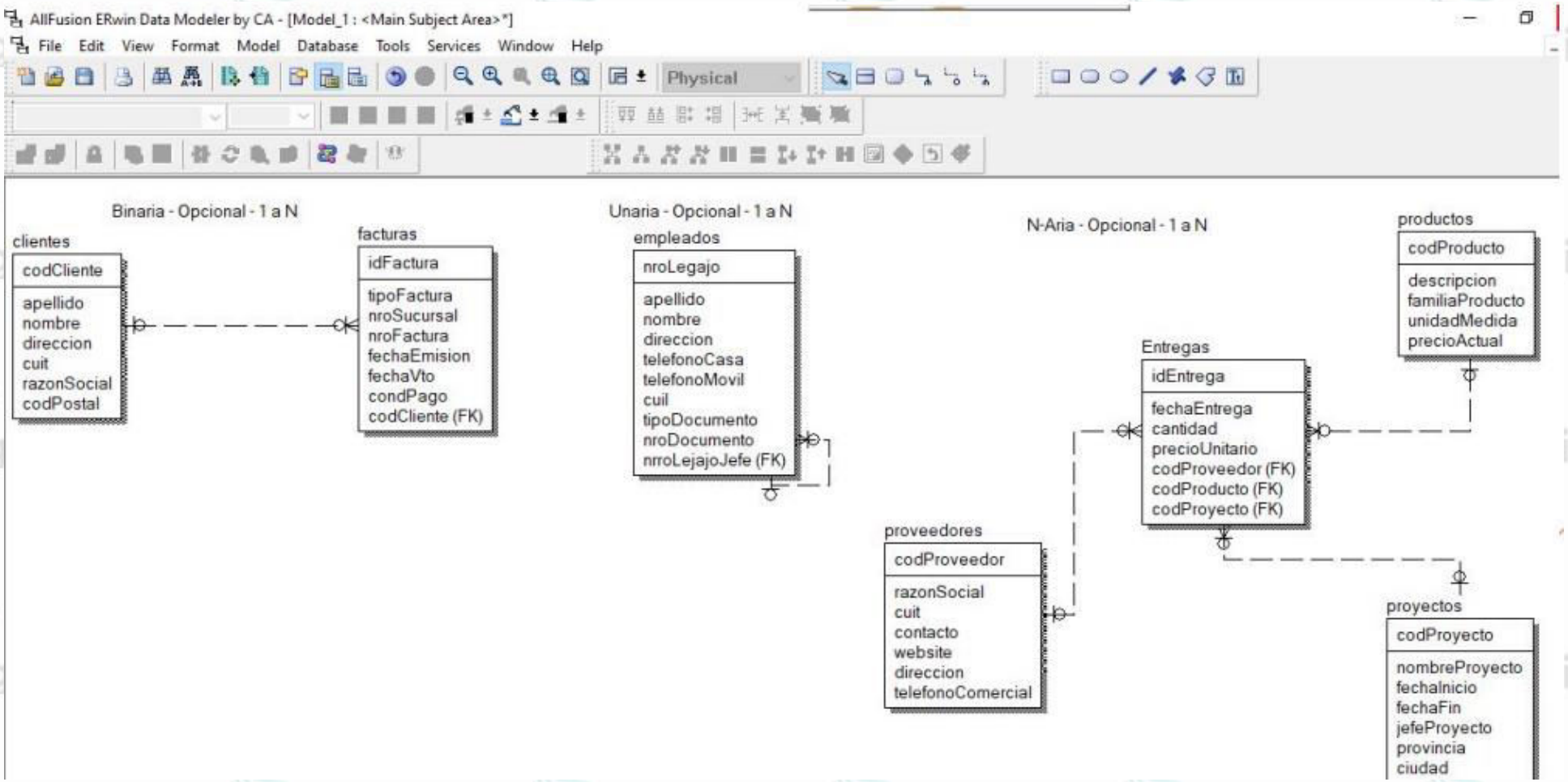


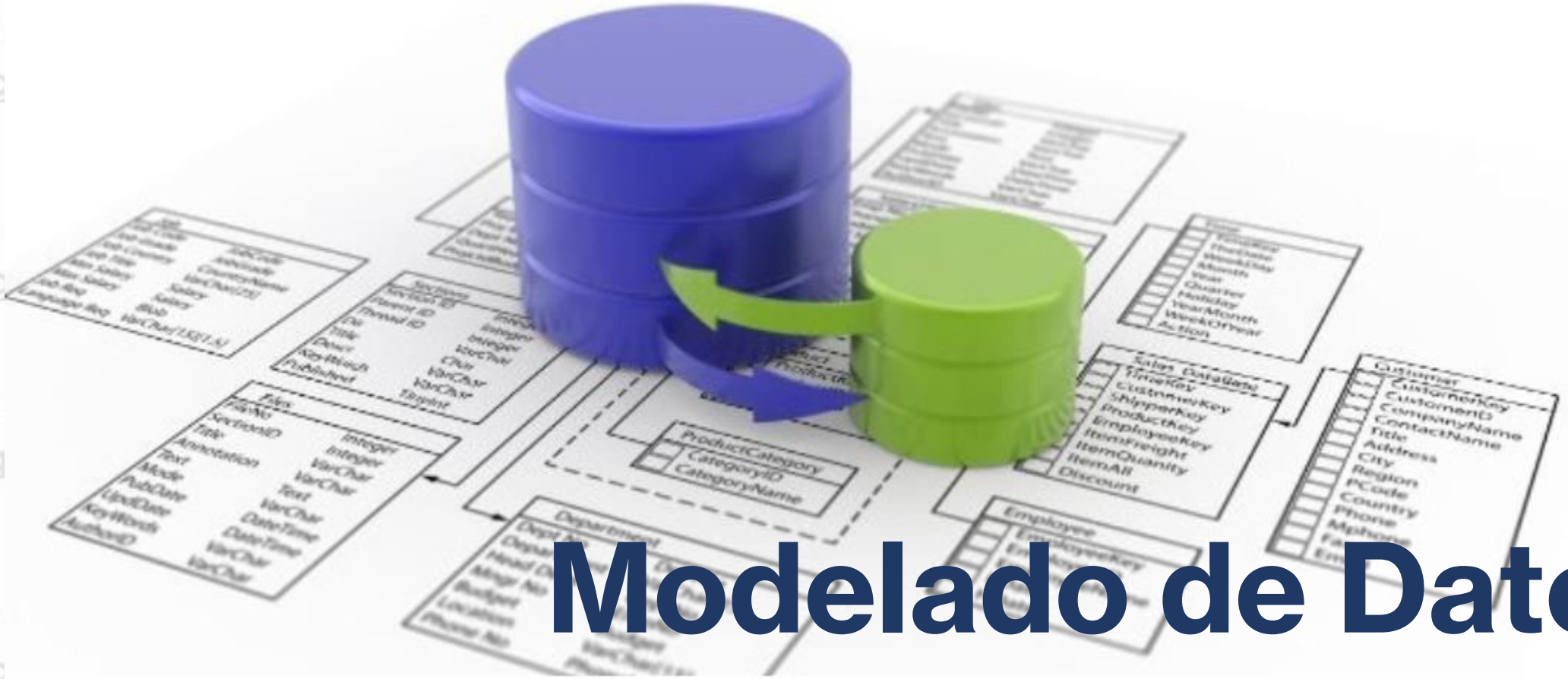
¿Cómo se implementa?

CePETel

Modelado de Datos

Herramientas para Modelado de Datos - CA Erwin





Modelado de Datos

Normalización / Desnormalización

Normalizar una base de datos significa ***transformar*** un conjunto de datos que ***tienen una cierta complejidad en su entendimiento*** y que su ***distribución en el modelo provoca problemas de lógica*** en las acciones de manipulación de datos, en una ***estructura de datos que posea un diseño claro***, donde estos ***datos guarden coherencia y no pierdan su estado de asociación***.

Objetivos de la normalización

- Reducir la redundancia de datos, y, por lo tanto, las inconsistencias.
- Facilitar el mantenimiento de los datos.
- Evitar anomalías en la manipulación de datos.
- Reducir el impacto de los cambios en los datos.

CePETel

Normalización / Desnormalización

Las **formas normales** son heurísticas o criterios que permiten resolver esas redundancias. En algunas bibliografías se habla de errores o inconsistencias, es verdad que las redundancias pueden surgir por errores en el diseño de los datos, pero veremos a continuación que esa redundancia puede ser buscada.

Cada **forma normal** introduce restricciones nuevas, donde la primera restricción que aplica es cumplir la forma normal anterior.

Nosotros para implementar nos vamos a quedar con sólo las primeras 3 Formas Normales.

CePETel

Normalización / Desnormalización

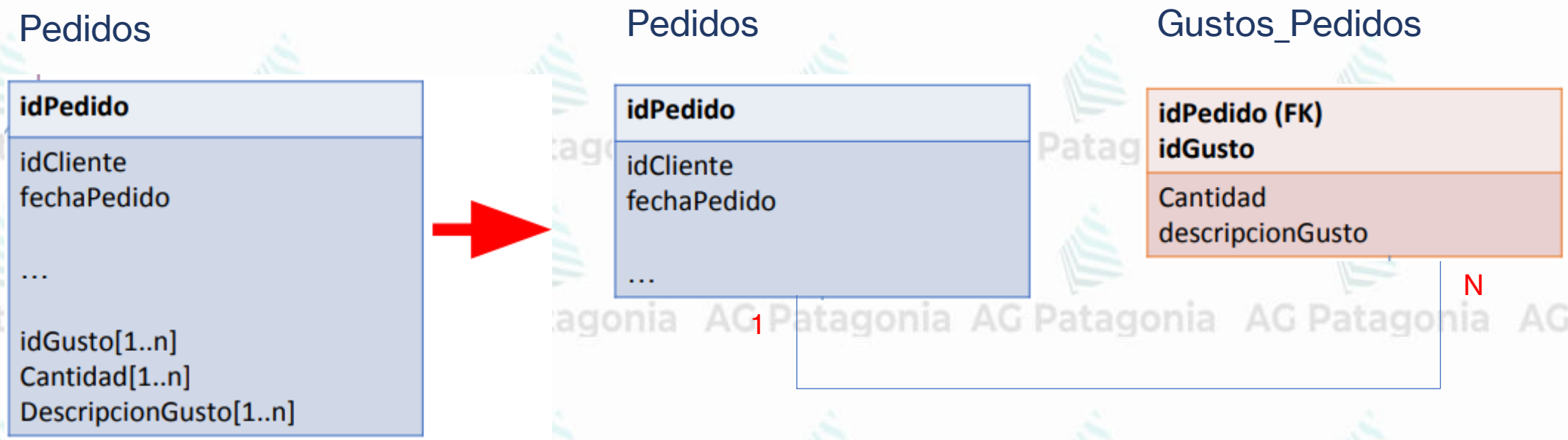
• Cada atributo debe contener valores atómicos. • Cada fila de la misma tabla debe ser única. • Debe prevalecer un crecimiento vertical de los datos y no horizontal. • No deben existir grupos repetidos de datos.	1FN	La característica del modelo relacional, según la cual un atributo debe ser atómico.
• Estar en 1FN. • La relación debe tener una clave principal, de preferencia simple. • Cada atributo de la tabla debe depender totalmente del atributo clave.	2FN	El concepto de dependencias funcionales.
• Debe estar en 2FN. • No deben existir atributos no principales que dependan transitivamente del atributo clave.	3FN	El concepto de dependencias transitivas.

CePETel

Normalización / Desnormalización

Primera Forma Normal

Una entidad que está ***en primera forma normal no puede tener campos repetitivos*** (arrrays, mismo campo repetido n veces), o campos multivaluados.



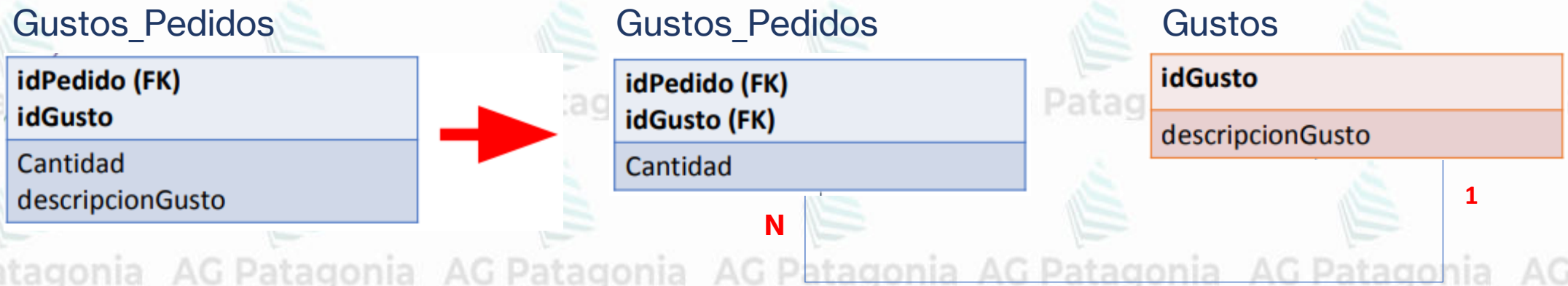
El resultado es una tabla que, si bien está en la primera forma normal, los valores duplicados siguen impidiendo procesar los datos de forma eficiente. Para reducir las redundancias se recomienda llevarla a la segunda forma normal.

CePETel

Normalización / Desnormalización

Segunda Forma Normal

Una estructura de datos está en 2FN si y sólo si ***no hay dependencias funcionales parciales entre un atributo no clave y parte de la clave compuesta*** y se satisface 1FN.



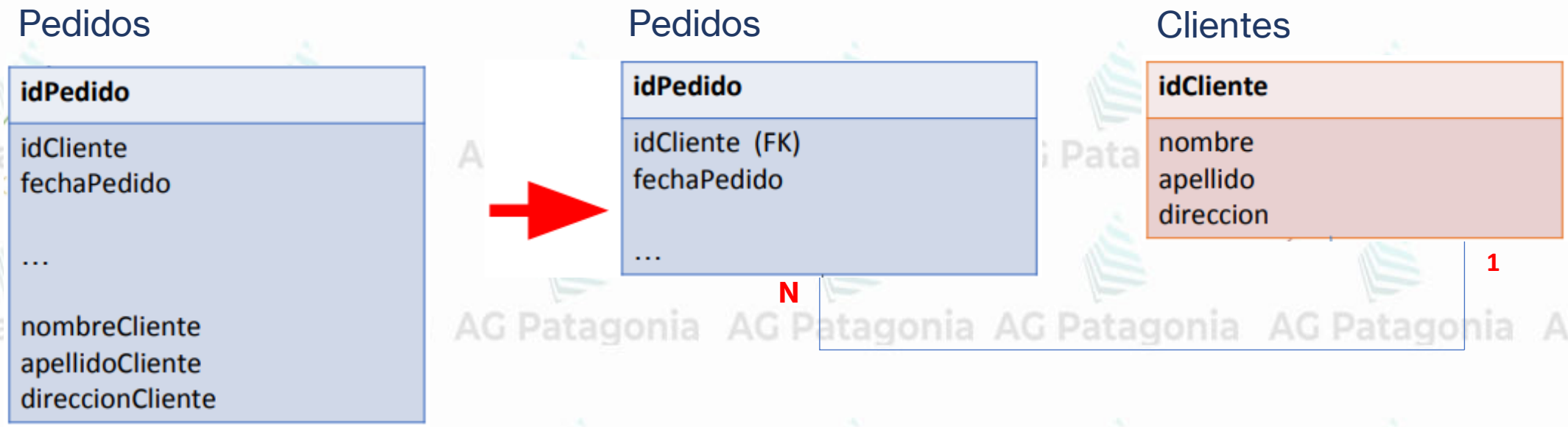
El atributo DescripcionGusto depende funcionalmente solo de idGusto, no tiene dependencia con el idPedido. Por lo que tiene una dependencia funcional parcial de la Clave Primaria.

CePETel

Normalización / Desnormalización

Tercera Forma Normal

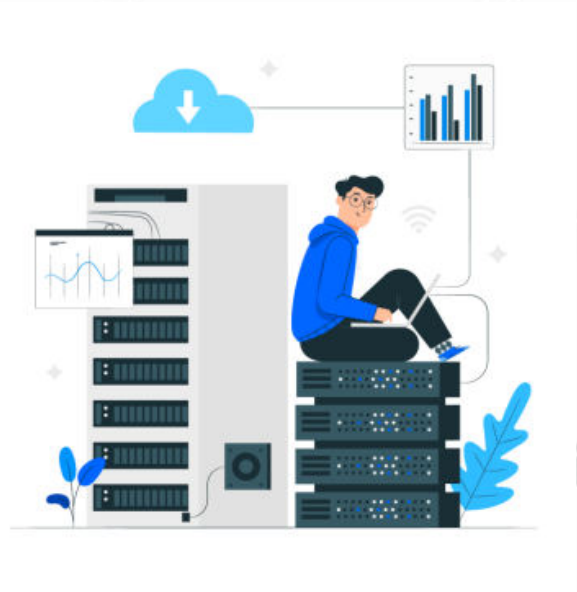
Una estructura de datos está en **3FN** si y sólo si ***no hay dependencias funcionales*** entre los atributos no claves (y se satisface 2FN).



Nota: La conexión por clave foránea o ajena (Foreign Key) permite consultar a dos tablas a la vez. Se habla entonces de un Join.
No siempre es posible evitar por completo los valores duplicados en las bases de datos relacionales.

CePETel

TP N° 2: Modelado de Datos, Normalización



Se quiere diseñar una base de datos relacional que almacene información relativa a los zoologicos existentes en el mundo, así como las especies animales que éstos albergan.

De cada zoo se conoce el nombre, ciudad y país donde se encuentra, tamaño (en m2) y presupuesto anual.

De cada especie animal se almacena el nombre vulgar y nombre científico, familia a la que pertenece y si se encuentra en peligro de extinción.

Además, se debe guardar información sobre cada animal que los zoos poseen, como su número de identificación, especie, sexo, año de nacimiento, país de origen y continente

CePETel

TP N° 2: Modelado de Datos, Normalización

- Armar un diagrama de DER tomando el ejemplo anterior de base de datos, normalizando o desnormalizando en los casos que crea conveniente.

- Genere un Diagrama de Entidad Relación, indicando la cardinalidad y las claves primarias y foráneas. (no se requiere utilizar ningún programa, sino que solo es necesario realizar el DER a mano alzada o con la herramienta que considere)

Zoológicos

Nombre

Ubicación

Tamaño

Presupuesto anual

*Código de zoológico

(Especies)

Especies

*Nombre científico

Familia

Si se encuentra en peligro de extinción

Animal

*Número de identificación

Nombre vulgar

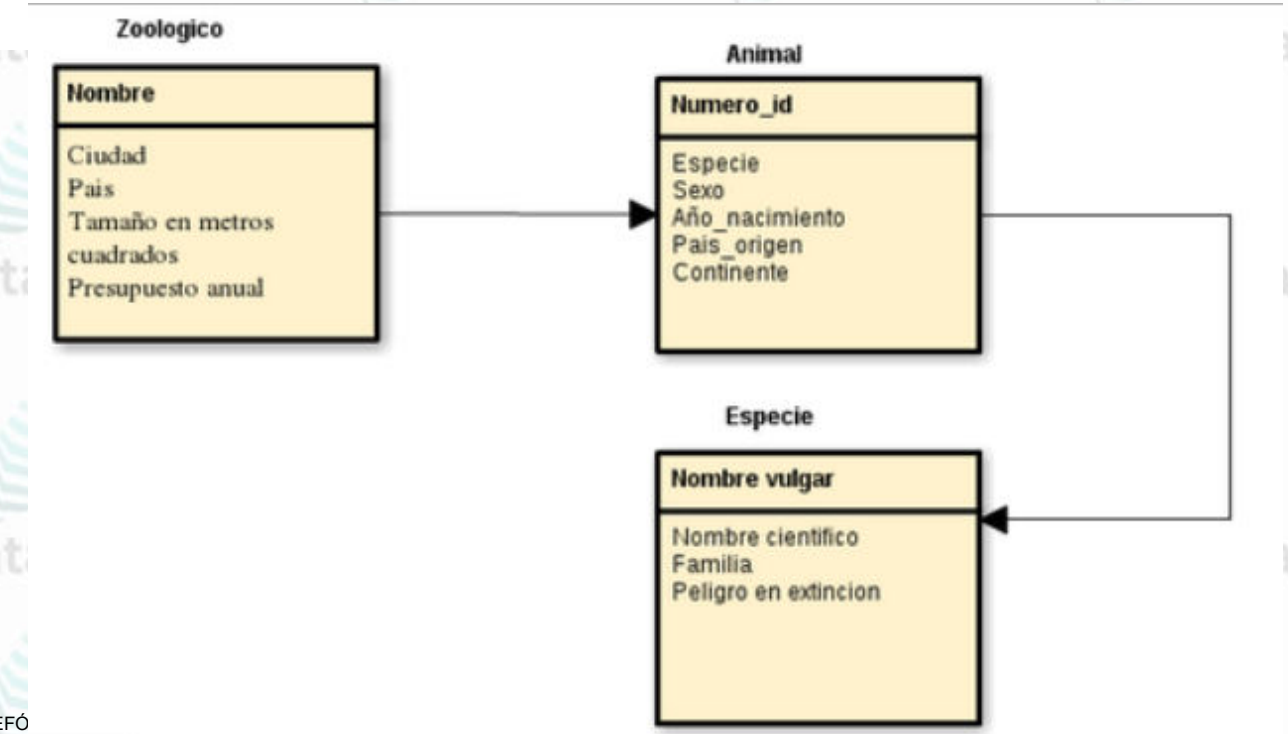
Sexo

Año de nacimiento

País de origen

Continente

Especie



CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de
Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 3: DDL (Data Definition Language)

Historia y Definición

CePETel

SQL

Video: Conociendo la historia SQL

Edgar Codd en los años 70 creó el modelo relacional y el álgebra relacional.

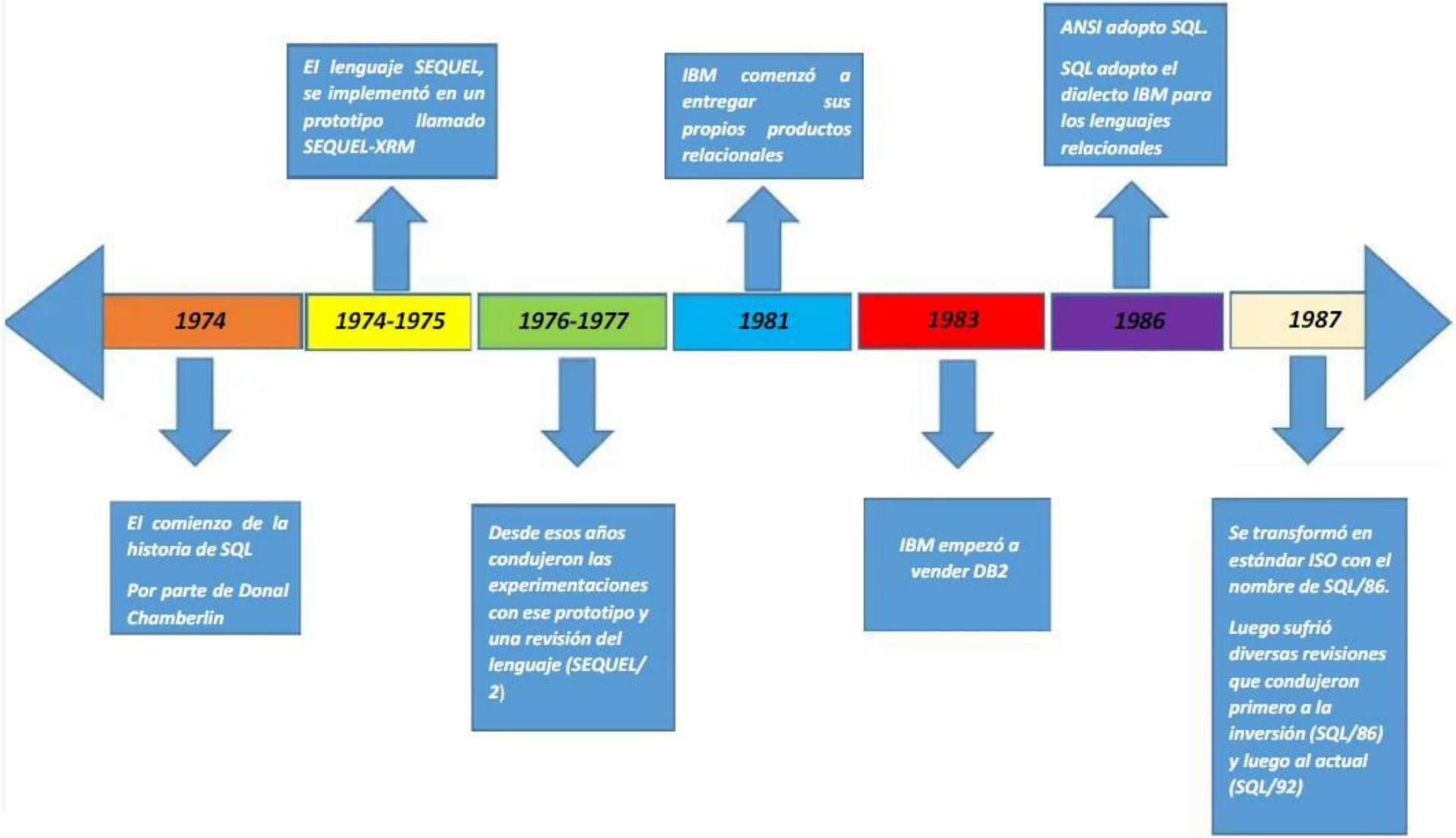
IBM, se creó el nuevo software de base de datos System R basado en el modelo teórico de E. Codd.

Allá por 1974 para gestionar los datos almacenados en System R, se creó el lenguaje SQL.

En un principio se llamó SEQUEL, un nombre que todavía se utiliza como una pronunciación alternativa para SQL, pero más tarde fue renombrado a sólo SQL.

SQL - Structured Query Language

Historia del SQL



Sublenguajes del SQL

- DDL - Data Definition Language
- DML - Data Manipulation Language
- TCL - Transaction Control Language
- DCL - Data Control Language


Data
Definition
Language

Data Definition Language

Es el sublenguaje que se encarga de la modificación de la estructura de los objetos de la base de datos, o sea de definir su metadata.

Posee instrucciones para modificar, borrar o definir tablas, vistas, base de datos, entre otros.

Estos comandos son:

- **CREATE**
- **ALTER**
- **DROP**
- **TRUNCATE**

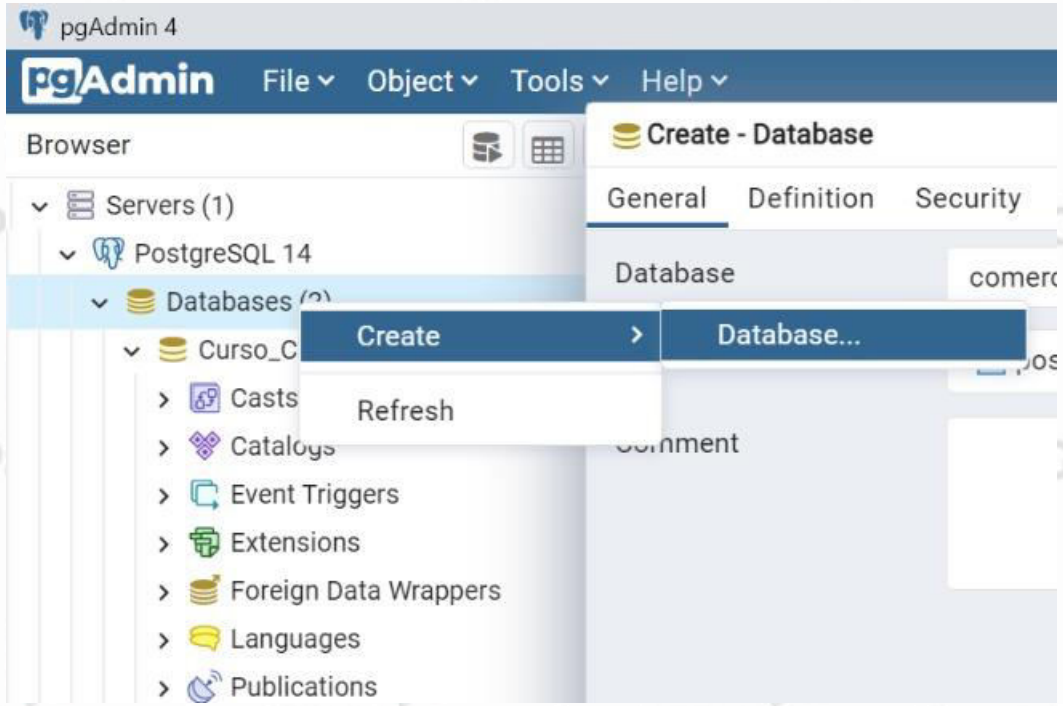
DataBase

¿Qué es una Base de Datos?

Una BD es un conjunto de datos persistentes e interrelacionados que es utilizado por los sistemas de aplicación de una empresa, los mismos se encuentran almacenados en un conjunto independiente y sin redundancias o con redundancias mínimas.

DataBase

CREATE DATABASE curso_cepetel,

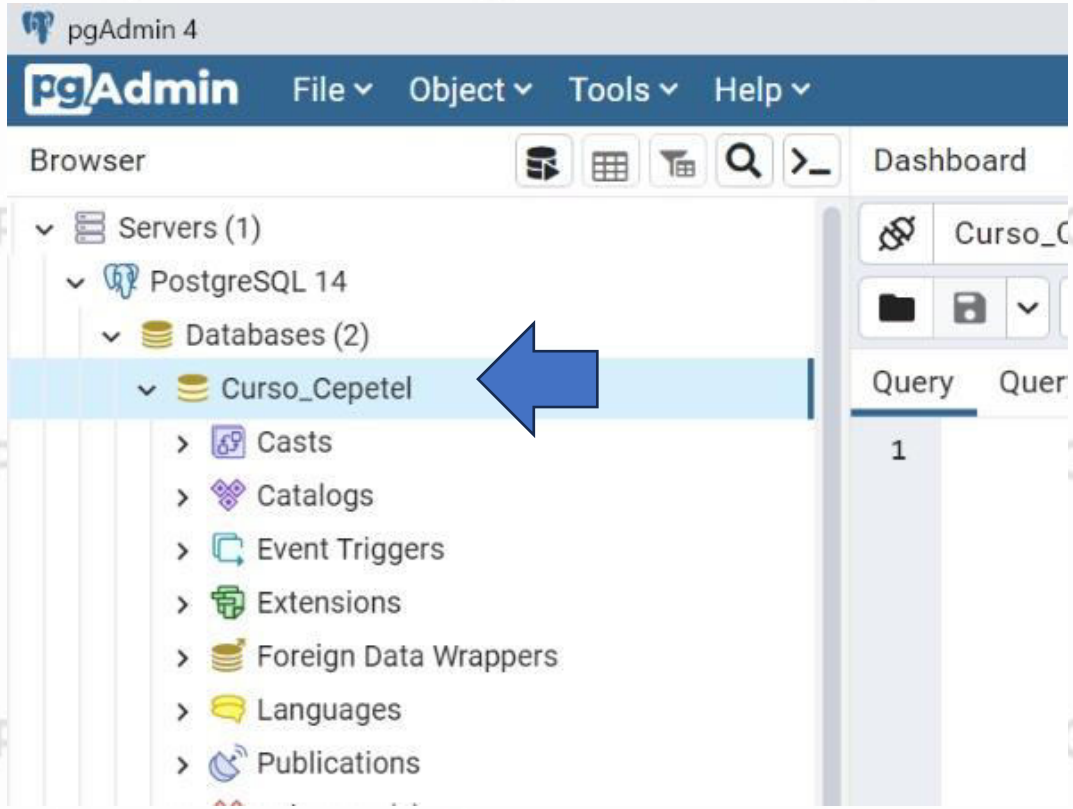


Este comando permite crear una base de datos con todas sus estructuras y componentes internas.
En general y dependiendo del motor de base de datos tiene un conjunto de parámetros asociados a estructuras físicas y espacios de discos de alocados.

Dentro del PGAdmin, botón derecho sobre databases y luego Create.

DataBase

USE curso_cepetel,



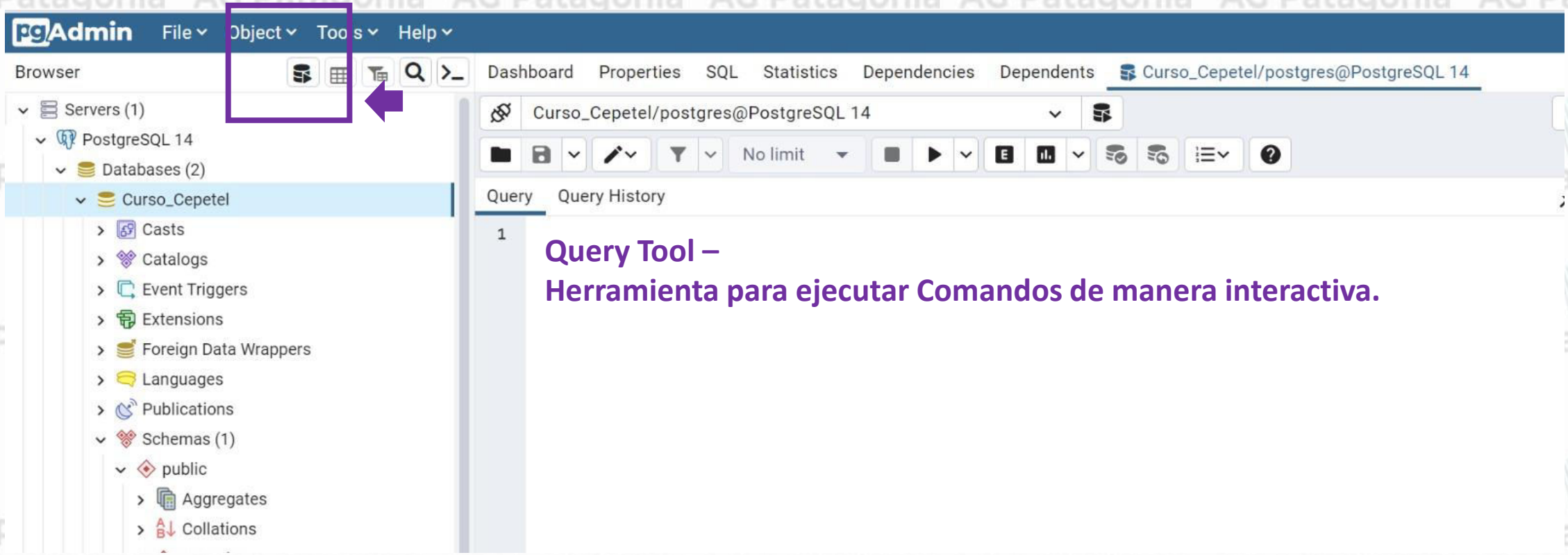
Este comando permite acceder a la Base de Datos, conectándonos a la misma.

Dentro del PGAdmin, doble click sobre la base de datos a acceder.

PgAdmin

GUI - Graphic User Interface

Desarrollada por la gente de PostgreSQL para acceder a las Bases de datos de una manera más amigable al usuario.



Query Tool –
Herramienta para ejecutar Comandos de manera interactiva.

PgAdmin

Tablas

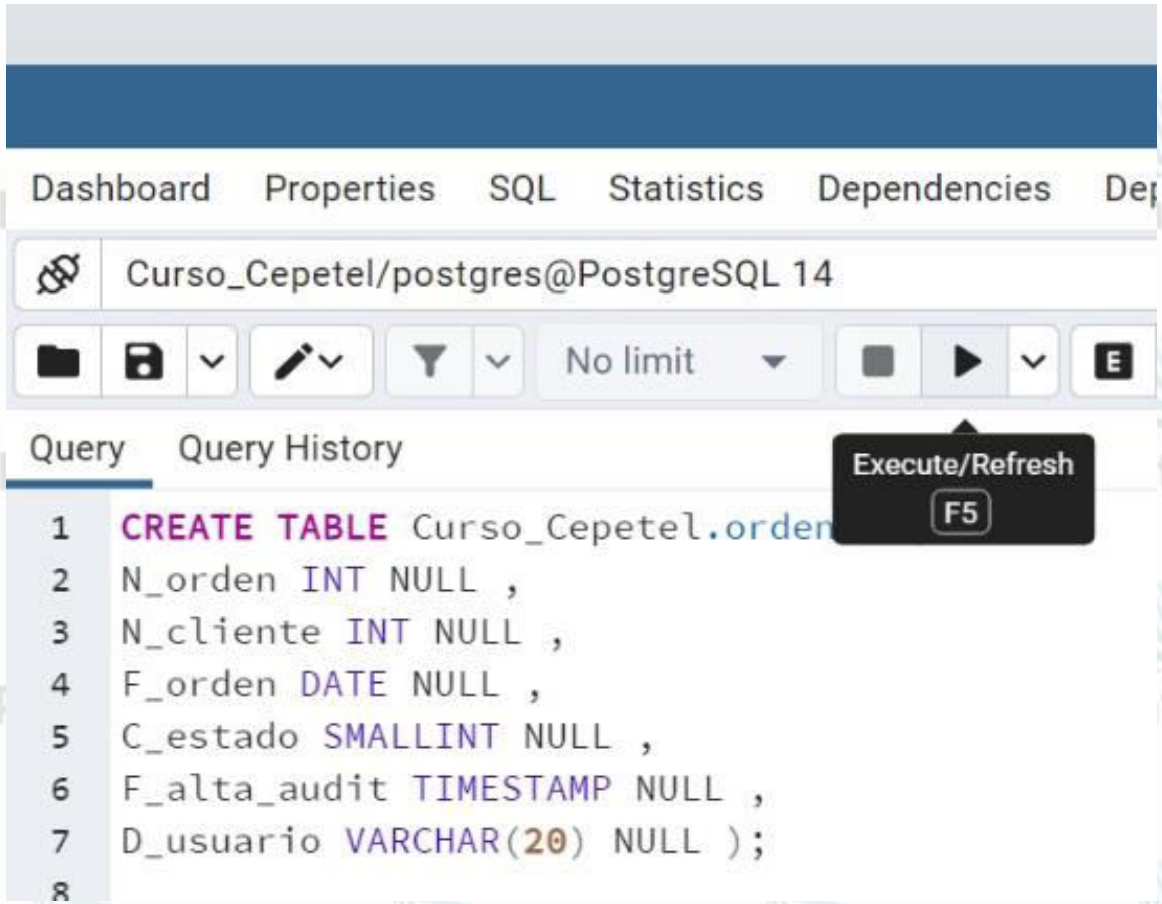
Es la unidad básica de almacenamiento de datos. Los datos están almacenados en filas y columnas. Son de existencia permanente y poseen un nombre identificatorio único por esquema o por base de datos (dependiendo del motor de base de datos).

Cada columna tiene entre otros datos un nombre, un tipo de datos y un ancho (este puede estar predeterminado por el tipo de dato).

PgAdmin

Tablas

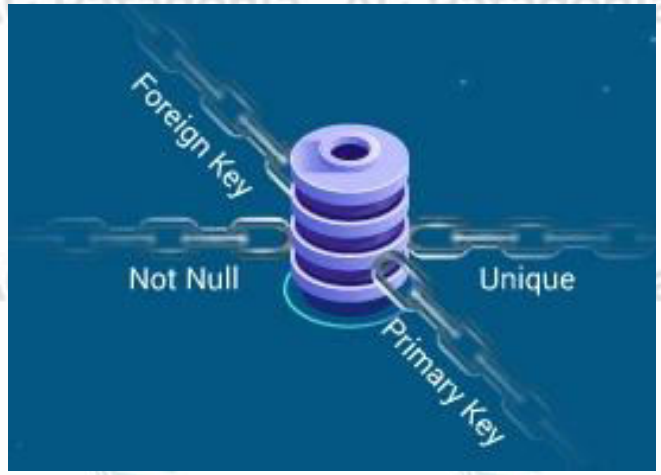
```
CREATE TABLE ordenes (
N_orden INT NULL ,
N_cliente INT NULL ,
F_orden DATE NULL ,
C_estado SMALLINT NULL ,
F_alta_audit TIMESTAMP NULL ,
D_usuario VARCHAR(20) NULL )
```



SQL Constraints

- NOT NULL Constraint
- CHECK Constraint
- UNIQUE Constraint
- PRIMARY KEY Constraint
- FOREIGN KEY Constraint
- DEFAULT Constraint

Constraints



Integridad de Entidad

La integridad de entidades es usada para asegurar que los datos pertenecientes a una misma tabla tienen una única manera de identificarse, es decir que cada fila de cada tabla tenga una primary key capaz de identificar unívocamente una fila y esa no puede ser nula

PRIMARY KEY CONSTRAINT: Puede estar compuesta por una o más columnas, y deberá representar unívocamente a cada fila de la tabla. No debe permitir valores nulos.

Integridad Referencial

La integridad referencial es usada para asegurar la coherencia entre datos de dos tablas.

FOREIGN KEY CONSTRAINT: Puede estar compuesta por una o más columnas, y estará referenciando a la PRIMARY KEY de otra tabla.

Los constraints referenciales permiten a los usuarios especificar claves primarias y foráneas para asegurar una relación PADRE-HIJO (MAESTRO-DETALLE).

CePETel

Constraints

Tipos de Constraints Referenciales

Ciclic Referential Constraint

Asegura una relación de PADRE-HIJO entre tablas. Es el más común.

Ej. CLIENTE FACTURAS

Self Referencing Constraint

Asegura una relación de PADRE-HIJO entre la misma tabla.

Ej. EMPLEADOS EMPLEADOS

Multiple Path Constraint

Se refiere a una PRIMARY KEY que tiene múltiples FOREIGN KEYS.
Este caso también es muy común.

Ej. CLIENTES FACTURAS
CLIENTES RECLAMOS

Constraints

Integridad Semántica

La integridad semántica es la que nos asegura que los datos que vamos a almacenar tengan una apropiada configuración y que respeten las restricciones definidas sobre los dominios o sobre los atributos.

- **DATA TYPE**
- **DEFAULT**
- **UNIQUE**
- **NOT NULL**
- **CHECK**

Constraints

Integridad Semántica

DATA TYPE: Este define el tipo de valor que se puede almacenar en una columna.

DEFAULT CONSTRAINT: Es el valor insertado en una columna cuando al insertar un registro ningún valor fue especificado para dicha columna. El valor default por default es el NULL.
Se aplica a columnas no listadas en una sentencia INSERT.
El valor por default puede ser un valor literal o una función SQL (USER, TODAY, etc.)
Aplicado sólo durante un INSERT (NO UPDATE).

UNIQUE CONSTRAINT: Específica sobre una o más columnas que la inserción o actualización de una fila contiene un valor único en esa columna o conjunto de columnas.

NOT NULL CONSTRAINT: Asegura que una columna contenga un valor durante una operación de INSERT o UPDATE. Se considera el NULL como la ausencia de valor.

Constraints

```
CREATE TABLE parts (
    part_no VARCHAR(18) PRIMARY KEY,
    description VARCHAR(40),
    cost DECIMAL(10,2 ) NOT NULL CHECK (cost >= 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0)
);
```

Integridad Semántica

CHECK CONSTRAINT: Especifica condiciones para la inserción o modificación en una columna. Cada fila insertada en una tabla debe cumplir con dichas condiciones. Actúa tanto en el INSERT, como en el UPDATE.

Es una expresión que devuelve un valor booleano de TRUE o FALSE.
Son aplicados para cada fila que es INSERTADA o MODIFICADA.

Todas las columnas a las que referencia deben ser de la misma tabla (la corriente).
No puede contener subconsultas, secuencias, funciones (de fecha, usuario) ni pseudocolumnas.

Todas las filas existentes en una tabla deben pasar un nuevo constraint creado para dicha tabla. En el caso de que alguna de las filas no cumpla, no se podrá crear dicho constraint o se creará en estado deshabilitado.

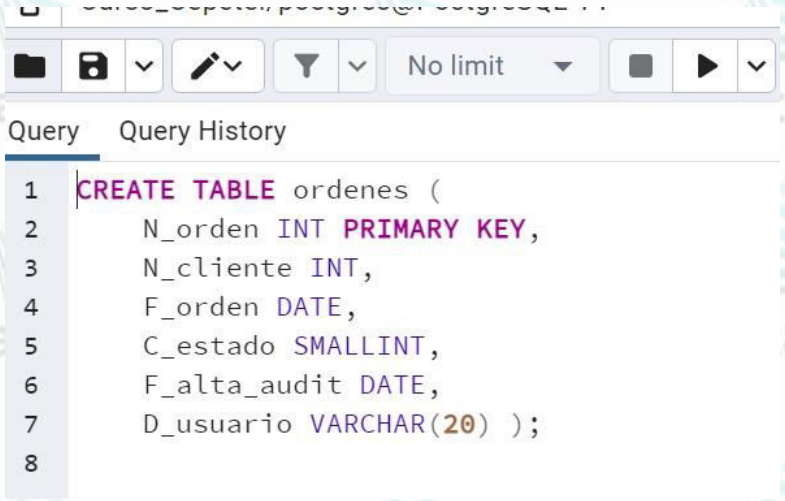
Constraints

Tipos de Constraints

Existen dos métodos para definir Constraints.

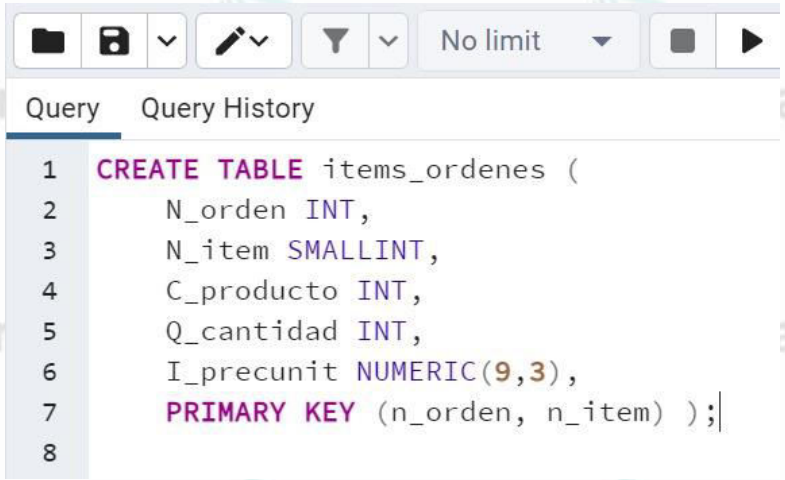
Restricciones a nivel de Columna

Ejemplo



Restricciones a nivel de Tabla

Ejemplo

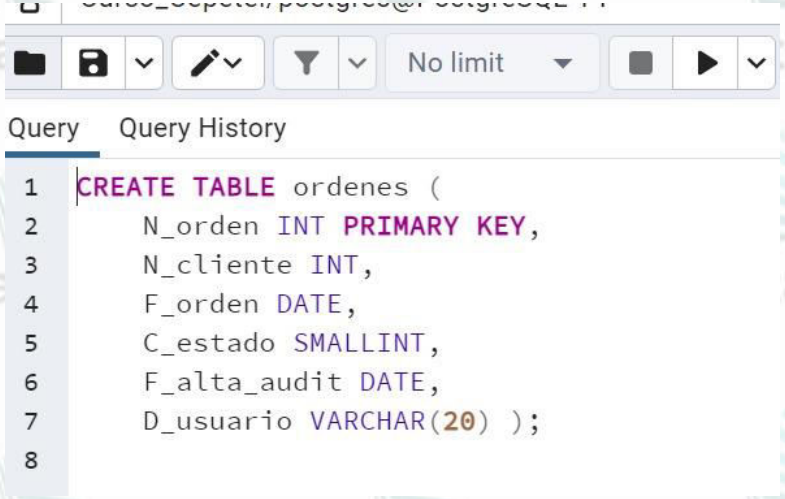


Cuando la restricción es sobre un grupo de columnas se debe utilizar una restricción a nivel de tabla, cuando es sobre sólo una columna puede utilizarse cualquiera de los dos modos.

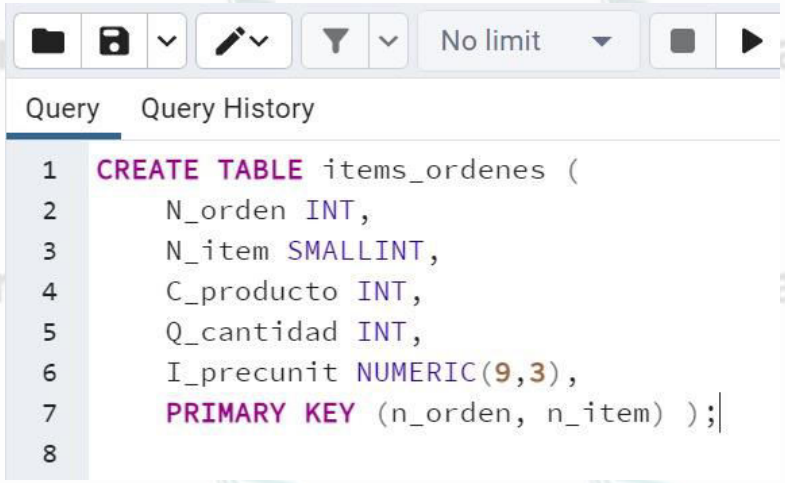
Constraints

Ejemplos de PRIMARY KEY

Restricciones a nivel de Columna
Ej



Restricciones a nivel de Tabla
Ej



Constraints

Ejemplos de FOREIGN KEY

Restricciones a nivel de Columna

Ej

```
CREATE TABLE ordenes (
    N_orden INT PRIMARY KEY,
    N_cliente INT,
    F_orden DATE,
    C_estado SMALLINT,
    F_alta_audit DATE,
    D_usuario VARCHAR(20) );
```

```
CREATE TABLE items_ordenes (
    N_orden INT REFERENCES ordenes,
    N_item SMALLINT,
    C_producto INT,
    Q_cantidad INT,
    I_precunit NUMERIC(9,3),
    PRIMARY KEY (n_orden, n_item));
```

Restricciones a nivel de Tabla

Ej

```
CREATE TABLE items_ordenes (
    N_orden INT,
    N_item SMALLINT,
    C_producto INT,
    Q_cantidad INT,
    I_precunit NUMERIC(9,3),
    PRIMARY KEY (n_orden, n_item) );
```

```
CREATE TABLE mov_stock (
    N_stock INT,
    C_movimiento SMALLINT,
    N_orden INT,
    N_item SMALLINT,
    C_producto INT,
    Q_cantidad INT,
    FOREIGN KEY (n_orden, n_item)
    REFERENCES items_ordenes);
```

Constraints

Ejemplos de SELF REFERENCING CONSTRAINT

Restricciones a nivel de Columna

Ej

Query | Query History

```
1  CREATE TABLE empleados (
2      N_empleado INTEGER PRIMARY KEY,
3      D_Apellido VARCHAR(60),
4      D_nombres VARCHAR(60),
5      N_cuil NUMERIC(11,0),
6      N_jefe INTEGER,
7      FOREIGN KEY (n_jefe) REFERENCES empleados (n_empleado) );
```

El motor no permitirá ingresar un empleado cuyo nro. de jefe no exista como número de empleado.
Lo que si permitirá es ingresar un nro. de jefe NULO.

Constraints

Ejemplos de DEFAULT

Query | Query History

```
1  CREATE TABLE ordenes
2  (
3      N_orden INT NOT NULL,
4      N_cliente INT,
5      F_orden DATE,
6      C_estado SMALLINT DEFAULT 1,
7      F_alta_audit DATE DEFAULT CURRENT_DATE,
8      D_usuario VARCHAR(20) DEFAULT USER
9  );
```

Al de ejecutar el siguiente comando

```
INSERT INTO ordenes (n_orden, n_cliente, f_orden) VALUES (117, 10, '2020-10-05');
```

n_orden integer	n_cliente integer	f_orden date	c_estado smallint	f_alta_audit date	d_usuario character varying (20)
117	10	2020-10-05	1	2020-10-04	postgres

Constraints

Ejemplos de NOT NULL

Query | Query History

```
1  CREATE TABLE ordenes
2  (
3      N_orden INT NOT NULL,
4      N_cliente INT,
5      F_orden DATE,
6      C_estado SMALLINT NOT NULL,
7      F_alta_audit DATE,
8      D_usuario VARCHAR(20)
9  );
10
11
```

Constraints

Ejemplos de UNIQUE

Restricciones a nivel de Columna Ej

Query | Query History

```
1  CREATE TABLE empleados (
2      N_empleado INT PRIMARY KEY,
3      D_Apellido VARCHAR(60),
4      D_nombres VARCHAR(60),
5      N_cuil BIGINT UNIQUE,
6      F_nacimiento DATE,
7      F_ingreso DATE,
8      N_jefe INT);
9
```

Restricciones a nivel de Tabla Ej

Query | Query History

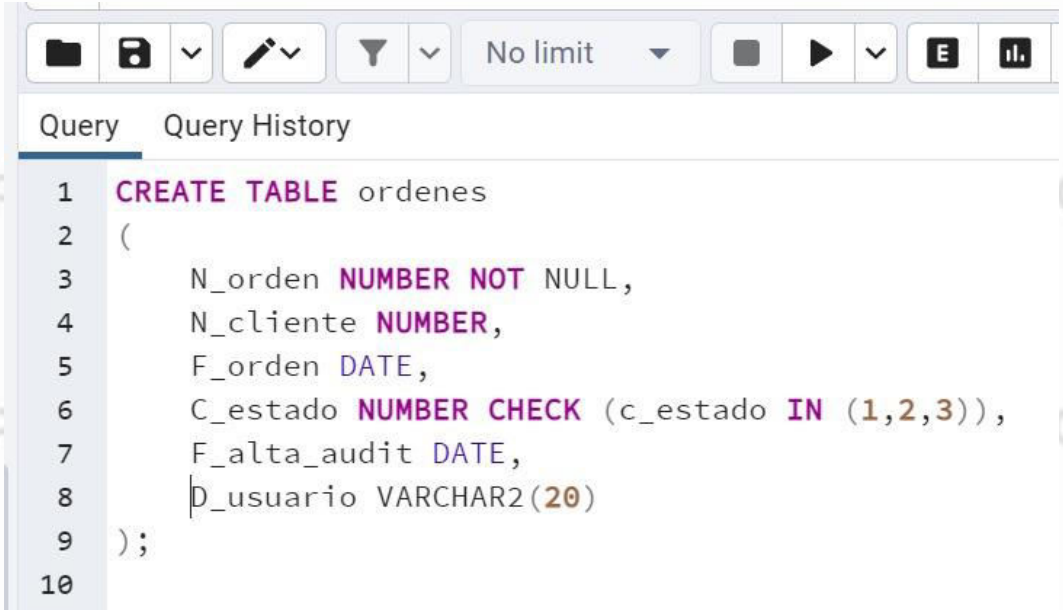
```
1  CREATE TABLE empleados (
2      N_empleado INT PRIMARY KEY,
3      D_Apellido VARCHAR(60),
4      D_nombres VARCHAR(60),
5      T_docum SMALLINT,
6      N_docum NUMERIC(11,0),
7      F_nacimiento DATE,
8      F_ingreso DATE,
9      N_jefe INT,
10     UNIQUE (t_docum, n_docum) );
11
```

La tabla empleados tiene cómo clave primaria al atributo **n_empleado**.
Con una restricción de UNIQUE podemos representar claves alternas.
En el primer ejemplo, el n_cuil es un atributo que posee valores únicos para cada fila de la tabla **empleados**.
En el segundo ejemplo, la clave compuesta por los atributos t_docum y n_docum (tipo y número de documento) posee valores únicos para cada fila de la tabla **empleados**.

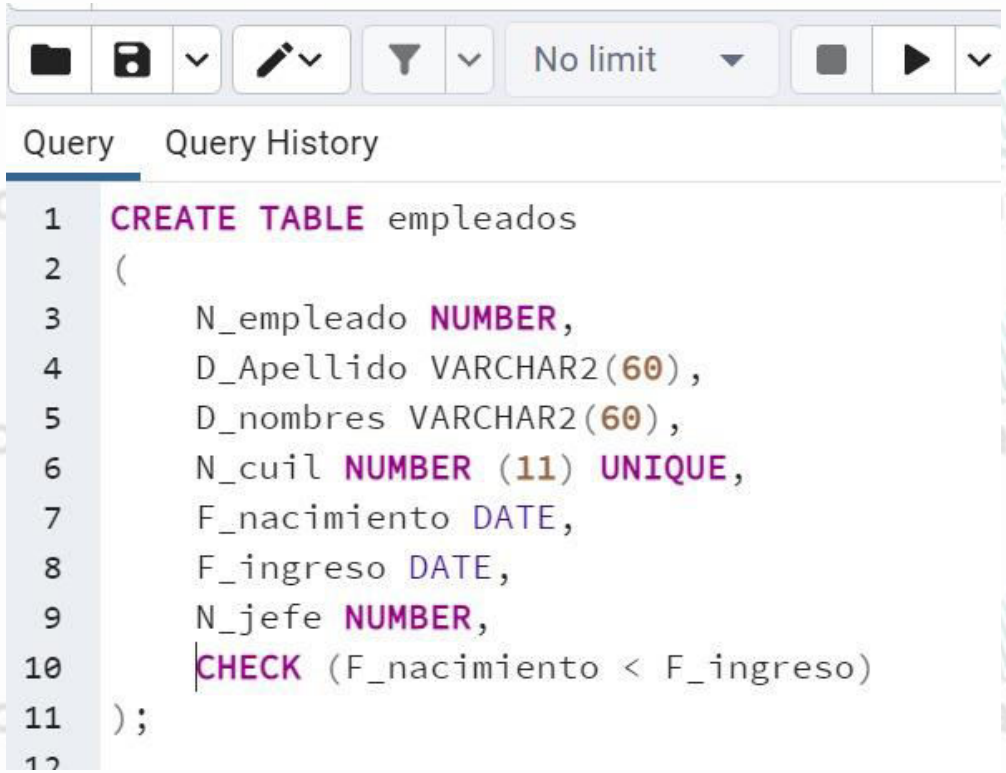
Constraints

Ejemplos de CHECK

Restricciones a nivel de Columna



Restricciones a nivel de Tabla



CePETel

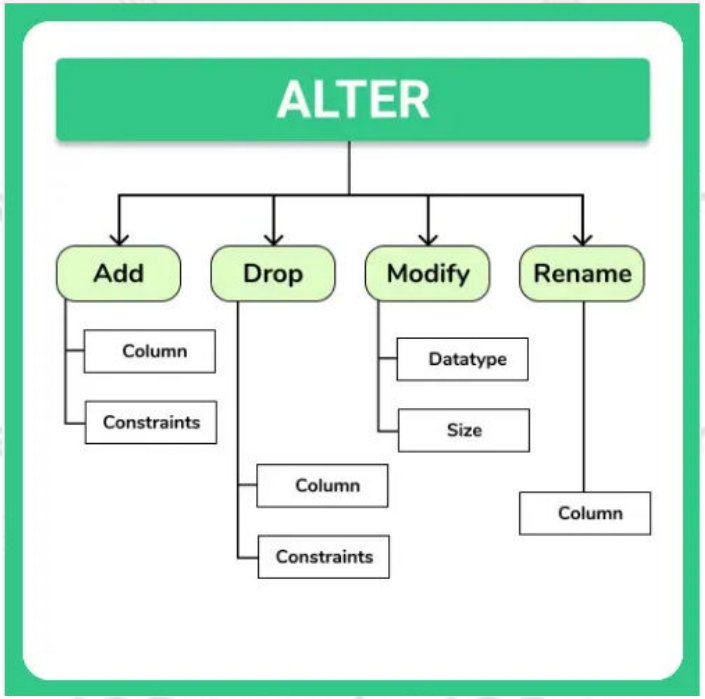
ALTER TABLE

Video: Alter Table ADD COLUMN

Ejemplos de ALTER TABLE

Este comando permite modificar la estructura y metadata de una tabla dada.

```
Renombrar una tabla
ALTER TABLE ordenes RENAME TO ordenesCompra;

Renombrar una columna
ALTER TABLE ordenes RENAME COLUMN n_orden TO nroOrden;

Agregar una nueva columna
ALTER TABLE ordenes ADD COLUMN fechaEmbarque DATE;

Eliminar una columna existente
ALTER TABLE ordenes DROP COLUMN address RESTRICT;
(Restrict no permite eliminarla si tiene dependencias)

Agregar un Constraint a la tabla
ALTER TABLE ordenes ADD CONSTRAINT Fembchk CHECK (fechaEmbarque >= fechaEmision);

Agregar un Constraint a la tabla
ALTER TABLE ONLY ordenes DROP CONSTRAINT Fembchk;
```



CePETel

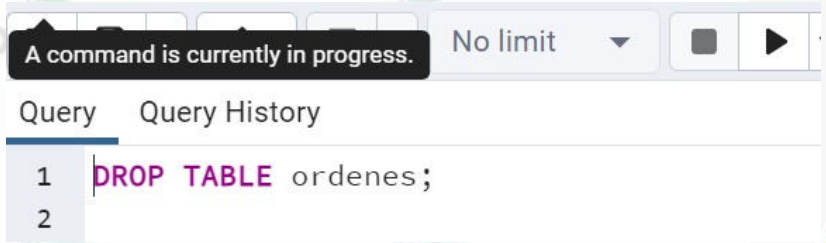
Comando DROP

Ejemplos de DROP

DROP TABLE

DROP TABLE ordenes;

Elimina la tabla órdenes y su contenido.
El motor de base de datos chequeará que no existan dependencias de otras tablas hacia la tabla a borrar.



DROP DATABASE

DROP DATABASE comercial;

Elimina la base de datos con todos sus objetos creados y su contenido.
El motor de base de datos chequeará que no existan conexiones vigentes en la Base a borrar.

Comando TRUNCATE TABLE

Ejemplos de DROP

TRUNCATE TABLE

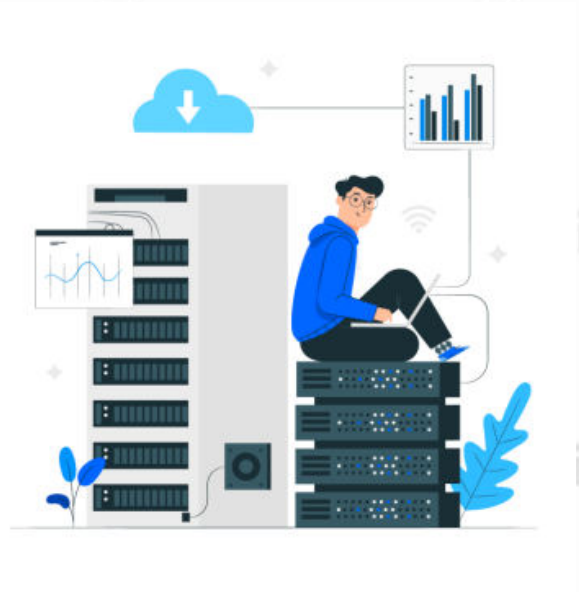
Query | Query History

```
1  TRUNCATE TABLE ordenes;
2
```

Elimina el contenido de la tabla ordenes, sin eliminar su Definición.
El motor de base de datos chequeará que no existan dependencias de otras tablas hacia los datos de la tabla a borrar.

Es muy utilizado cuando uno quiere eliminar todos los datos de una tabla, ya que es mucho más eficiente quel el comando del DML - DELETE FROM ordenes.

TP N° 3: Data Definition Languaje



Desarrolle en un notepad los comandos de creación de las Tablas y contraints ó restricciones (PK,FK, NOT NULL, UNIQUE, CHECK, DEFAULT) correspondientes a la base de datos desarrollada en el TP N°2

Por ejemplo:

```
CREATE DATABASE zoológicos_mundo;

USE zoologicos_mundo;

CREATE TABLE zoologicos
(
cod_zoo SMALLINT PRIMARY KEY,
nombre_zoo VARCHAR(100) NOT NULL
ubicacion  …….
…..
);
```


CePETel

TP N° 3: Data Definition Languaje

Ejemplos de datos que tendrian las tablas

TABLA ZOO

NOMBRE	CUIDAD	PAIS	TAMAÑO	PRESUPUESTO
Denver zoo	Denver	USA	850.000 m^2	$250.000 USD
Brooklyn Zoo	New york	USA	500.000 m^2	$300.000 USD

TABLA ESPECIE

Nombre cientifico	Nombre Vulgar	Familia	Peligro
Felis catus	Gato	Felidae	No
Loxodontoa africana	Elefante africano	Elephantidae	Si

TABLA ANIMALES

ID	NOMBRE ZOO	NOMBRE ESPECIE	SEXO	AÑO NACIMIENTO	PAIS	CONTINENTE
8567	Denver	Felis silvestris	Masculino	25-07-95	Peru	Sur america
2048	Brooklyn	Elephas maximus	Femenino	18-05-02	Congo	Africa

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

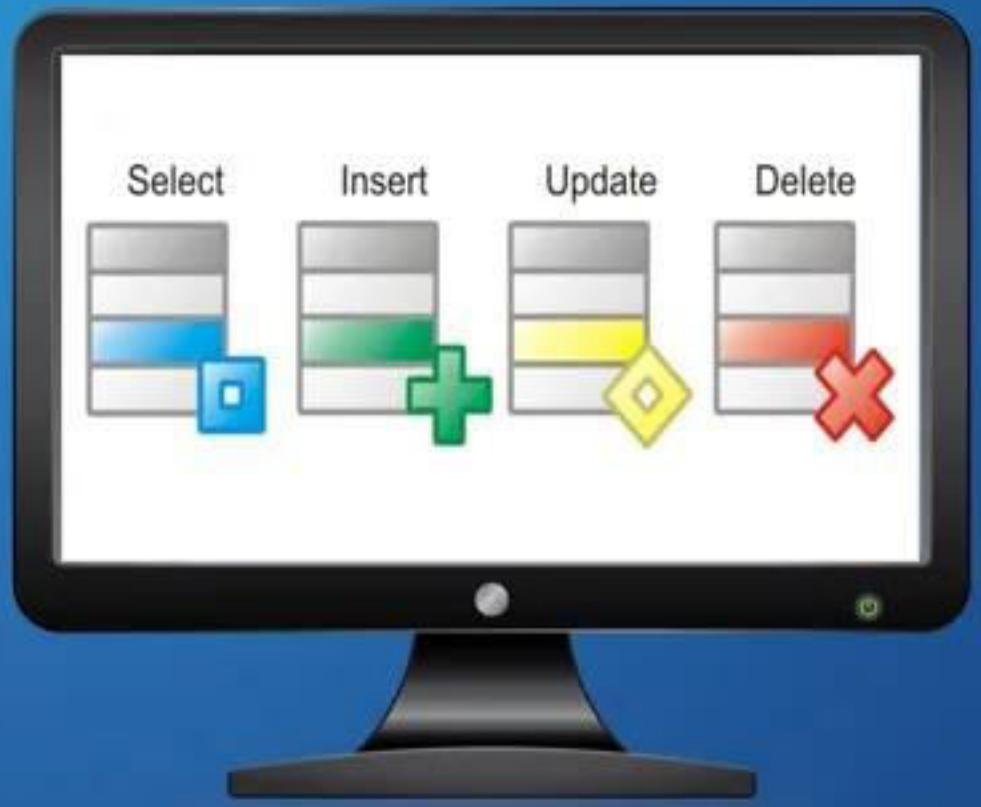
Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 4: DML (Data Manipulation Language) - Select

Data Manipulation Language

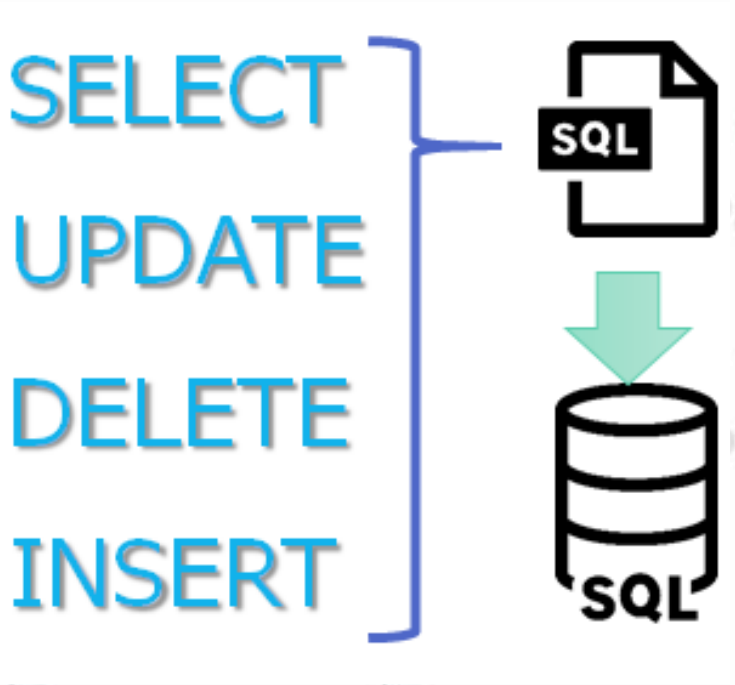


SQL - Structured Query Language

Video: Conociendo la historia SQL

Sublenguajes del SQL

- DDL - Data Definition Language
- DML - Data Manipulation Language
  - Select
  - Insert
  - Update
  - Delete



SQL - Operador Select

SELECT * | lista de columnas

FROM nom_tabla | lista de tablas

WHERE condiciones ó filtros

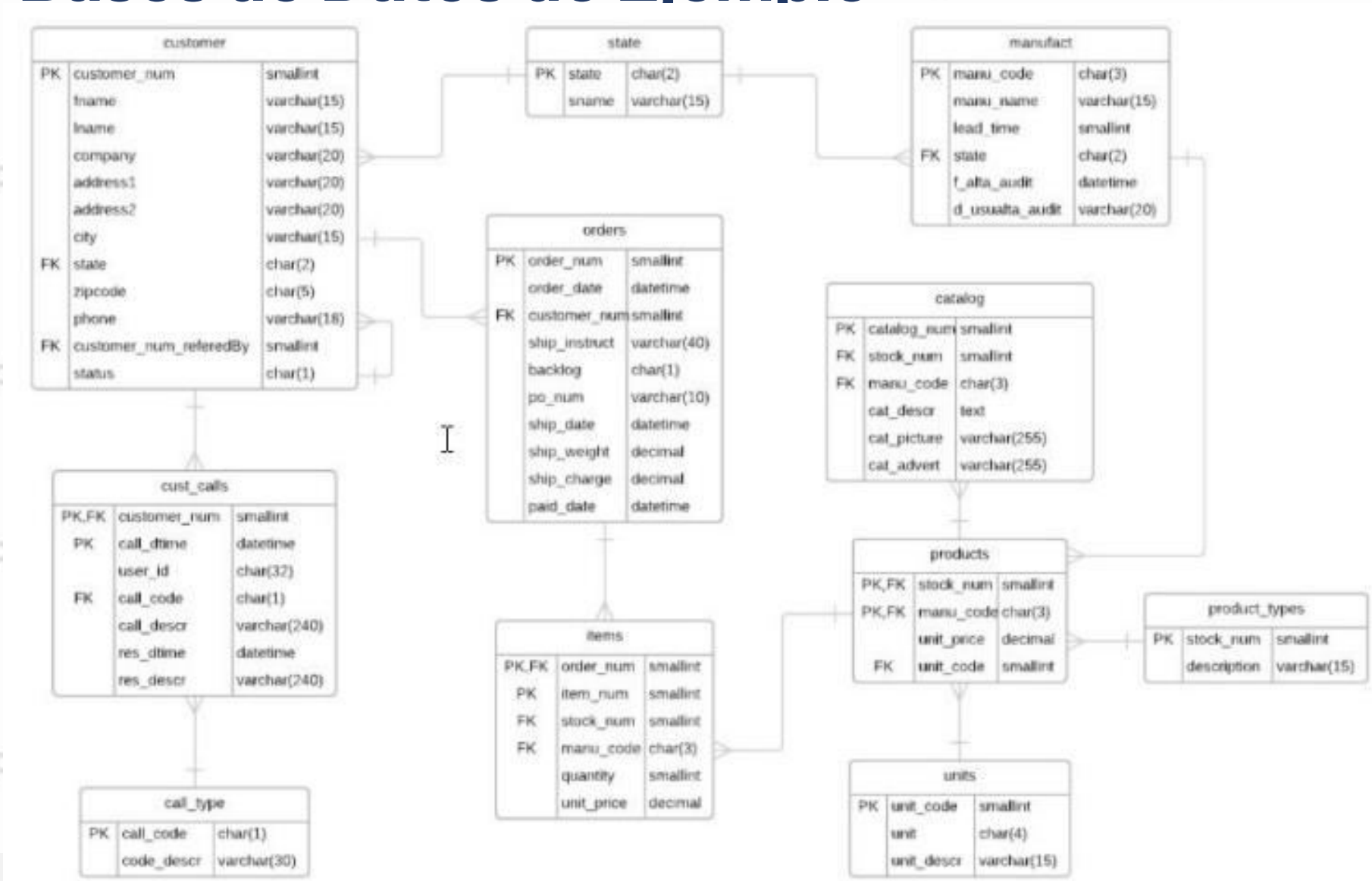
GROUP BY columnas clave de agrupamiento

HAVING condiciones sobre lo agrupado

ORDER BY columnas clave de ordenamiento

LIMIT limita la cantidad de filas a mostrar

Bases de Datos de Ejemplo



CePETel

SQL - Operador Select

```
SELECT *
FROM customer
```

Consulta todas las columnas de la tabla customer.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT * FROM customer
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	company character varying (20)	address1 character varying (20)	address2 character varying (20)	city character varying (15)	state character (2)	zipcode character (5)
1	101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086
2	102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117
3	103	Philip	Currie	Phil s Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303
4	104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026
5	105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022
6	106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063
7	107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304
8	108	Donald	Quinn	Quinn s Sports	587 Alvarado		Redwood City	CA	94063
9	109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086
10	110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062
11	111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085
12	112	Margaret	Lawson	Runners & Others	234 Wyandotte Way		Los Altos	CA	94022
13	113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025
14	114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062
15	115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025
16	116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040
17	117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063
18	118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609

CePETel

SQL - Operador Select

```
SELECT  lname, fname, state, city
FROM    customer
```

Consulta de distintas columnas pertenecientes a la tabla customer.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT   lname, fname, state, city
2  FROM     customer
3
```

Data Output | Explain | Messages | Notifications

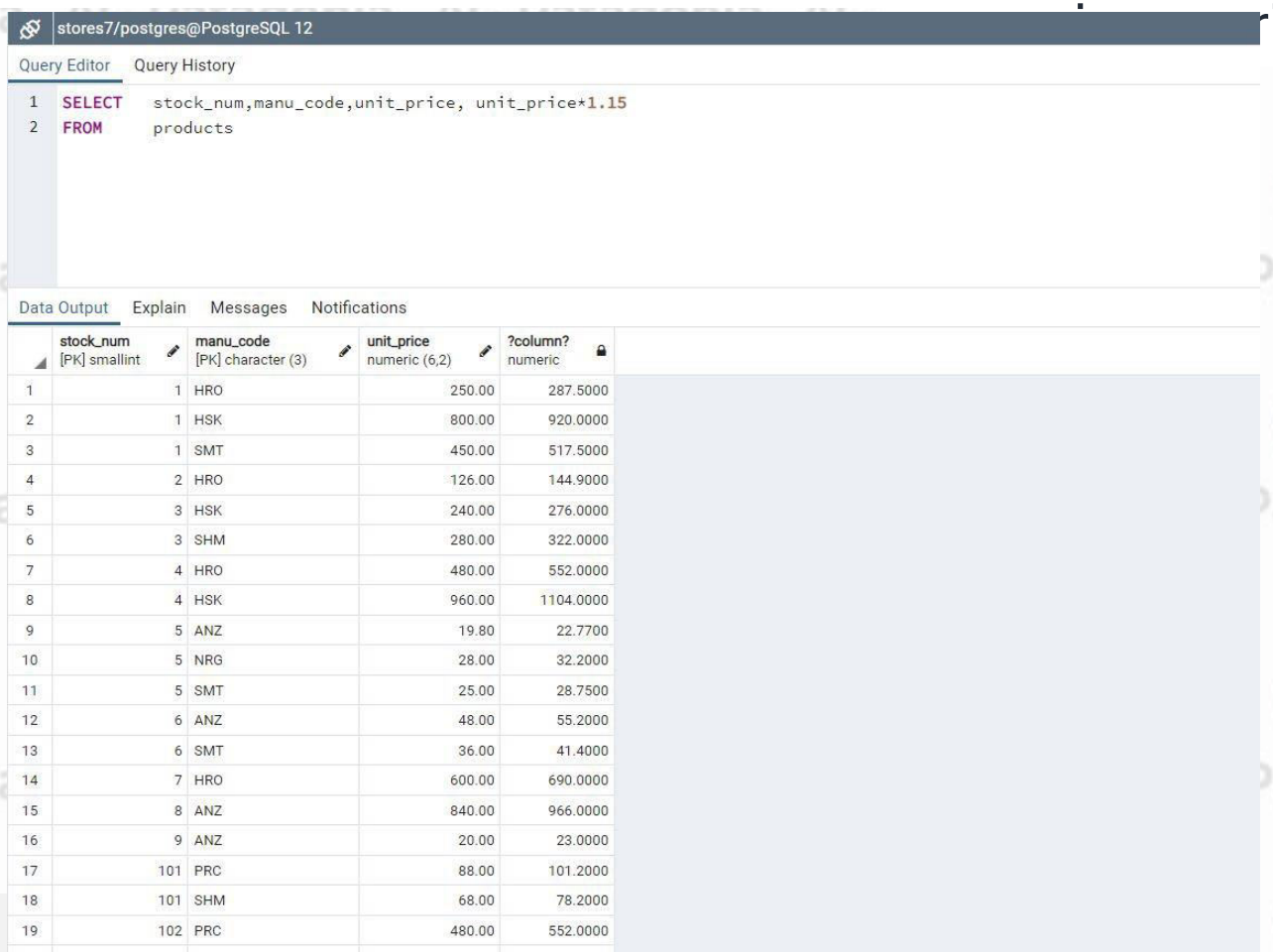
	lname character varying (15)	fname character varying (15)	state character (2)	city character varying (15)
1	Pauli	Ludwig	CA	Sunnyvale
2	Sadler	Carole	CA	San Francisco
3	Currie	Philip	CA	Palo Alto
4	Higgins	Anthony	CA	Redwood City
5	Vector	Raymond	CA	Los Altos
6	Watson	George	CA	Mountain View
7	Ream	Charles	CA	Palo Alto
8	Quinn	Donald	CA	Redwood City
9	Miller	Jane	CA	Sunnyvale
10	Jaeger	Roy	CA	Redwood City
11	Keyes	Frances	CA	Sunnyvale
12	Lawson	Margaret	CA	Los Altos
13	Beatty	Lana	CA	Menlo Park
14	Albertson	Frank	CA	Redwood City
15	Grant	Alfred	CA	Menlo Park
16	Parmelee	Jean	CA	Mountain View
17	Sipes	Arnold	CA	Redwood City
18	Baxter	Dick	CA	Oakland
19	Shorter	Bob	NJ	Cherry Hill

CePETel

SQL - Operador Select

```
SELECT   stock_num,manu_code,unit_price, unit_price*1.15
FROM     products
```

Consulta de distintas columnas con
ritméticas.



	stock_num [PK] smallint	manu_code [PK] character (3)	unit_price numeric (6,2)	?column? numeric
1	1	HRO	250.00	287.5000
2	1	HSK	800.00	920.0000
3	1	SMT	450.00	517.5000
4	2	HRO	126.00	144.9000
5	3	HSK	240.00	276.0000
6	3	SHM	280.00	322.0000
7	4	HRO	480.00	552.0000
8	4	HSK	960.00	1104.0000
9	5	ANZ	19.80	22.7700
10	5	NRG	28.00	32.2000
11	5	SMT	25.00	28.7500
12	6	ANZ	48.00	55.2000
13	6	SMT	36.00	41.4000
14	7	HRO	600.00	690.0000
15	8	ANZ	840.00	966.0000
16	9	ANZ	20.00	23.0000
17	101	PRC	88.00	101.2000
18	101	SHM	68.00	78.2000
19	102	PRC	480.00	552.0000

CePETel

SQL - Operador Select

```
SELECT   stock_num,manu_code,unit_price, unit_price*1.15 NewPrice
FROM     products
```

Alias de Columnas o Etiquetas

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT   stock_num,manu_code,unit_price, unit_price*1.15 NewPrice
2  FROM     products
```

Data Output | Explain | Messages | Notifications

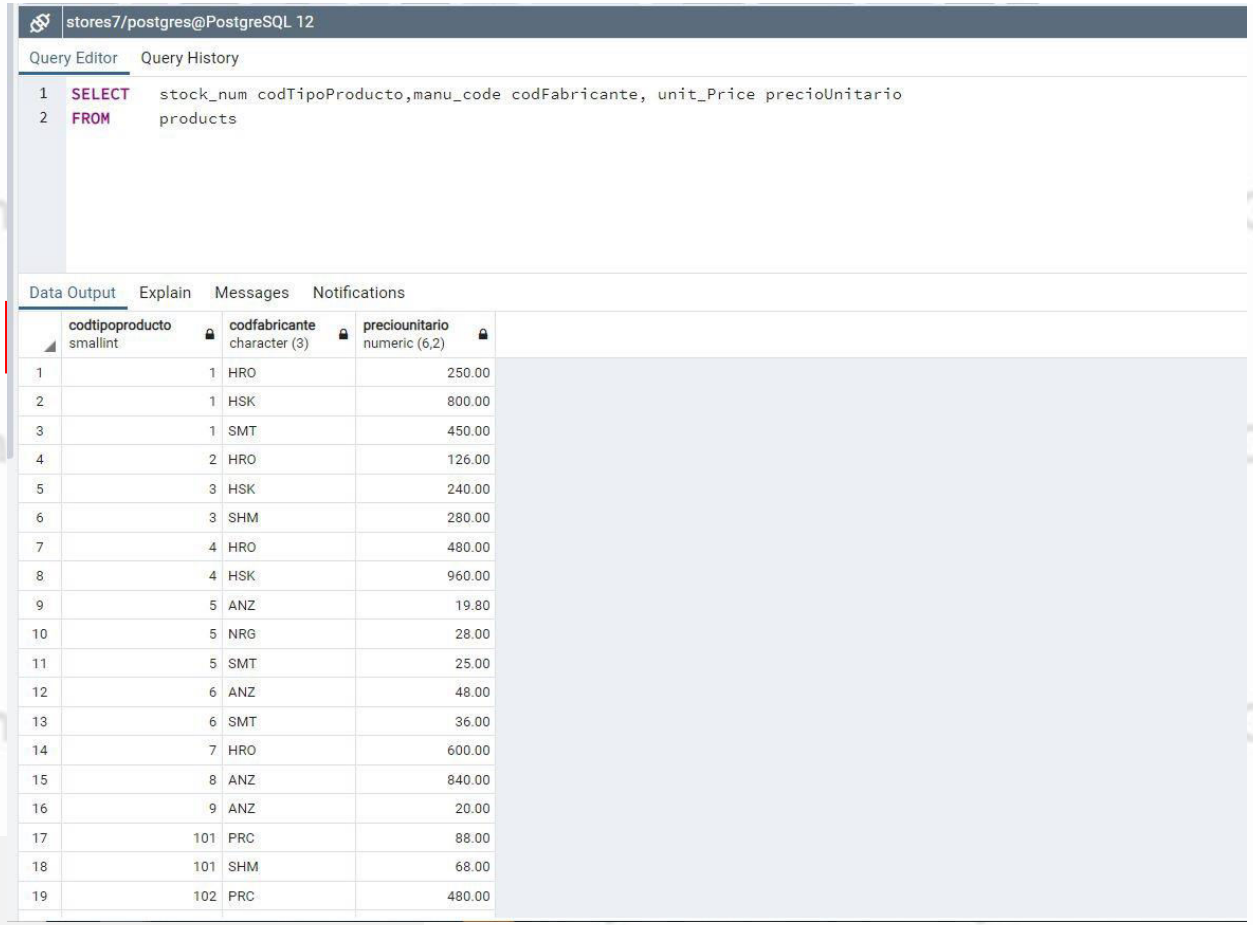
	stock_num [PK] smallint	manu_code [PK] character (3)	unit_price numeric (6,2)	newprice numeric
1	1	HRO	250.00	287.5000
2	1	HSK	800.00	920.0000
3	1	SMT	450.00	517.5000
4	2	HRO	126.00	144.9000
5	3	HSK	240.00	276.0000
6	3	SHM	280.00	322.0000
7	4	HRO	480.00	552.0000
8	4	HSK	960.00	1104.0000
9	5	ANZ	19.80	22.7700
10	5	NRG	28.00	32.2000
11	5	SMT	25.00	28.7500
12	6	ANZ	48.00	55.2000
13	6	SMT	36.00	41.4000
14	7	HRO	600.00	690.0000
15	8	ANZ	840.00	966.0000
16	9	ANZ	20.00	23.0000
17	101	PRC	88.00	101.2000
18	101	SHM	68.00	78.2000
19	102	PRC	480.00	552.0000

CePETel

SQL - Operador Select

```
SELECT   stock_num codTipoProducto,manu_code codFabricante,
unit_Price precioUnitario
FROM     products
```

Alias de Columnas o Etiquetas



	codtipoproducto smallint	codfabricante character (3)	preciounitario numeric (6,2)
1	1	HRO	250.00
2	1	HSK	800.00
3	1	SMT	450.00
4	2	HRO	126.00
5	3	HSK	240.00
6	3	SHM	280.00
7	4	HRO	480.00
8	4	HSK	960.00
9	5	ANZ	19.80
10	5	NRG	28.00
11	5	SMT	25.00
12	6	ANZ	48.00
13	6	SMT	36.00
14	7	HRO	600.00
15	8	ANZ	840.00
16	9	ANZ	20.00
17	101	PRC	88.00
18	101	SHM	68.00
19	102	PRC	480.00

Se puede utilizar también para mostrar columnas con otros nombres

CePETel

SQL - Operador Select

```
SELECT lname || ', ' || fname apellidoYNombre, company, city
FROM   customer
```

Concatenar columnas en una sola nueva columna de salida.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  lname || ', ' || fname apellidoYNombre, company, city
2  FROM     customer
3
```

Data Output | Explain | Messages | Notifications

	apellidoynombre text	company character varying (20)	city character varying (15)
1	Pauli, Ludwig	All Sports Supplies	Sunnyvale
2	Sadler, Carole	Sports Spot	San Francisco
3	Currie, Philip	Phil s Sports	Palo Alto
4	Higgins, Anthony	Play Ball!	Redwood City
5	Vector, Raymond	Los Altos Sports	Los Altos
6	Watson, George	Watson & Son	Mountain View
7	Ream, Charles	Athletic Supplies	Palo Alto
8	Quinn, Donald	Quinn s Sports	Redwood City
9	Miller, Jane	Sport Stuff	Sunnyvale
10	Jaeger, Roy	AA Athletics	Redwood City
11	Keyes, Frances	Sports Center	Sunnyvale
12	Lawson, Margaret	Runners & Others	Los Altos
13	Beatty, Lana	Sportstown	Menlo Park
14	Albertson, Frank	Sporting Place	Redwood City
15	Grant, Alfred	Gold Medal Sports	Menlo Park
16	Parmelee, Jean	Olympic City	Mountain View
17	Sipes, Arnold	Kids Korner	Redwood City
18	Baxter, Dick	Blue Ribbon Sports	Oakland
19	Shorter, Bob	The Triathletes Club	Cherry Hill

Además, en el ejemplo ponemos el alias **apellidoYNombre**

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, DATE_PART('year',order_date) anio,
        DATE_PART('month',order_date) mes,
        DATE_PART('day',order_date) dia,
        CURRENT_USER
FROM    orders
```

Utilización de funciones especiales.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  order_num, order_date, DATE_PART('year',order_date) anio,
2                  DATE_PART('month',order_date) mes, DATE_PART('day',order_date) dia,
3                  current_user
4  FROM    orders
5
```

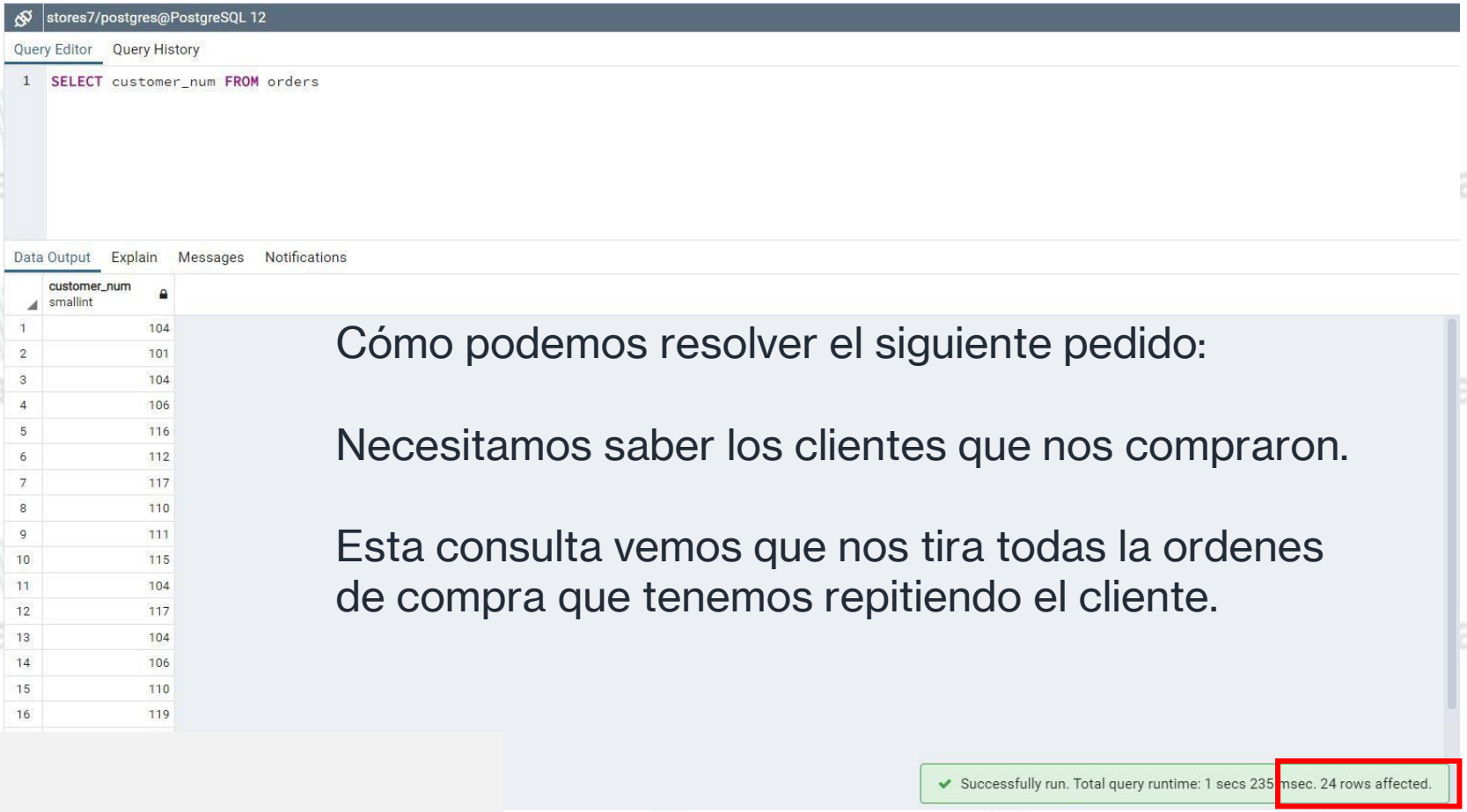
Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	anio double precision	mes double precision	dia double precision	current_user name
1	1001	2015-05-16	2015	5	16	postgres
2	1002	2015-05-17	2015	5	17	postgres
3	1003	2015-05-18	2015	5	18	postgres
4	1004	2015-05-18	2015	5	18	postgres
5	1005	2015-05-20	2015	5	20	postgres
6	1006	2015-05-26	2015	5	26	postgres

ARITMÉTICAS
TRIGONOMÉTRICAS
FINANCIERAS
DE FECHA
DE STRINGS
Entre otras.

SQL - Operador Select

```
SELECT customer_num
FROM orders
```



Cómo podemos resolver el siguiente pedido:

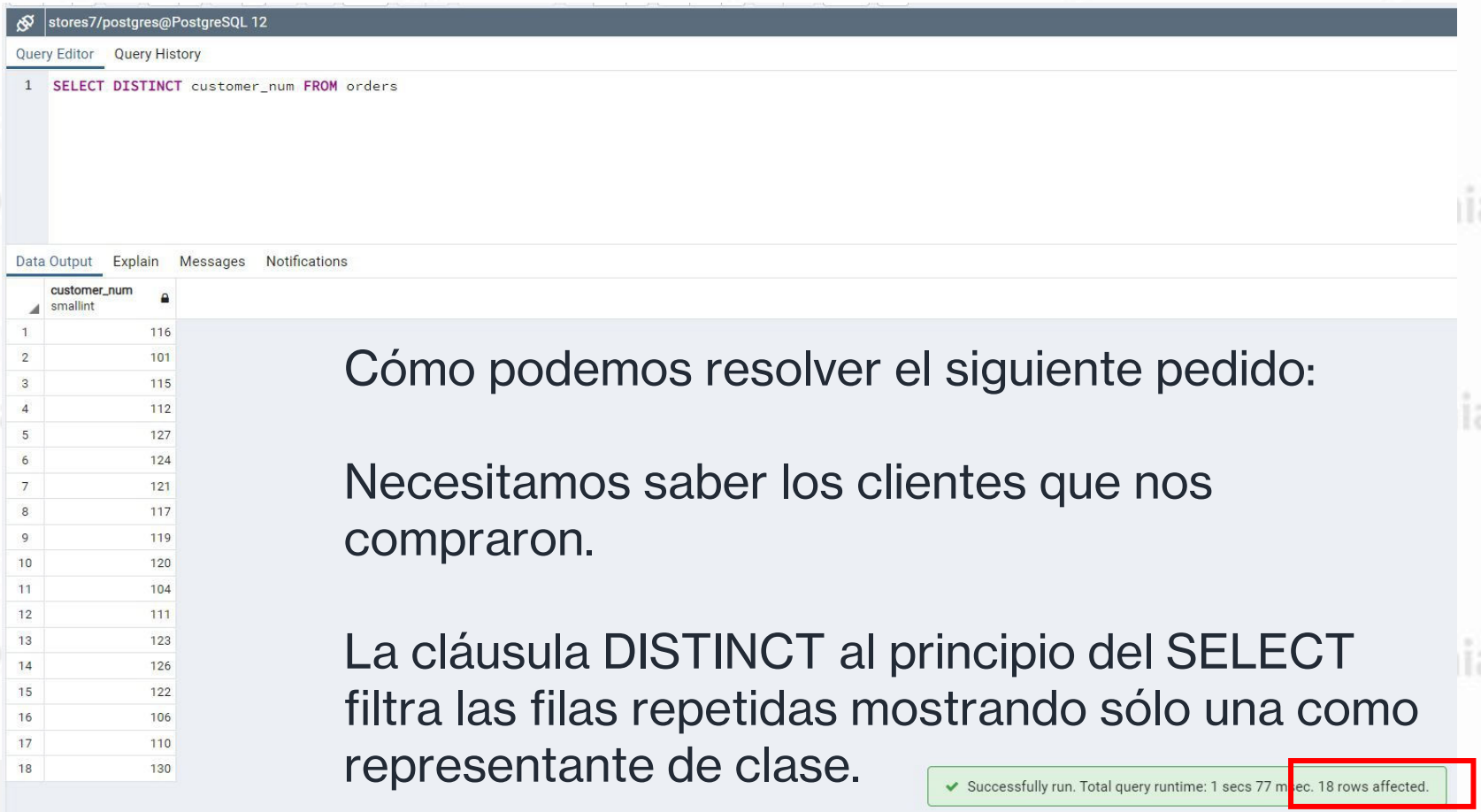
Necesitamos saber los clientes que nos compraron.

Esta consulta vemos que nos tira todas la ordenes de compra que tenemos repitiendo el cliente.

CePETel

SQL - Operador Select

```
SELECT DISTINCT customer_num
FROM orders
```



Cómo podemos resolver el siguiente pedido:

Necesitamos saber los clientes que nos compraron.

La cláusula DISTINCT al principio del SELECT filtra las filas repetidas mostrando sólo una como representante de clase.

CePETel

COMANDOS
DDL Y DML

DDL

- Son aquellos que permiten crear y definir nuevas bases de datos, campos e índices.

Create
Drop
Alter

DML

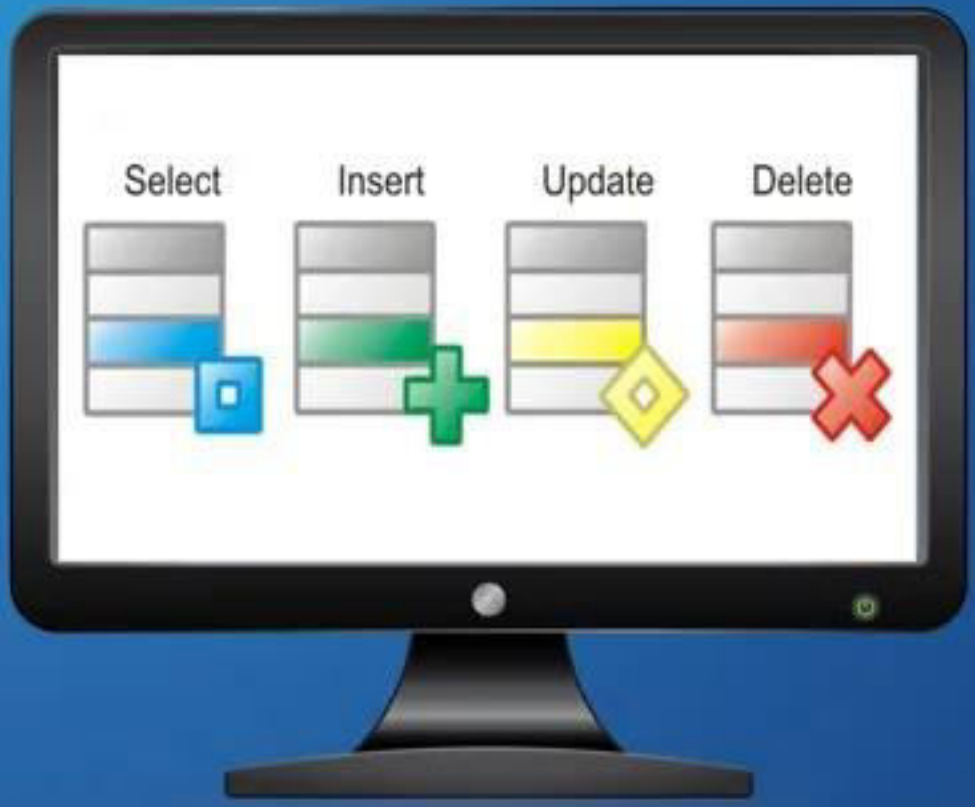
- Son aquellos que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

Select
Insert
Update
Delete

Data Manipulation Language

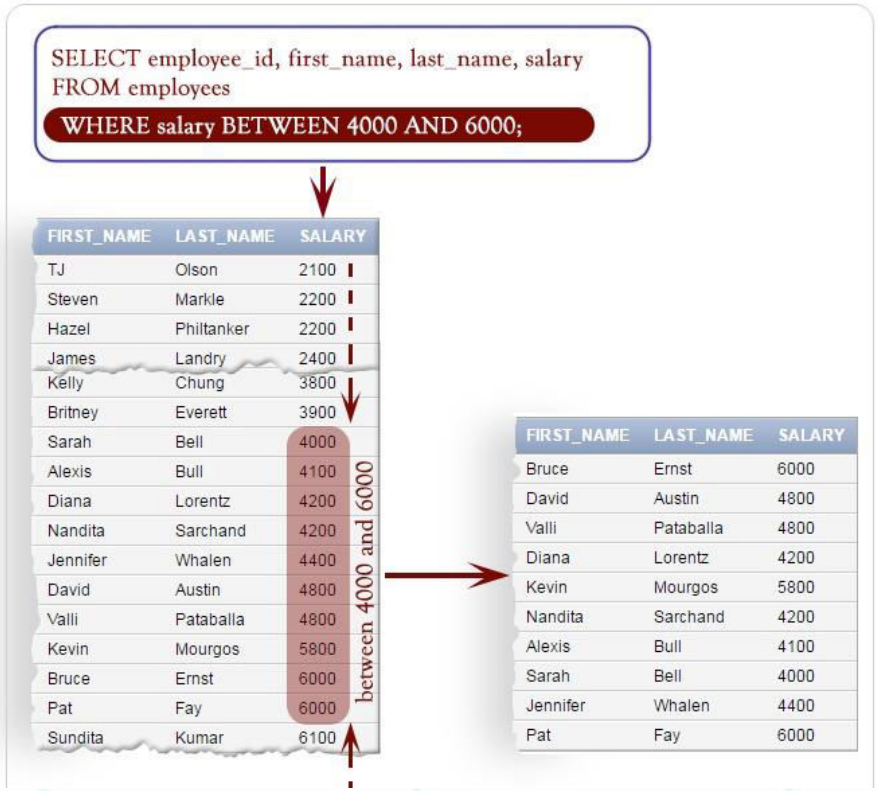
Comando SELECT de 1 Tabla

- SELECT FROM
- WHERE



SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE condiciones
```



FIRST_NAME	LAST_NAME	SALARY
TJ	Olson	2100
Steven	Markle	2200
Hazel	Philtanker	2200
James	Landry	2400
Kelly	Chung	3800
Britney	Everett	3900
Sarah	Bell	4000
Alexis	Bull	4100
Diana	Lorentz	4200
Nandita	Sarchand	4200
Jennifer	Whalen	4400
David	Austin	4800
Valli	Pataballa	4800
Kevin	Mourgos	5800
Bruce	Ernst	6000
Pat	Fay	6000
Sundita	Kumar	6100

FIRST_NAME	LAST_NAME	SALARY
Bruce	Ernst	6000
David	Austin	4800
Valli	Pataballa	4800
Diana	Lorentz	4200
Kevin	Mourgos	5800
Nandita	Sarchand	4200
Alexis	Bull	4100
Sarah	Bell	4000
Jennifer	Whalen	4400
Pat	Fay	6000

Condiciones:

AND, OR, NOT

= igualdad

<> != distinto

>, >= mayor, mayor igual

<, <= menor, menor igual

[NOT] LIKE validar substrings

[NOT] BETWEEN entre rango

[NOT] IN en lista de valores

IS [NOT] NULL es o no es nulo

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE customer_num=104
```

Condiciones por igualdad

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  order_num, order_date, customer_num, paid_date
2  FROM    orders
3  WHERE customer_num=104
```

Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date
1	1001	2015-05-16	104	2015-07-18
2	1003	2015-05-18	104	2015-06-10
3	1011	2015-06-14	104	2015-08-25
4	1013	2015-06-18	104	2015-07-27

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE customer_num=104
        AND order_num >1010
```

Varias condiciones con AND

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  order_num, order_date, customer_num, paid_date
2  FROM    orders
3  WHERE customer_num=104
4                 AND order_num >1010
5
```

Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date
1	1011	2015-06-14	104	2015-08-25
2	1013	2015-06-18	104	2015-07-27

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date,ship_date
FROM    orders
WHERE ship_date IS NULL
```

Condiciones por columnas con NULOS

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  order_num, order_date, customer_num, paid_date,ship_date
2  FROM    orders
3  WHERE ship_date IS NULL
4
```

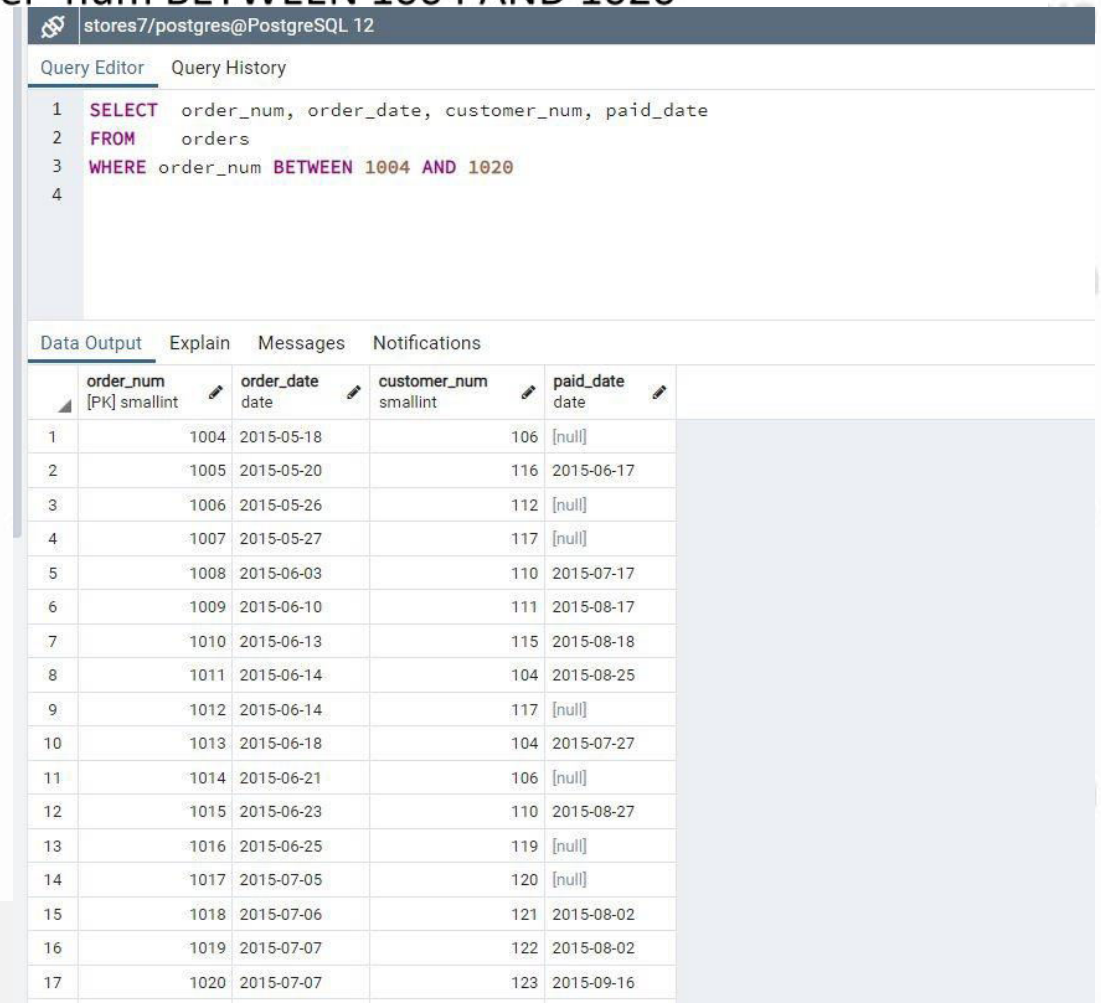
Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date	ship_date date
1	1006	2015-05-26	112	[null]	[null]
2	1014	2015-06-21	106	[null]	[null]
3	1250	2020-01-10	130	[null]	[null]

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE order_num BETWEEN 1004 AND 1020
```



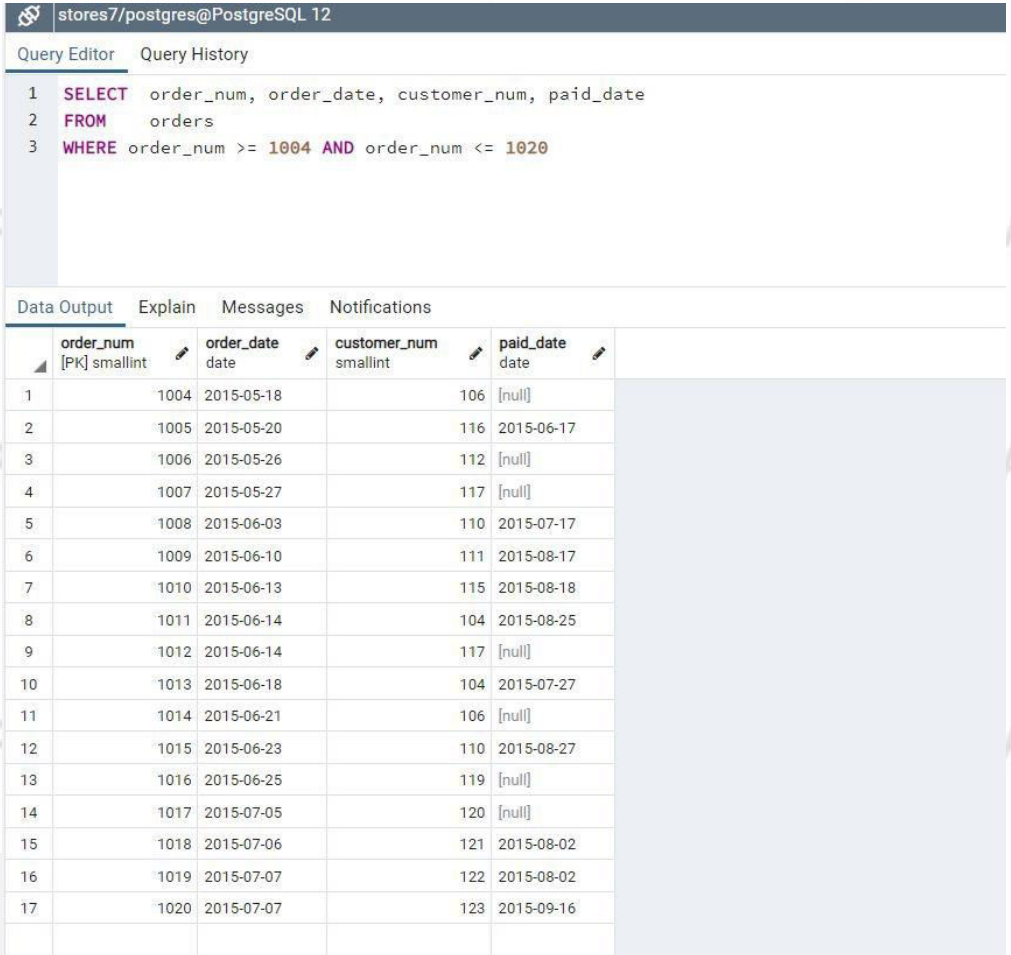
	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date
1	1004	2015-05-18	106	[null]
2	1005	2015-05-20	116	2015-06-17
3	1006	2015-05-26	112	[null]
4	1007	2015-05-27	117	[null]
5	1008	2015-06-03	110	2015-07-17
6	1009	2015-06-10	111	2015-08-17
7	1010	2015-06-13	115	2015-08-18
8	1011	2015-06-14	104	2015-08-25
9	1012	2015-06-14	117	[null]
10	1013	2015-06-18	104	2015-07-27
11	1014	2015-06-21	106	[null]
12	1015	2015-06-23	110	2015-08-27
13	1016	2015-06-25	119	[null]
14	1017	2015-07-05	120	[null]
15	1018	2015-07-06	121	2015-08-02
16	1019	2015-07-07	122	2015-08-02
17	1020	2015-07-07	123	2015-09-16

Condición por un rango de valores para una columna.

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE order_num >= 1004 AND order_num <= 1020
```



	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date
1	1004	2015-05-18	106	[null]
2	1005	2015-05-20	116	2015-06-17
3	1006	2015-05-26	112	[null]
4	1007	2015-05-27	117	[null]
5	1008	2015-06-03	110	2015-07-17
6	1009	2015-06-10	111	2015-08-17
7	1010	2015-06-13	115	2015-08-18
8	1011	2015-06-14	104	2015-08-25
9	1012	2015-06-14	117	[null]
10	1013	2015-06-18	104	2015-07-27
11	1014	2015-06-21	106	[null]
12	1015	2015-06-23	110	2015-08-27
13	1016	2015-06-25	119	[null]
14	1017	2015-07-05	120	[null]
15	1018	2015-07-06	121	2015-08-02
16	1019	2015-07-07	122	2015-08-02
17	1020	2015-07-07	123	2015-09-16

Condición por un rango de valores para una columna. Devuelve lo mismo que el Between.

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE customer_num IN (104,110,127)
```

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  order_num, order_date, customer_num, paid_date
2  FROM    orders
3  WHERE customer_num IN (104,110,127)
4
```

Data Output | Explain | Messages | Notifications

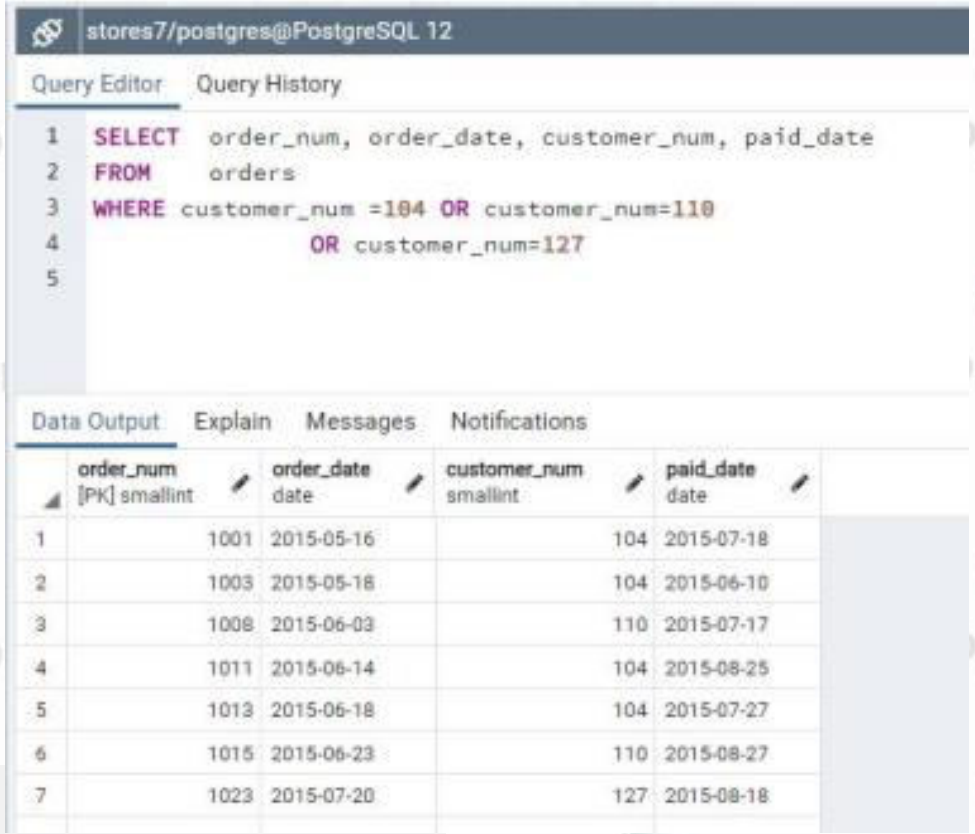
	order_num [PK] smallint	order_date date	customer_num smallint	paid_date date
1	1001	2015-05-16	104	2015-07-18
2	1003	2015-05-18	104	2015-06-10
3	1008	2015-06-03	110	2015-07-17
4	1011	2015-06-14	104	2015-08-25
5	1013	2015-06-18	104	2015-07-27
6	1015	2015-06-23	110	2015-08-27
7	1023	2015-07-20	127	2015-08-18

Condición de una columna por una lista de valores.

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE customer_num =104 OR customer_num=110
        OR customer_num=127
```

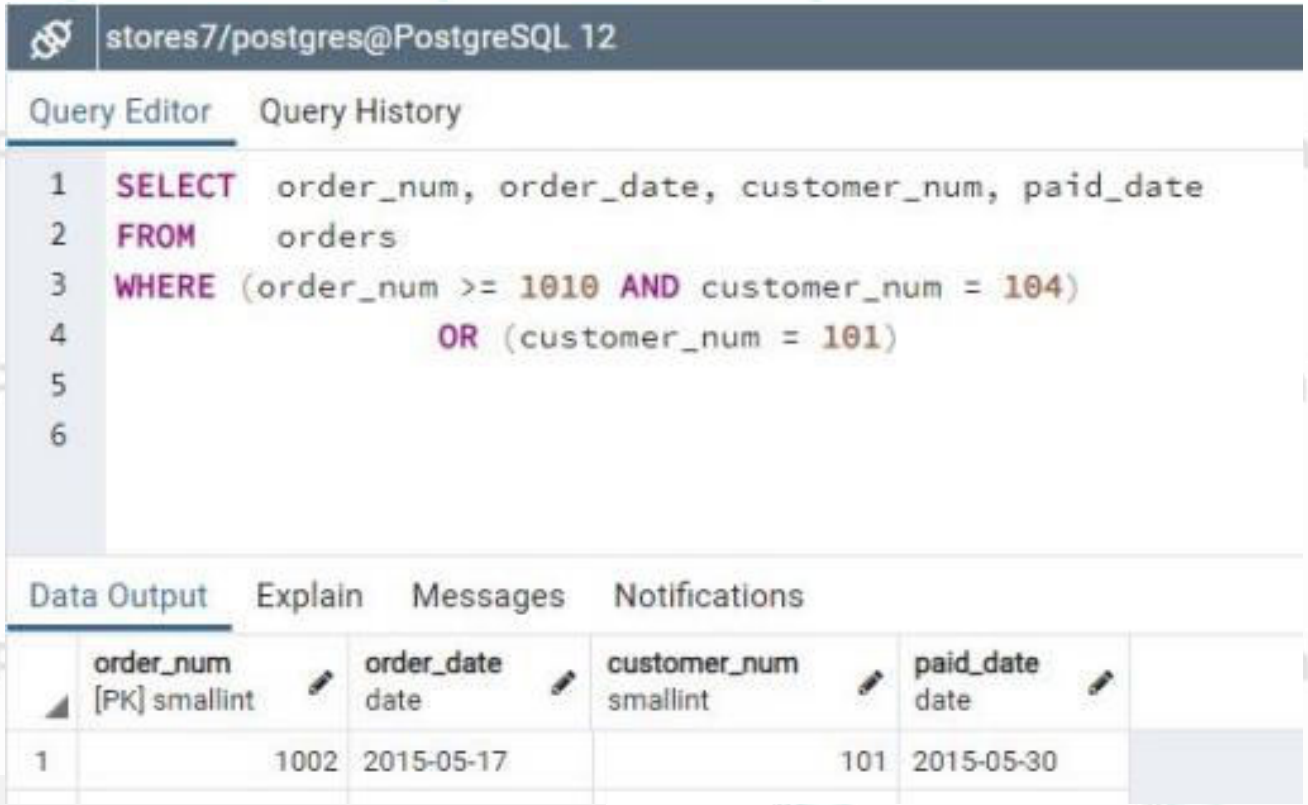


Condición de una columna por una lista de valores. Devuelve igual resultado que el **IN.**

CePETel

SQL - Operador Select

```
SELECT  order_num, order_date, customer_num, paid_date
FROM    orders
WHERE (order_num >= 1010 AND customer_num = 104)
        OR (customer_num = 101)
```



Condiciones con **AND** y **OR**.

CePETel

SQL - Operador Select

```
SELECT  customer_num, lname, fname, company, city
FROM    customer
WHERE condiciones
```

Condiciones con operador LIKE

lname LIKE 'A%' apellidos que comiencen con 'A'.

lname LIKE '%th%' apellidos que contenga 'th' en cualquier parte.

lname LIKE 'A_ _ _' apellidos que comiencen con 'A' y tengan 4 letras

SQL - Operador Select

```
SELECT  customer_num, lname, fname, company, city
FROM    customer
WHERE lname LIKE 'A%'
```

% reemplaza a 0 o más caracteres.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  customer_num, lname, fname, company, city
2  FROM    customer
3  WHERE lname LIKE 'A%'
4
5
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	lname character varying (15)	fname character varying (15)	company character varying (20)	city character varying (15)
1	114	Albertson	Frank	Sporting Place	Redwood City

SQL - Operador Select

```
SELECT  customer_num, lname, fname, company, city
FROM    customer
WHERE lname LIKE '%i%'
```

% reemplaza a 0 o más caracteres.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  customer_num, lname, fname, company, city
2  FROM    customer
3  WHERE lname LIKE '%i%'
4
5
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	lname character varying (15)	fname character varying (15)	company character varying (20)	city character varying (15)
1	101	Pauli	Ludwig	All Sports Supplies	Sunnyvale
2	103	Currie	Philip	Phil s Sports	Palo Alto
3	104	Higgins	Anthony	Play Ball!	Redwood City
4	108	Quinn	Donald	Quinn s Sports	Redwood City
5	109	Miller	Jane	Sport Stuff	Sunnyvale
6	117	Sipes	Arnold	Kids Korner	Redwood City
7	122	O Brian	Cathy	The Sporting Life	Princeton
8	126	Neelie	Eileen	Neelie s Discount Sp	Denver
9	127	Satifer	Kim	Big Blue Bike Shop	Blue Island
10	131	Negrini	Alan	DBLandIT	[null]
11	132	Zaffaroni	Juan	DBLandIT	[null]

En este caso muestra todos los clientes cuyo apellido contenga una i, en cualquier lado.

CePETel

SQL - Operador Select

```
SELECT customer_num, lname, fname, company, city
FROM   customer
WHERE lname LIKE 'P_ _ _ _'
```

_ Reemplaza a 1 solo caracter..

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  customer_num, lname, fname, company, city
2  FROM    customer
3  WHERE lname LIKE 'P____'
4
5
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	lname character varying (15)	fname character varying (15)	company character varying (20)	city character varying (15)
1	101	Pauli	Ludwig	All Sports Supplies	Sunnyvale

En este caso muestra todos los clientes cuyo apellido comience con P y tenga 5 letras.

CePETel

SQL - Operador Select

```
SELECT customer_num, lname, fname, company, city
FROM   customer
WHERE lname LIKE 'P%'
```

Diferencia entre _ y %.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  customer_num, lname, fname, company, city
2  FROM    customer
3  WHERE lname LIKE 'P%'
4
5
```

Data Output | Explain | Messages | Notifications

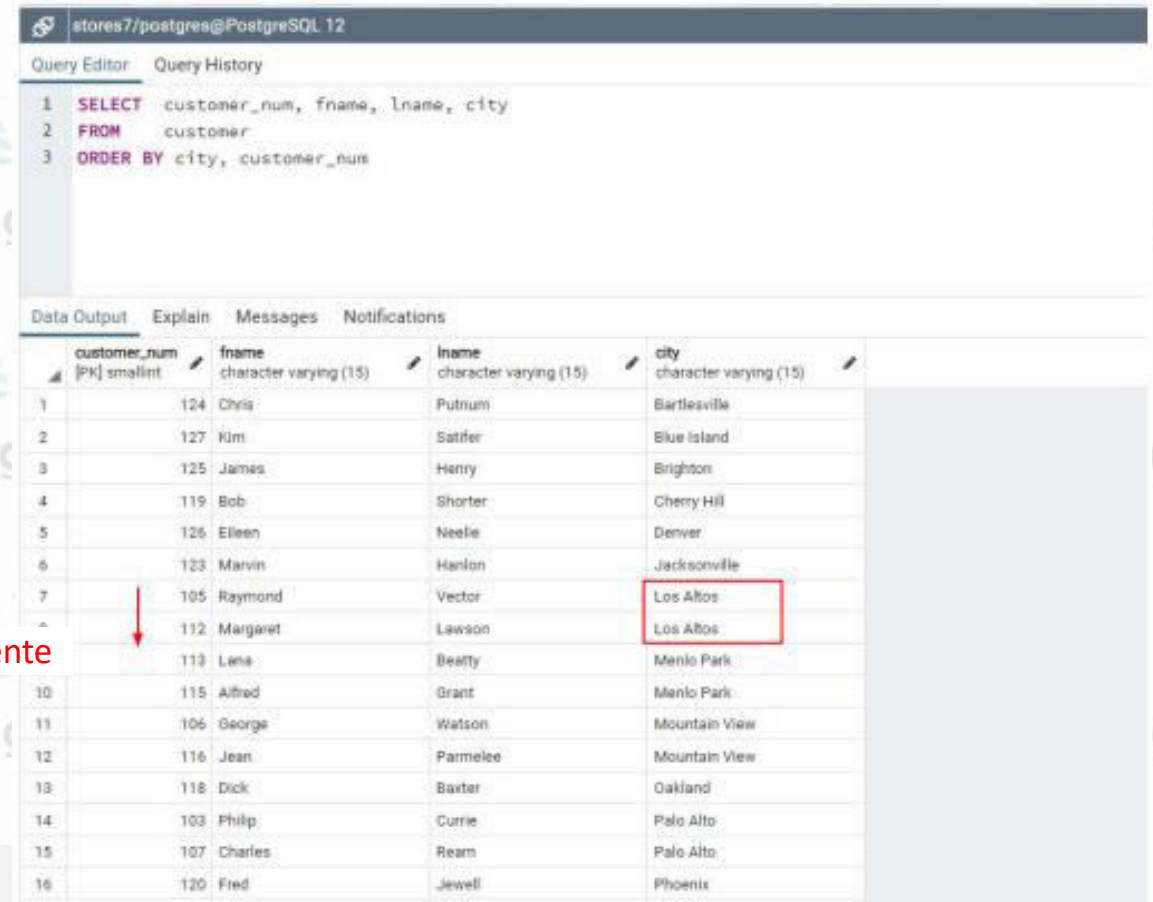
	customer_num [PK] smallint	lname character varying (15)	fname character varying (15)	company character varying (20)	city character varying (15)
1	101	Pauli	Ludwig	All Sports Supplies	Sunnyvale
2	116	Parmelee	Jean	Olympic City	Mountain View
3	124	Putnum	Chris	Putnum s Putters	Bartlesville

En este caso muestra todos los clientes cuyo apellido comience con P, sin importar la cantidad de letras.


Comando SELECT de 1 Tabla
• SELECT FROM
• WHERE
• GROUP BY
• HAVING
• ORDER BY
• LIMIT
DATA MANIPULATION LANGUAGE

SQL - Operador Select

```
SELECT  customer_num, fname, lname, city
FROM    customer
ORDER BY city, customer_num
```



Ascendente

Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Observamos que las filas están ordenadas por ciudad ascendente y a igual ciudad ordena por customer_num también ascendente.

CePETel

SQL - Operador Select

```
SELECT  customer_num, fname, lname, city
FROM    customer
ORDER BY city, customer_num DESC
```

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT  customer_num, fname, lname, city
2  FROM    customer
3  ORDER BY city, customer_num DESC
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	city character varying (15)
1	124	Chris	Putnum	Bartlesville
2	127	Kim	Satifer	Blue Island
3	125	James	Henry	Brighton
4	119	Bob	Shorter	Cherry Hill
5	126	Eileen	Neelie	Denver
6	123	Marvin	Hanlon	Jacksonville
7	112	Margaret	Lawson	Los Altos
8	105	Raymond	Vector	Los Altos
9	115	Alfred	Grant	Menlo Park
10	113	Lana	Beatty	Menlo Park
11	116	Jean	Parmelee	Mountain View
12	106	George	Watson	Mountain View
13	118	Dick	Baxter	Oakland
14	107	Charles	Ream	Palo Alto
15	103	Philip	Currie	Palo Alto
16	128	Frank	Lessor	Phoenix

Descendente

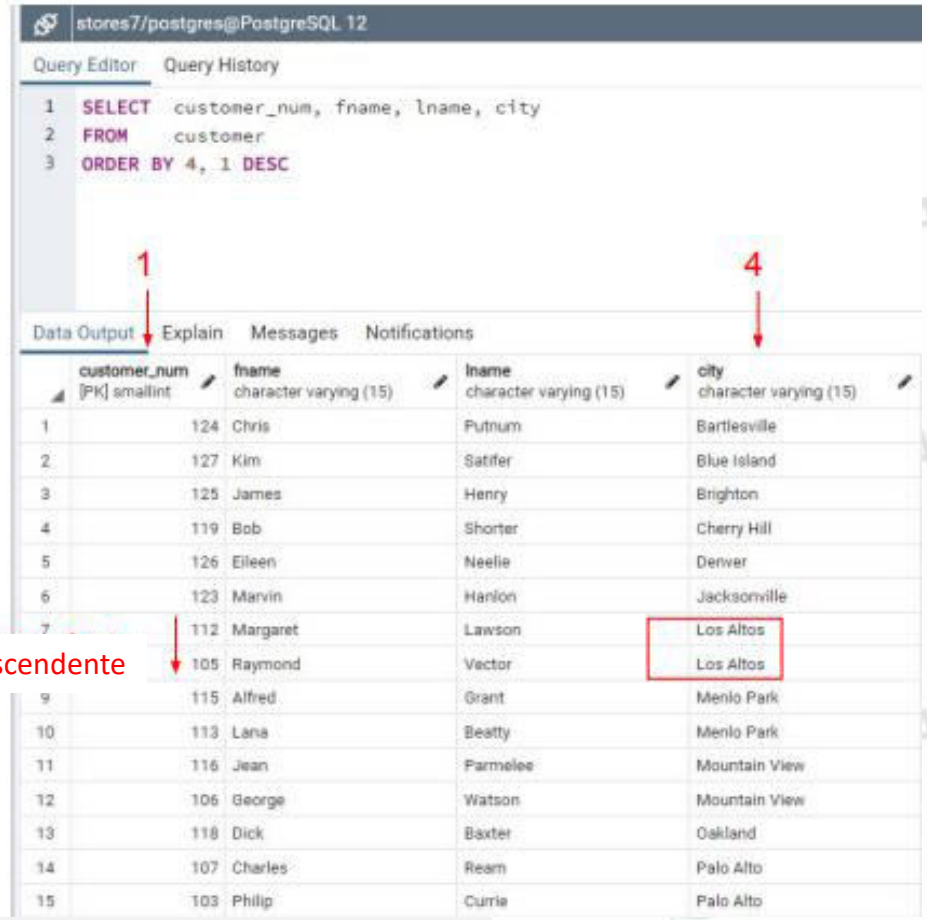
Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Observamos que las filas están ordenadas por ciudad **ASCENDENTE** (default) y a igual ciudad ordena por customer_num **DESCENDENTE**.

CePETel

SQL - Operador Select

```
SELECT  customer_num, fname, lname, city
FROM    customer
ORDER BY 4, 1 DESC
```



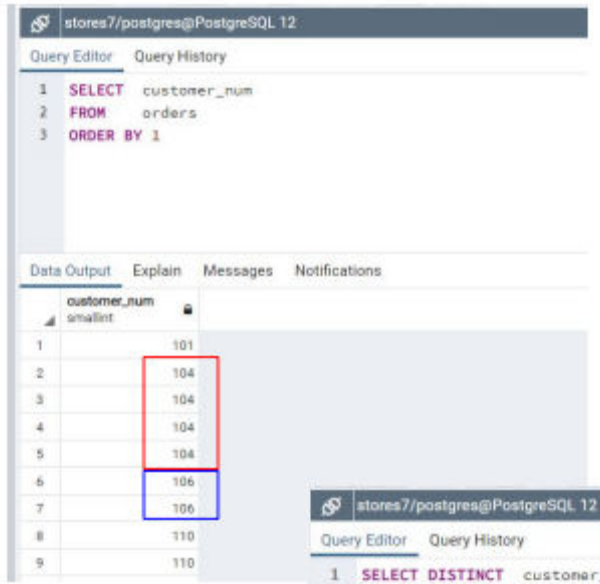
Ordenamiento del resultado de la consulta por una clave o múltiples claves.

Se puede observar en este ejemplo que en el ORDER BY se pueden poner números que indican la posición de la columna en la consulta, en lugar del nombre.

CePETel

SQL - Operador Select

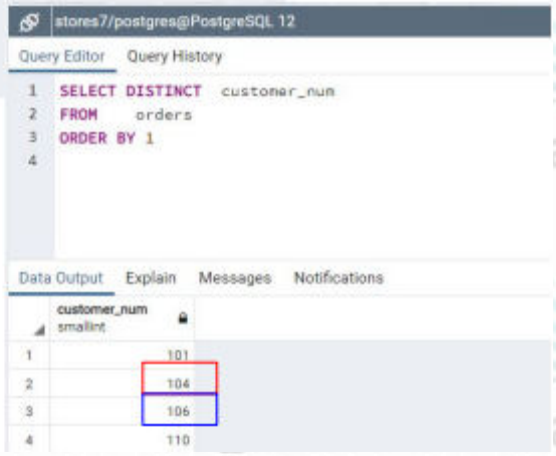
```
SELECT customer_num
FROM   orders
ORDER BY 1
```



Listar valores únicos para una columna, ante una repetición de valores de esa columna.

vs.

```
SELECT DISTINCT customer_num
FROM   orders
ORDER BY 1
```



SQL - Operador Select

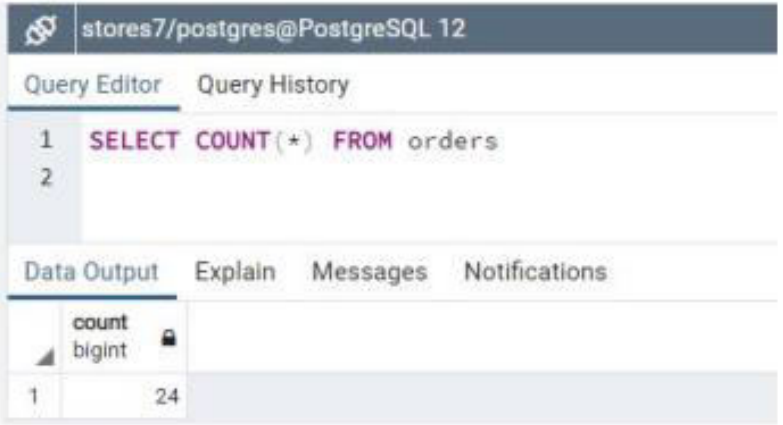
Funciones Agregadas

SUM(columna)	Suma los valores de una columna.
COUNT(*)	Cuenta todas las filas de la tabla.
COUNT(columna)	Cuenta filas con dicha columna No Nula.
COUNT(DISTINCT columna)	Cuenta solo una vez cada valor.
MIN(columna)	Devuelve el valor mínimo de una columna.
MAX(columna)	Devuelve el valor máximo de una columna.
AVG(columna)	Promedia la suma de los valores de una columna con la cantidad de filas.

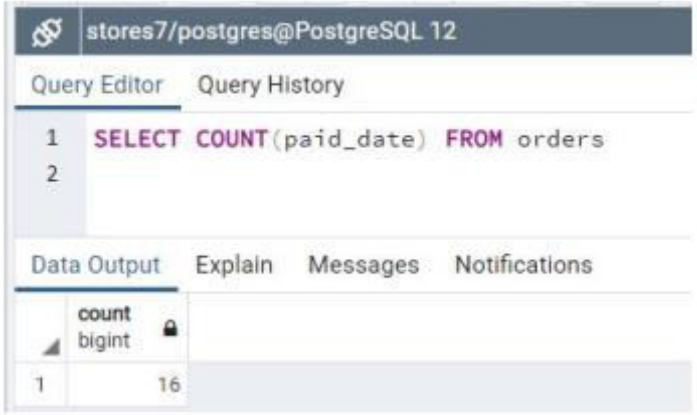
Son funciones que dado un conjunto de datos realizan operaciones agregadas devolviendo un único valor cómo resultado.

SQL - Operador Select

Función COUNT (*) – Mostrar cantidad de Órdenes de Compra.



Función COUNT (paid_date) – Mostrar cantidad de Órdenes de Compra con fecha de pago No Nula.



CePETel

SQL - Operador Select

Función COUNT (DISTINCT customer_num) – Mostrar cuántos clientes nos pusieron Órdenes de Compra.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT COUNT(DISTINCT customer_num) cantidadClientes FROM orders
2
```

Data Output | Explain | Messages | Notifications

	count bigint
1	16

Función SUM (unit_price*quantity) – Función Agregada de fórmulas aritméticas. Mostrar cuánto dinero nos ingresó por los ítems de todas las Órdenes de compra.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT SUM(unit_price*quantity) totalVendido FROM items
```

Data Output | Explain | Messages | Notifications

	totalvendido numeric
1	46255.71

CePETel

SQL - Operador Select

Varias funciones combinadas. Informar primera fecha de Orden de Compra, última fecha de orden de compra y cantidad de órdenes de compra.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT MIN(order_date) primeraCompra,MAX(order_date) ultimaCompra,
2         COUNT(*) cantidadCompras
3    FROM orders
```

Data Output | Explain | Messages | Notifications

	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015-05-16	2020-01-10	24

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Estas cláusulas de agrupamiento son muy poderosas en conjunto con la utilización de funciones Agregadas

SELECT * | lista de columnas

FROM nom_tabla | lista de tablas

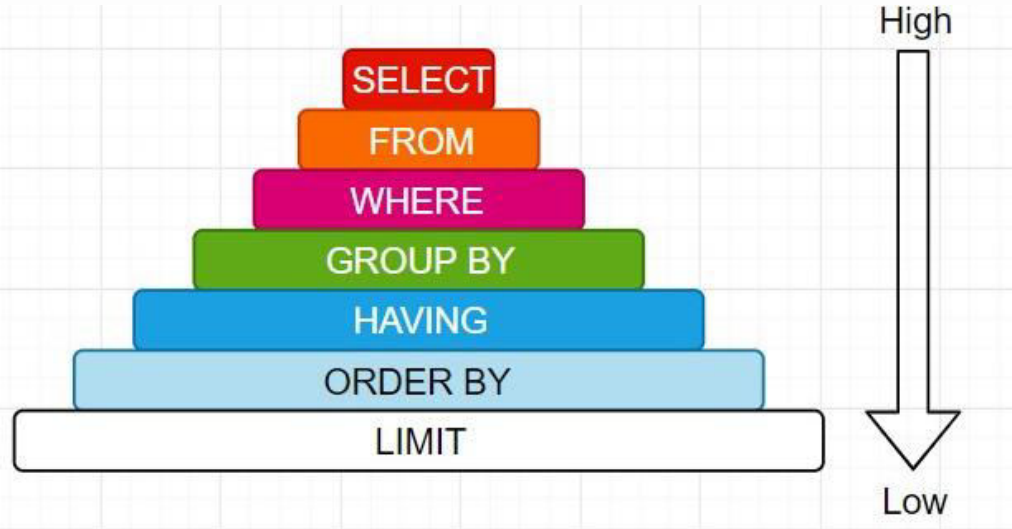
WHERE condiciones ó filtros

GROUP BY columnas clave de agrupamiento

HAVING condiciones sobre lo agrupado

ORDER BY columnas clave de ordenamiento

LIMIT limita la cantidad de filas a mostrar



CePETel

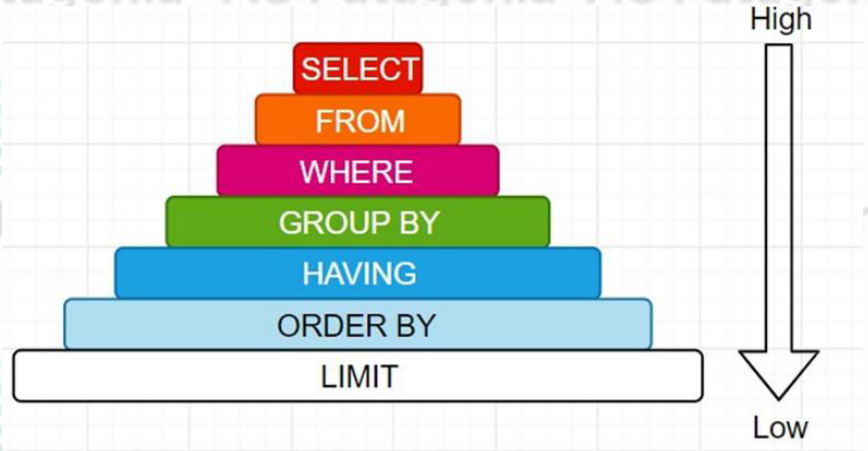
SQL - Operador Select

Cláusulas GROUP BY y HAVING

La cláusula ***GROUP BY*** sirve para agrupar filas a partir de una o varias columnas tomadas como clave.

Su potencial máximo se logra cuando lo combinamos con funciones agregadas, ya que rápidamente y con pequeños cambios en la consulta podríamos tener resultados sumarizados o agregados por distintas dimensiones.

Obviamente que, dependiendo del volumen de datos, esto implicaría implementar algún mecanismo para agilizar la consulta, como por ejemplo índices.



CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Volviendo a esta consulta, vemos que nos muestra, el primer orden de compra, la última orden y la cantidad de órdenes de compra.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT MIN(order_date) primeraCompra,MAX(order_date) ultimaCompra,
2         COUNT(*) cantidadCompras
3    FROM orders
```

Data Output | Explain | Messages | Notifications

	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015-05-16	2020-01-10	24

Vamos a modificar esta consulta, obteniendo la misma información, pero agrupada por cliente.

CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Vamos a modificar esta consulta, obteniendo la misma información, pero agrupada por cliente.

```
SELECT customer_num, MIN(order_date) primeraCompra,
       MAX(order_date) ultimaCompra,
       COUNT(*) cantidadCompras
FROM orders
```

SQL - Operador Select

Cláusulas GROUP BY y HAVING

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT customer_num, MIN(order_date) primeraCompra,MAX(order_date) ultimaCompra,
2         COUNT(*) cantidadCompras
3    FROM orders
```

Por qué da error?

Data Output | Explain | Messages | Notifications

```
ERROR:  column "orders.customer_num" must appear in the GROUP BY clause or be used in an aggregate function
LINE 1: SELECT customer_num, MIN(order_date) primeraCompra,MAX(order...
               ^
SQL state: 42803
Character: 8
```

Cuando queremos combinar en un **SELECT** atributos desagrupados con funciones agregadas, es necesario que dichos atributos sean incluidos en la cláusula **GROUP BY.**

No se pueden combinar atributos desagregados con funciones agregados, es necesario agruparlos en la cláusula **GROUP BY**.

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Vamos a modificar esta consulta, obteniendo la misma información, pero agrupada por cliente.

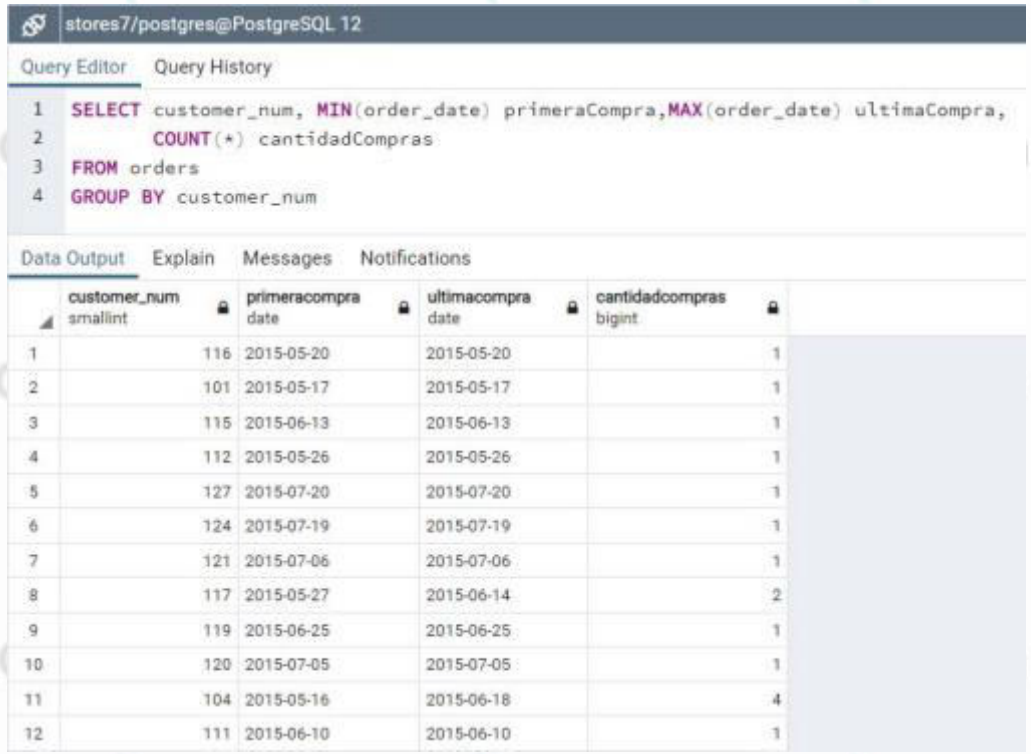
```
SELECT    customer_num, MIN(order_date) primeraCompra,
          MAX(order_date) ultimaCompra,
          COUNT(*) cantidadCompras
FROM      orders
GROUP BY  customer_num
```

En la cláusula GROUP BY ponemos los atributos no agregados que están en la cláusula SELECT.

SQL - Operador Select

Cláusulas GROUP BY y HAVING

```
SELECT      customer_num, MIN(order_date) primeraCompra,
            MAX(order_date) ultimaCompra,
            COUNT(*) cantidadCompras
FROM        orders
GROUP BY    customer_num
```



	customer_num smallint	primeracompra date	ultimacompra date	cantidadcompras bigint
1	116	2015-05-20	2015-05-20	1
2	101	2015-05-17	2015-05-17	1
3	115	2015-06-13	2015-06-13	1
4	112	2015-05-26	2015-05-26	1
5	127	2015-07-20	2015-07-20	1
6	124	2015-07-19	2015-07-19	1
7	121	2015-07-06	2015-07-06	1
8	117	2015-05-27	2015-06-14	2
9	119	2015-06-25	2015-06-25	1
10	120	2015-07-05	2015-07-05	1
11	104	2015-05-16	2015-06-18	4
12	111	2015-06-10	2015-06-10	1

Observamos de los clientes que compraron, tenemos la cantidad de Órdenes de compra, la fecha de la primer y ultima compra.

CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Observamos que, haciendo un pequeño cambio a la consulta, tenemos la cantidad de Órdenes de compra, la fecha de la primer y ultima compra, agrupados por año y mes.

```
SELECT      DATE_PART('year',order_date) anio,
            DATE_PART('month',order_date) mes, MIN(order_date)
            primeraCompra, MAX(order_date) ultimaCompra,
            COUNT(*) cantidadCompras
FROM  orders
GROUP BY    DATE_PART('year',order_date),
            DATE_PART('month',order_date)
```

SQL - Operador Select

Cláusulas GROUP BY y HAVING

```
SELECT    DATE_PART('year',order_date) anio,
          DATE_PART('month',order_date) mes, MIN(order_date)
          primeraCompra, MAX(order_date) ultimaCompra,
          COUNT(*) cantidadCompras
FROM      orders
GROUP BY  DATE_PART('year',order_date),
          DATE_PART('month',order_date)
```

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT DATE_PART('year',order_date) anio,
2         DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra,
3         MAX(order_date) ultimaCompra, COUNT(*) cantidadCompras
4  FROM orders
5  GROUP BY DATE_PART('year',order_date),DATE_PART('month',order_date)
```

Data Output | Explain | Messages | Notifications

	anio double precision	mes double precision	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015	6	2015-06-03	2015-06-25	9
2	2020	1	2020-01-10	2020-01-10	1
3	2015	5	2015-05-16	2015-05-27	7
4	2015	7	2015-07-05	2015-07-20	7

SQL - Operador Select

Cláusulas GROUP BY y HAVING

```
SELECT    DATE_PART('year',order_date) anio,
          DATE_PART('month',order_date) mes, MIN(order_date)
          primeraCompra, MAX(order_date) ultimaCompra,
          COUNT(*) cantidadCompras
FROM      orders
GROUP BY  DATE_PART('year',order_date),
          DATE_PART('month',order_date)
ORDER BY  1,2
```

Ordenamos la salida por año y mes.

CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Tomamos la misma consulta del slide anterior y ordenamos la salida por CantidadCompras de forma DESCENDENTE.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT DATE_PART('year',order_date) anio,
2         DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra,
3         MAX(order_date) ultimaCompra, COUNT(*) cantidadCompras
4  FROM orders
5  GROUP BY DATE_PART('year',order_date),DATE_PART('month',order_date)
6  ORDER BY cantidadCompras DESC
7
```

Data Output | Explain | Messages | Notifications

	anio double precision	mes double precision	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015	6	2015-06-03	2015-06-25	9
2	2015	5	2015-05-16	2015-05-27	7
3	2015	7	2015-07-05	2015-07-20	7
4	2020	1	2020-01-10	2020-01-10	1

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Tomamos la misma consulta del slide anterior y ordenamos la salida por CantidadCompras de forma DESCENDENTE.

Aplicamos la cláusula HAVING para filtrar las filas de nuestro conjunto de datos de salida cuya cantidad de compras sea menor a 9.

```
SELECT          DATE_PART('year',order_date) anio,
                DATE_PART('month',order_date) mes, MIN(order_date)
                primeraCompra, MAX(order_date) ultimaCompra,
                COUNT(*)cantidadCompras
FROM orders
GROUP BY        DATE_PART('year',order_date),
                DATE_PART('month',order_date)
HAVING  COUNT(*) < 9
ORDER BY cantidadCompras DESC
```

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Aplicamos la cláusula HAVING para filtrar las filas de nuestro conjunto de datos de salida cuya cantidad de compras sea menor a 9.

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT DATE_PART('year',order_date) anio,
2  DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra, COUNT(*) cantidadCompras
3  FROM orders
4  GROUP BY DATE_PART('year',order_date), DATE_PART('month',order_date)
5  HAVING  COUNT(*) < 9
6  ORDER BY cantidadCompras DESC
7
```

Data Output | Explain | Messages | Notifications

	anio double precision	mes double precision	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015	5	2015-05-16	2015-05-27	7
2	2015	7	2015-07-05	2015-07-20	7
3	2020	1	2020-01-10	2020-01-10	1

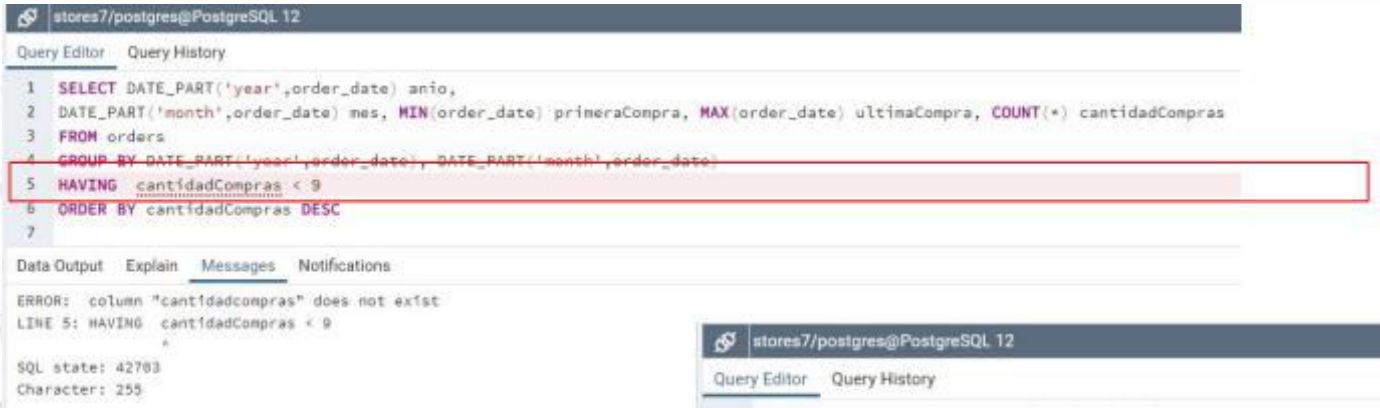
En el **HAVING** sólo podemos poner cómo condiciones funciones agregadas o atributos agrupados en el **GROUP BY.**

CePETel

SQL - Operador Select

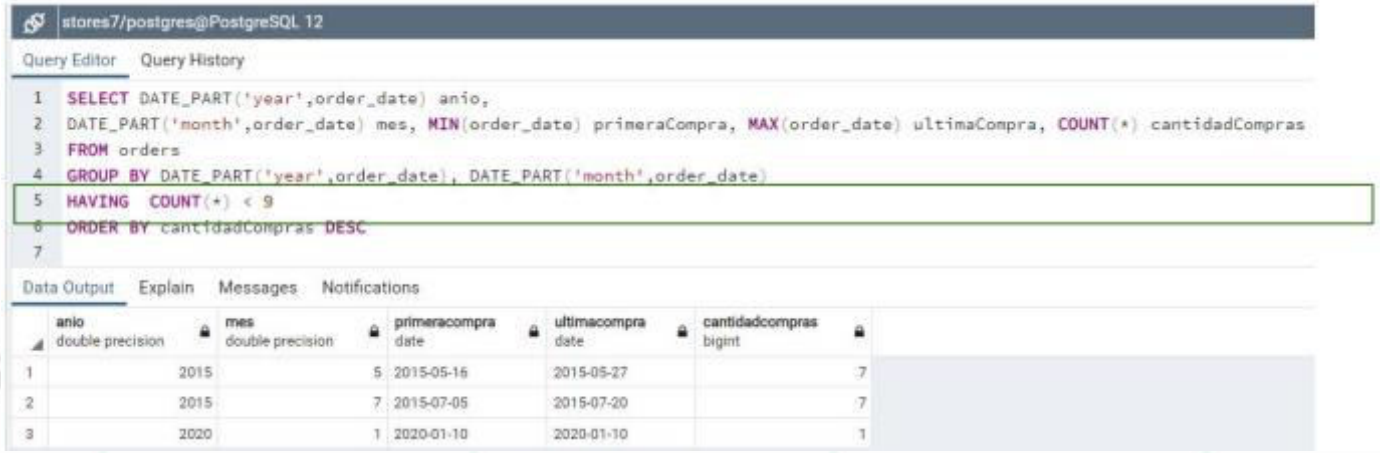
Cláusulas GROUP BY y HAVING

Cláusula HAVING con error ya que no se pueden poner alias o labels, sino que se debe poner la función agregada.



Poner una función agregada en lugar del label o alias.

Forma correcta de realizarlo:



CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Cláusula **HAVING** con error ya que no se pueden atributos que no estén en la cláusula **GROUP BY**

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT DATE_PART('year',order_date) anio,
2  DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra, COUNT(*) cantidadCompras
3  FROM orders
4  GROUP BY DATE_PART('year',order_date), DATE_PART('month',order_date)
5  HAVING  order_num >1010
6  ORDER BY cantidadCompras DESC
7
```

Data Output | Explain | Messages | Notifications

```
ERROR:  column "orders.order_num" must appear in the GROUP BY clause or be used in an aggregate function
LINE 5: HAVING  order_num >1010
                ^
SQL state: 42803
Character: 255
```

Poner la condición en la cláusula **WHERE**.

Forma correcta de realizarlo

stores7/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT DATE_PART('year',order_date) anio,
2  DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra, COUNT(*) cantidadCompras
3  FROM orders
4  WHERE  order_num >1010
5  GROUP BY DATE_PART('year',order_date), DATE_PART('month',order_date)
6  ORDER BY cantidadCompras DESC
7
```

Data Output | Explain | Messages | Notifications

	anio double precision	mes double precision	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015	7	2015-07-05	2015-07-20	7
2	2015	6	2015-06-14	2015-06-25	6
3	2020	1	2020-01-10	2020-01-10	1

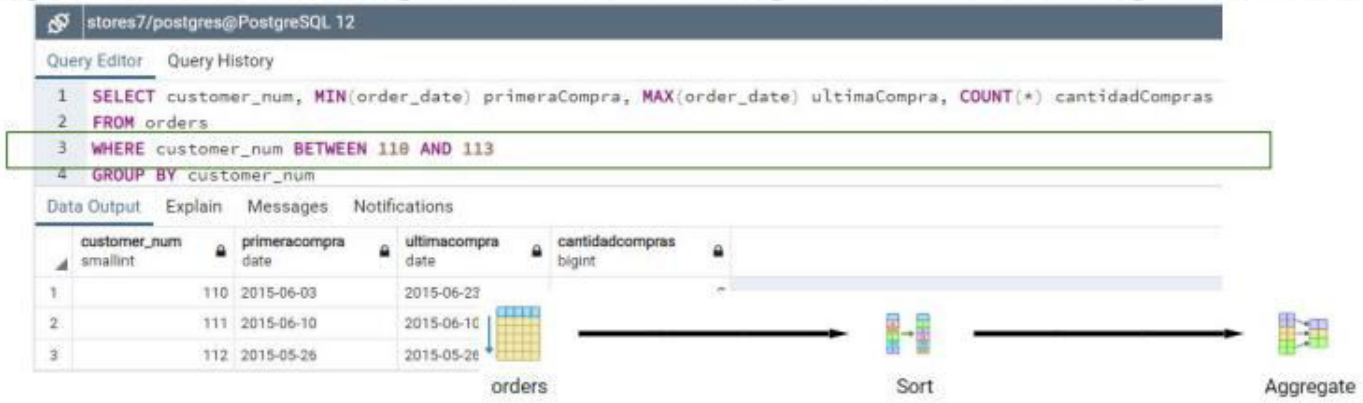
CePETel

SQL - Operador Select

Cláusulas GROUP BY y HAVING

Cláusula **HAVING** con una condición sobre atributos existentes en la cláusula **GROUP BY** vs. igual condición en la cláusula **WHERE**.

Poner la condición en la cláusula **WHERE**.



CePETel

SQL - Operador Select

Cláusula LIMIT

Modificamos la consulta de slide anteriores para obtener los datos del año y mes con la mayor cantidad de órdenes:

```
SELECT      DATE_PART('year',order_date) anio,
            DATE_PART('month',order_date) mes,
            MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra,
            COUNT(*)cantidadCompras
FROM orders
GROUP BY    DATE_PART('year',order_date), DATE_PART('month',order_date)
ORDER BY    cantidadCompras DESC
LIMIT 1
```

```
1  SELECT DATE_PART('year',order_date) anio,
2  DATE_PART('month',order_date) mes, MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra, COUNT(*)  cantidadCompras
3  FROM orders
4  GROUP BY DATE_PART('year',order_date), DATE_PART('month',order_date)
5  ORDER BY cantidadCompras DESC
6  LIMIT 1
```

Data Output | Explain | Messages | Notifications

	anio double precision	mes double precision	primeracompra date	ultimacompra date	cantidadcompras bigint
1	2015	6	2015-06-03	2015-06-25	9

CePETel

SQL - Operador Select

Cláusula LIMIT

Otro caso para el LIMIT, obtener los tres clientes con mayor cantidad de órdenes de compra.

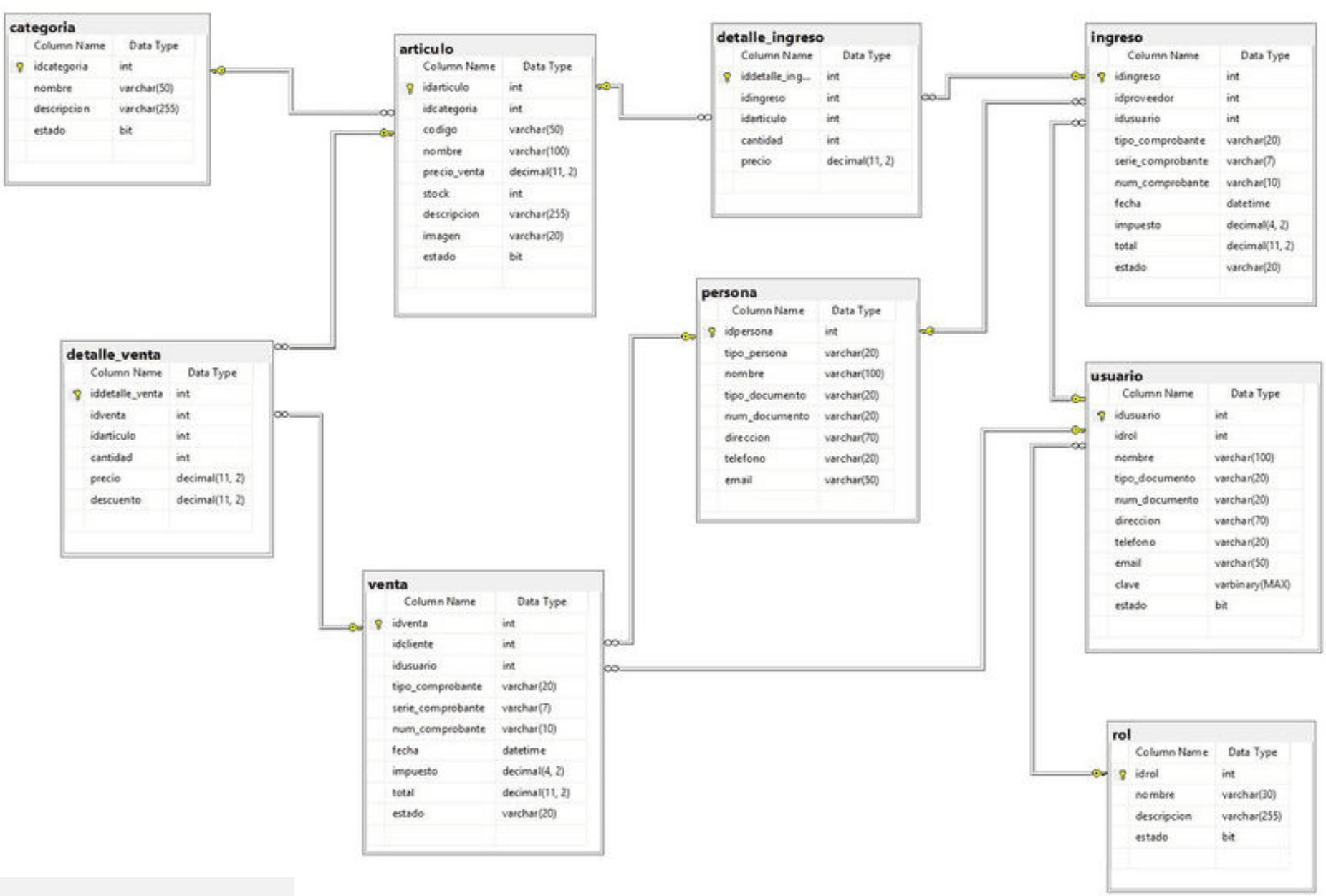
```
SELECT      customer_num, MIN(order_date) primeraCompra, MAX(order_date)
            ultimaCompra,
            COUNT(*)cantidadCompras
FROM orders
GROUP BY    customer_num
ORDER BY    cantidadCompras DESC
LIMIT 3
```

```
1  SELECT customer_num, MIN(order_date) primeraCompra, MAX(order_date) ultimaCompra, COUNT(*)  cantidadCompras
2  FROM orders
3  GROUP BY customer_num
4  ORDER BY cantidadCompras DESC
5  LIMIT 3
6
```

Data Output | Explain | Messages | Notifications

	customer_num smallint	primeracompra date	ultimacompra date	cantidadcompras bigint
1	104	2015-05-16	2015-06-18	4
2	117	2015-05-27	2015-06-14	2

CePETel



TP N° 4: SQL 1 - Select

Conectese al siguiente link y descargue la DVD Rental Sample Database

https://www.postgresqltutorial.com/postgresql-getting-started/postgresql-sample-database/

Download the PostgreSQL sample database

You can download the PostgreSQL DVD Rental sample database via the following link:

Download DVD Rental Sample Database

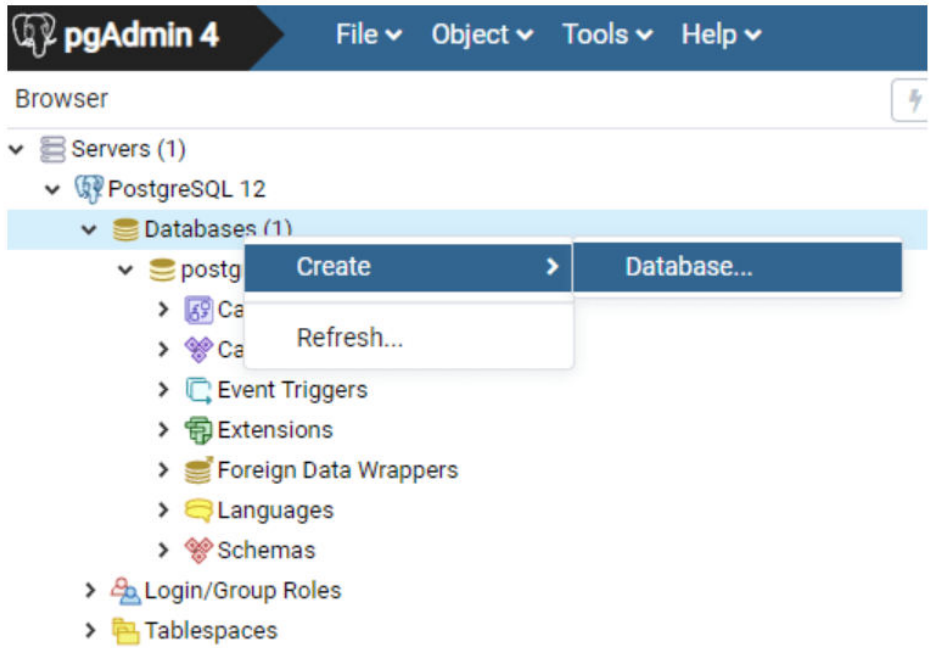
The database file is in `zip` format (`dvdrental.zip`) so you need to extract it to `dvdrental.tar` before loading the sample database into the PostgreSQL database server.

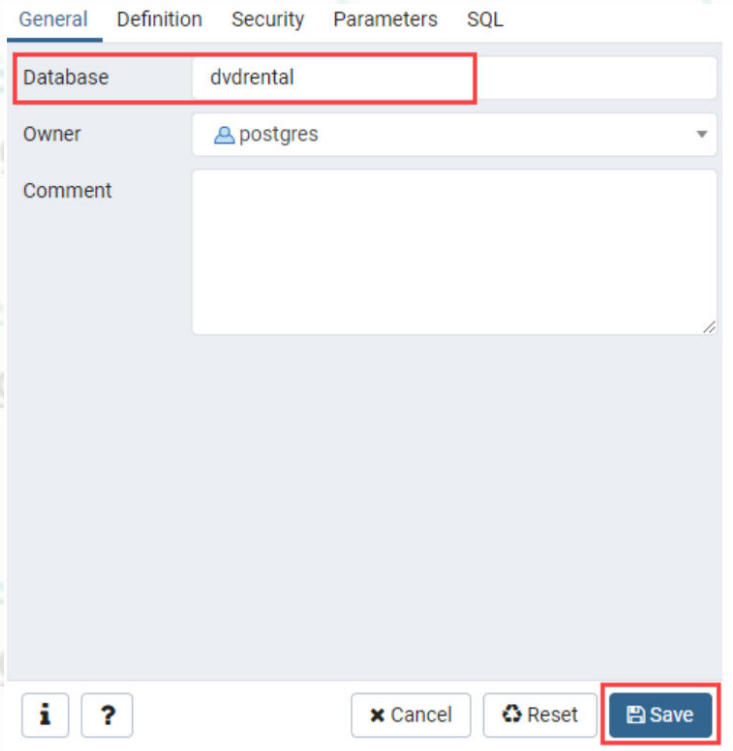
Extraiga del archivo dvdrental.zip el archivo dvdrental.tar en alguna parte de su equipo.

TP N° 4: SQL 1 - Select

CARGA DE LA DATABASE DVD RENTAL EN POSTGRES USANDO pgAdmin

Primero ingrese a pgAdmin y cree una base de datos y llámela “dvdrental”, ingresa a la base y guardela

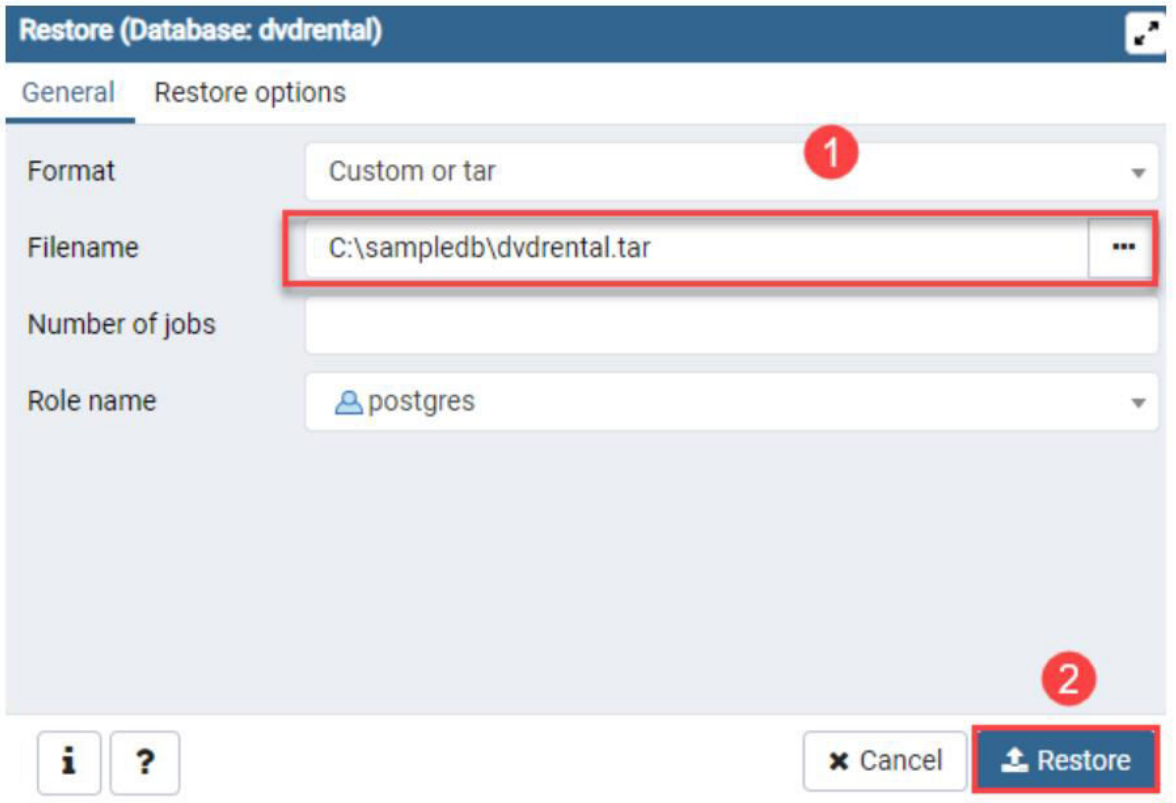




CePETel

TP N° 4: SQL 1 - Select

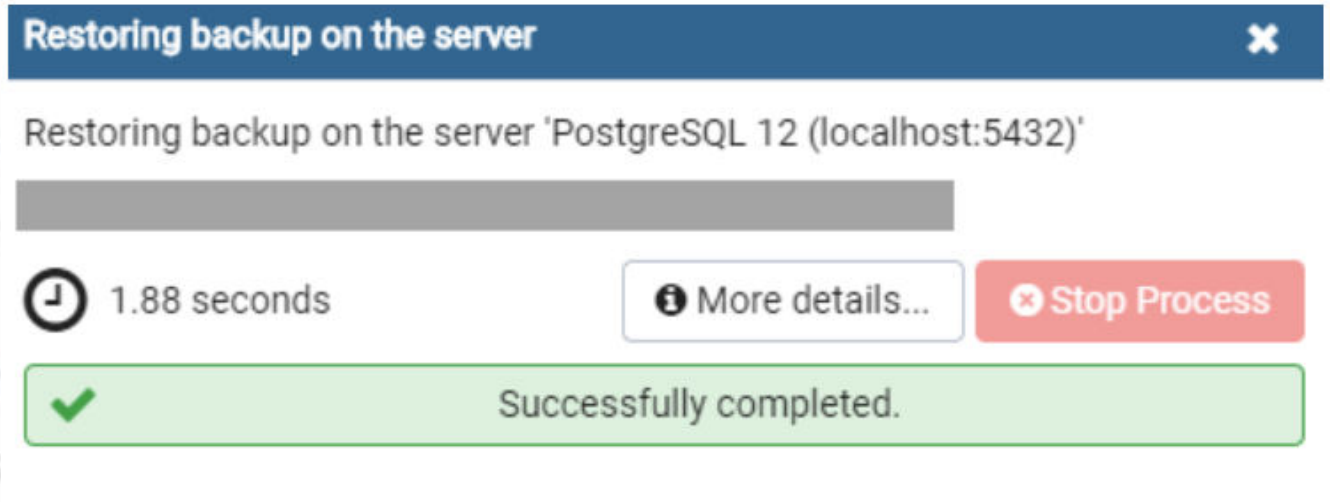
Haga right-click sobre la database dvdrental y seleccione **Restore…** e ingrese la direccion donde se encuenta la base de datos, por ejemplo **c:\sampledb\dvdrental.tar** y haga click en el boton **Restore**

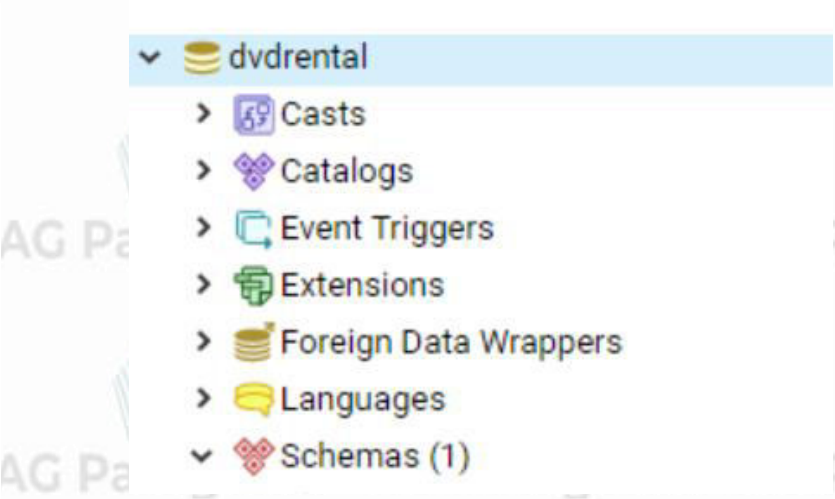


CePETel

TP N° 4: SQL 1 - Select

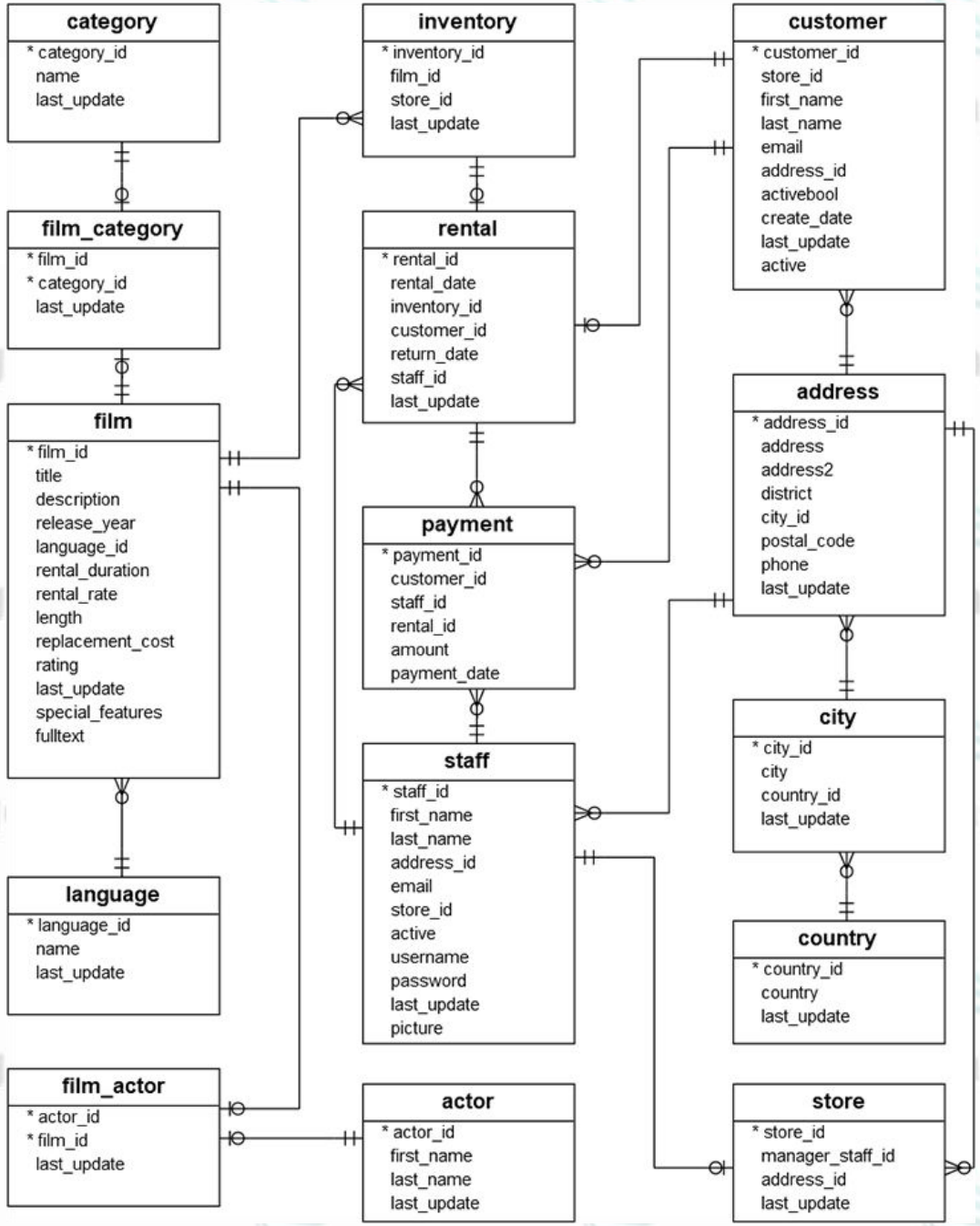
Si todo anduvo bien el proceso de restore se completara en unos pocos segundos y se mostrara el siguiente dialogo cuando termine, y podra abrir la base de datos “dvdrental”





En caso de error, setear en Files>Preferences>Paths>Binary Paths>PostgreSQL Binary Path la direccion donde se encuentra instalado PostreSQL (ver imagen)

TP N° 4: SQL 1 - Select



CePETel

TP N° 4: SQL 1 - Select

1. Obtener un listado de todos los clientes (número, nombre, apellido y email

2. Obtener un listado de actores cuyo primer nombre sea Julia

3. Obtener una lista de actors cuyo primer nombre sea Chris, Cameron o Cuba.

4. Seleccionar la fila de la tabla cliente cuyo nombre complete sea Jamie Rice.

5. Obtener el monto (amount) y la fecha de pago (payment_date) de la table payment donde el monto pagado fue menos que $1.

6. Cuales son los IDs y apellido de los 3 ultimos cliens que devolvieron una pelicula alquilada?

7. Cuanto films son calificados (rating) NC-17? Y cuanto estan calificados PG or PG-13?

CePETel

TP N° 4: SQL 1 - Select

8. Cuantos clientes diferentes hay en la tabla “Rental”?

9. El costo promedio de reemplazo (average replacement cost) de un film varia de acuerdo a su rating?

10. Hay algunos clientes con el mismo apellido (last name)?

11. Cual el precio promedio de alquiler de peliculas? Puede mostrar el resultado con 2 decimales?

12. ¿Qué película (id) tiene más actores?

13. ¿Qué actor (id) aparece en más películas?

14. Selecciona los 10 actores que tengan los nombres más largos (nombre y apellido combinados).

CePETel

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

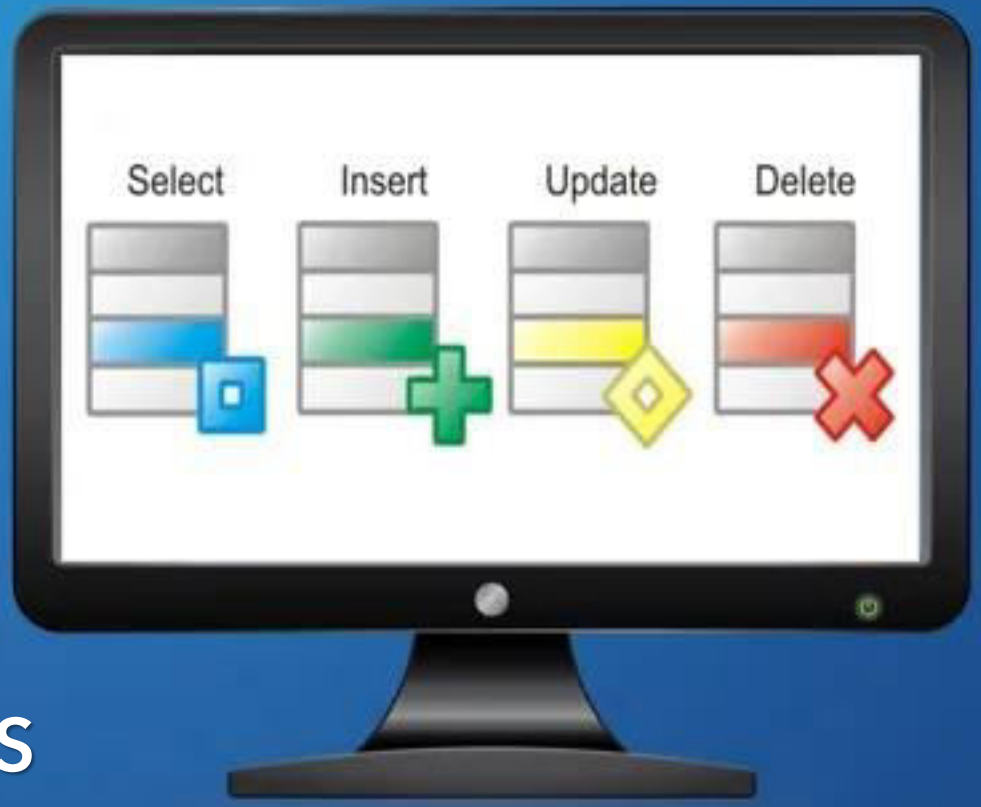
Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 5:
DML - (Update – Insert - Delete)

Data Manipulation Language

- INSERT
- DELETE
- UPDATE
- TRANSACCIONES

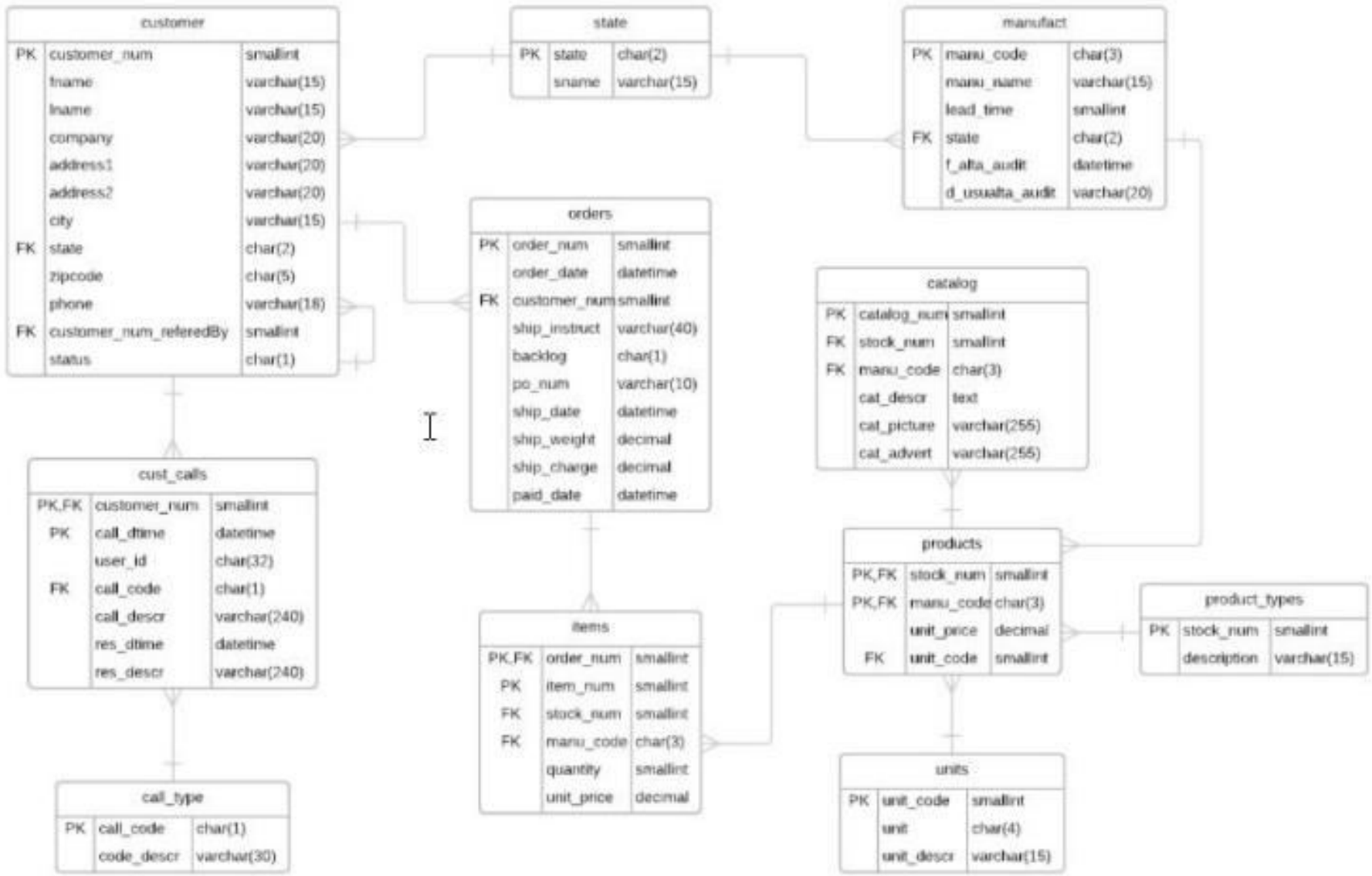


SQL Insert Values Into a Table

Syntax -

```
INSERT INTO table_name
(column1_name, column2_name,... )
VALUES ( column1_value, column2_value, ... )
```

Bases de Datos de Ejemplo



***Este
***This

CePETel

SQL - Structured Query Language

Video: Conociendo la historia SQL

DML - Data Manipulation Language

- Insert
- Update
- Delete

SQLQuery2.sql ...ks (dev (56))* | SQLQuery3.sql ...ks (dev (57))*

```
SELECT *
FROM StudentTotalMarks
ORDER BY StudentID
GO
```

New Result

	StudentID	StudentMarks
1	1	255
2	3	225
3	4	25
4	5	25

Original Result

	StudentID	StudentMarks	
1	1	230	Updated
2	2	255	Deleted
3	3	200	Updated

SQL - Operador INSERT

Se inserta una sola fila en la tabla

```
INSERT INTO nom_tabla [(lista de columnas)] (*) (**)
VALUES (lista de valores) (**)
```

Se insertan múltiples filas en la tabla

```
INSERT INTO nom_tabla [(lista de columnas)] (*) (**)
VALUES (lista de valores),
       (lista de valores),
       (lista de valores)
```

```
INSERT INTO nom_tabla [(lista de columnas)]
SELECT .... (***)
```

- `SELECT ... FROM ... WHERE ...`
- `INSERT INTO ... VALUES ...`
- `UPDATE ... SET ... WHERE ...`
- `DELETE FROM ... WHERE ...`

(*) La lista de columnas entre corchetes significa que es opcional.

(**) la lista de columnas y valores tienen como separador una coma.

(***) El select debe devolver una lista columnas similar a la que espera recibir el INSERT.

SQL - Operador INSERT

```
INSERT INTO product_types
VALUES (375,'Short Baño')
```

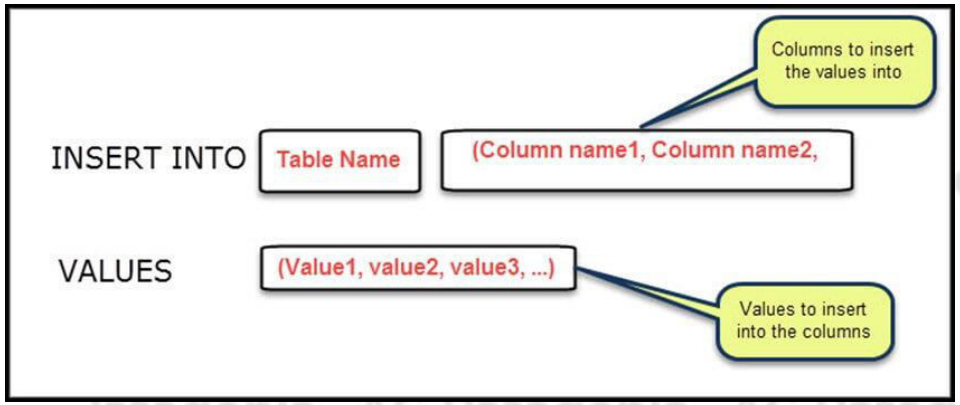
Inserta en la tabla tipo de productos un nuevo tipo de producto.

Query Editor | Query History

```
1  INSERT INTO product_types VALUES (375,'Short Baño')
```

Data Output | Explain | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 899 msec.
```



Query Editor | Query History

```
1  SELECT * FROM  product_types WHERE stock_num=375
```

Data Output | Explain | Messages | Notifications

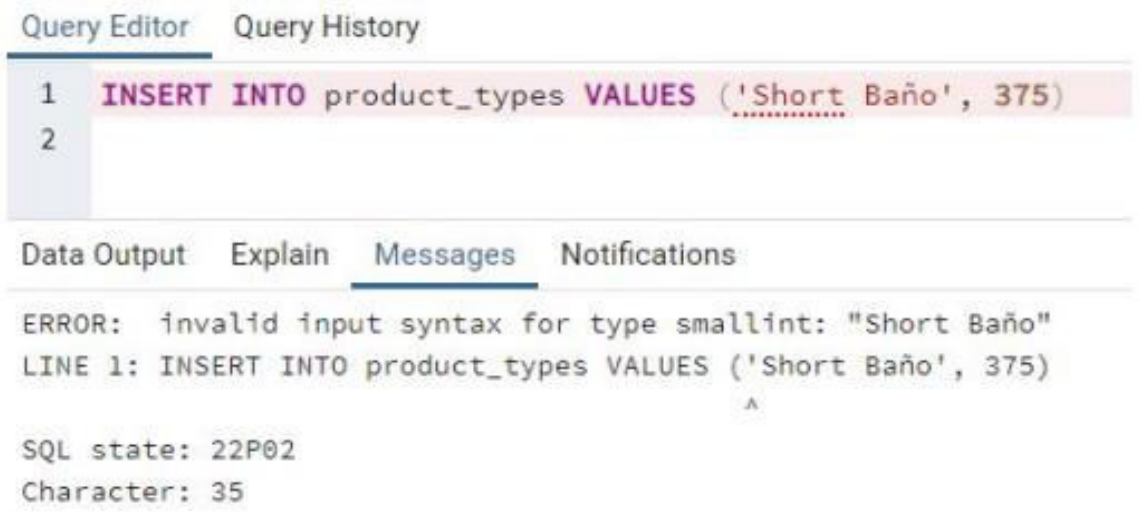
	stock_num [PK] smallint	description character varying (15)
1	375	Short Baño

***Este c
***This d

CePETel

SQL - Operador INSERT

```
INSERT INTO product_types
VALUES ('Short Baño', 375)
```



El problema de no poner la lista de columnas a insertar es que mi lista de valores debe respetar 100 el orden de las columnas en la tabla, acoplando mi consulta al modelo.

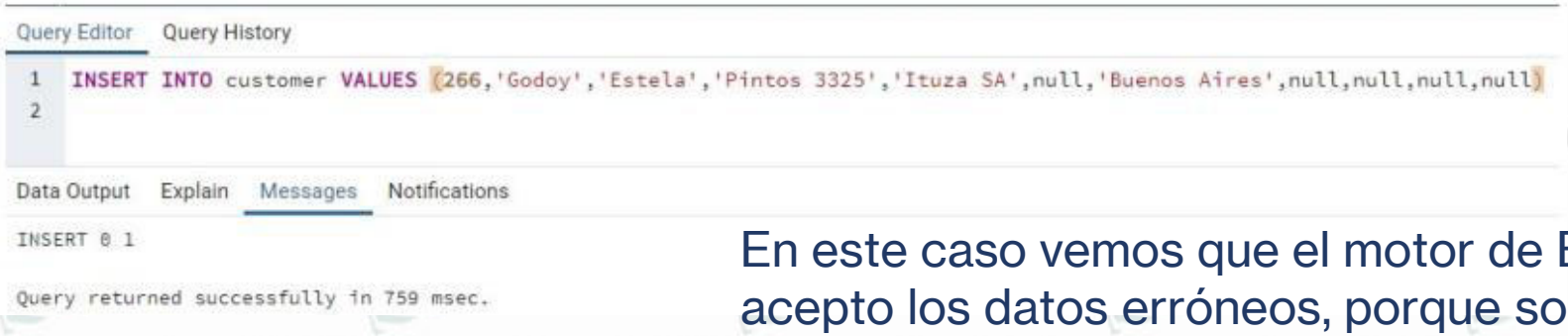
En este ejemplo, vemos que al querer insertar los valores fuera de orden, el comando falla debido a que espera un valor smallint en la primera posición.

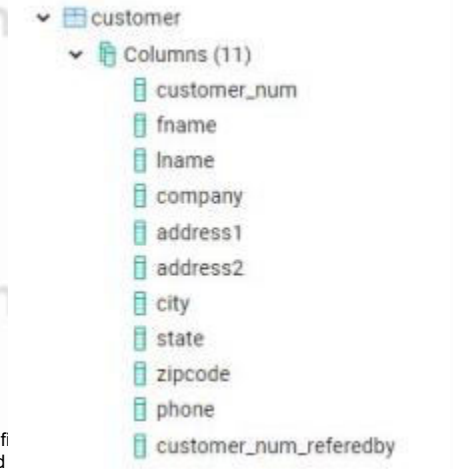
Si la tabla hubiese tenido dos campos varchar, no nos habríamos dado cuenta del error, ya que el Motor de BD habría aceptado los datos erróneos.

SQL - Operador INSERT

```
INSERT INTO customer
VALUES (266,'Godoy','Estela','Pintos
3325','Ituza SA',null,'Buenos Aires'
,null,null, null,null,null)
```

El problema de no poner la lista de columnas a insertar es que mi lista de valores debe respetar el orden de las columnas en la tabla, acoplando mi consulta al modelo.





En este caso vemos que el motor de BD acepto los datos erróneos, porque son del mismo tipo.

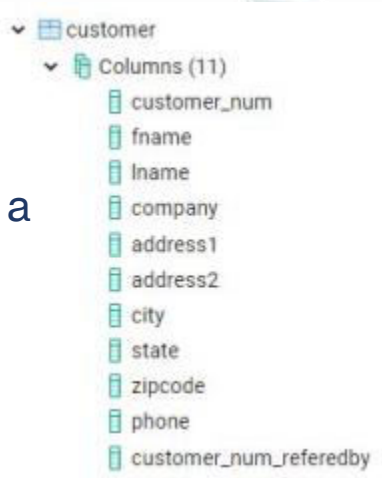
El apellido y nombre están invertidos y la direccion1 y la compañía también.

CePETel

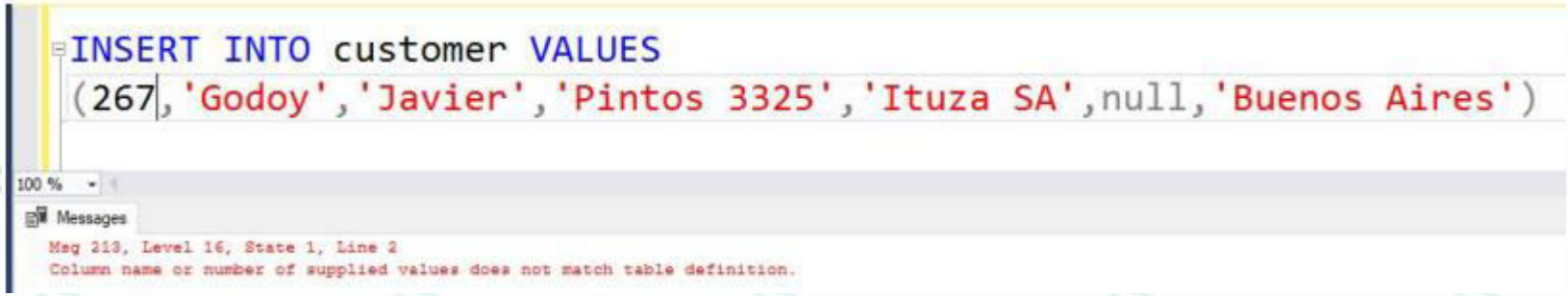
SQL - Operador INSERT

```
INSERT INTO customer
VALUES (266,'Godoy','Estela','Pintos 3325','Ituza SA',null,'Buenos Aires' )
```

En este caso vemos que el motor de BD ingreso los datos en la Base de forma errónea y los aceptó debido a que eran del mismo tipo.



En otros motores de Base de Datos esta operación da error (Ej. en MS SqlServer).



CePETel

SQL - Operador INSERT

```
INSERT INTO product_types (stock_num, description)
VALUES (376,'Short Rugby')
```

Inserta en la tabla tipo de productos un nuevo tipo de producto.

Agregando la lista de atributos podemos desacoplar la instrucción de la definición de la tabla.

Query Editor | Query History

```
1  INSERT INTO product_types
2  (stock_num, description)
3  VALUES (376,'Short Rugby')
```

Data Output | Explain | Messages | Notifications

```
INSERT 0 1

Query returned successfully in 569 msec
```

Query Editor | Query History

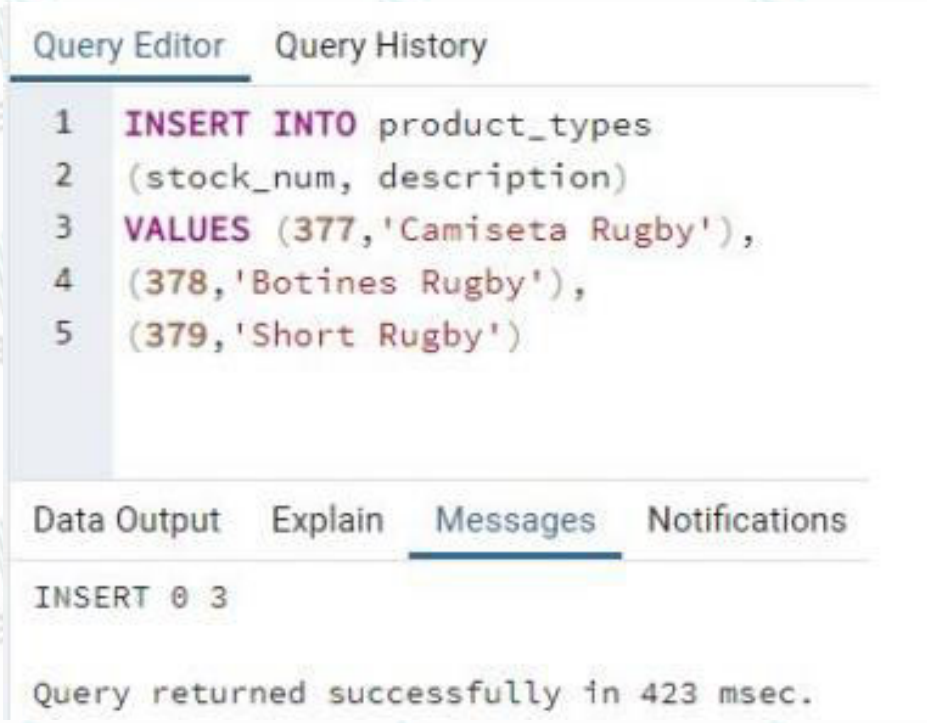
```
1  SELECT * FROM product_types WHERE stock_num=376
```

Data Output | Explain | Messages | Notifications

	stock_num [PK] smallint	description character varying (15)
1	376	Short Rugby

SQL - Operador INSERT

```
INSERT INTO product_types (stock_num, description)
VALUES (377,'Camiseta Rugby'), (378,'Botines Rugby'),
(379,'Short Rugby')
```



Inserta varias filas en la tabla tipo de productos de nuevos tipos de productos.

Agregando la lista de atributos podemos desacoplar la instrucción de la definición de la tabla.

SQL - Operador INSERT

```
INSERT INTO customer (stock_num, lname, fname, address1, Company,
address2, city)
VALUES (266,'Godoy','Estela','Pintos 3325', 'Ituza SA',null,'Buenos Aires')
```

Al agregar la lista de columnas, podemos insertar filas con una cantidad menor de valores a los que tiene la definición de la tabla y sin necesitar respetar el orden de los mismos en la tabla real.

Siempre y cuando la lista de columnas y la lista de valores coincidan

Query Editor | Query History

```
1  INSERT INTO customer
2  (customer_num, lname, fname, address1,Company, address2, city)
3  VALUES
4  (270,'Godoy','Estela','Pintos 3325','Ituza SA',null,'Buenos Aires');
5
6  SELECT customer_num, lname, fname, address1,Company, address2, city
7  FROM customer WHERE customer_num=270;
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	lname character varying (15)	fname character varying (15)	address1 character varying (20)	company character varying (20)	address2 character varying (20)	city character varying (15)
1	270	Godoy	Estela	Pintos 3325	Ituza SA	[null]	Buenos Aires

Se observa que se pudo insertar la fila en la tabla. Obviamente que las columnas no incluidas en el INSERT deben aceptar valores nulos o tener un DEFAULT definido.

SQL - Operador INSERT

INSERT INTO closed_orders

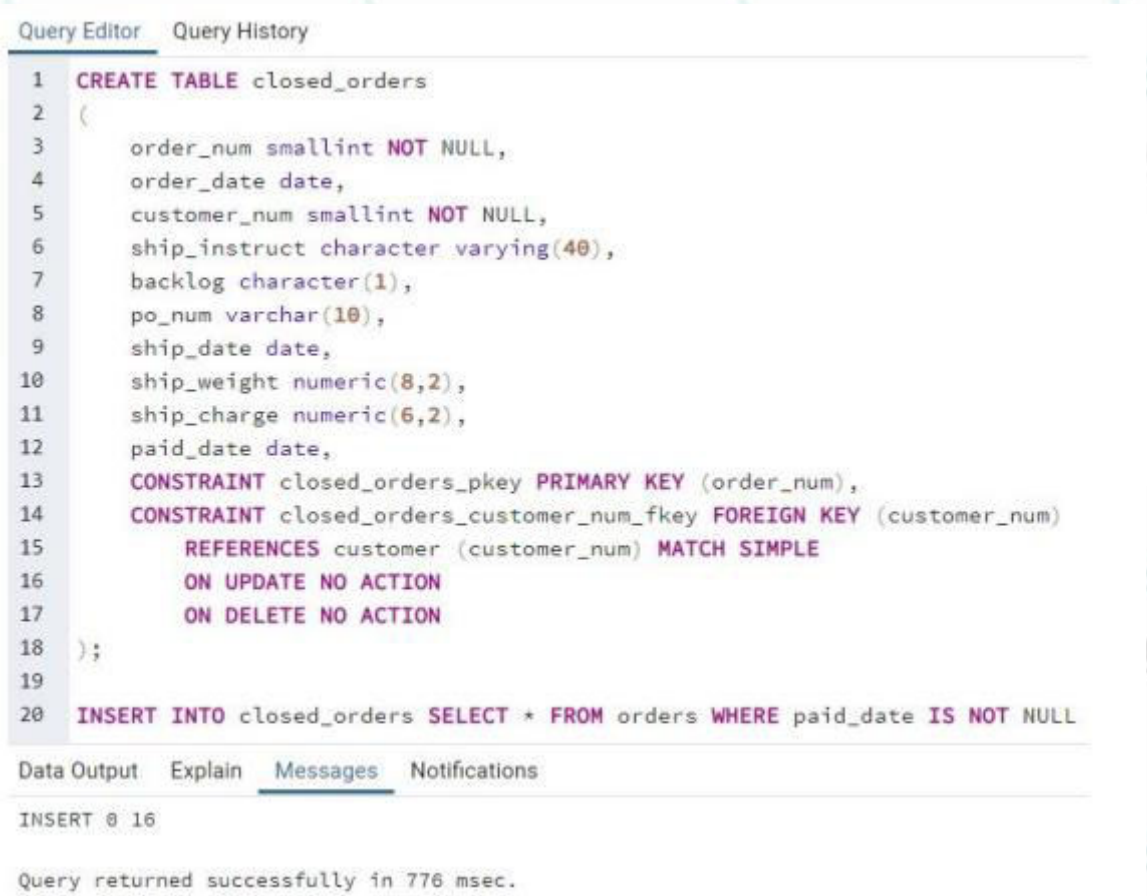
SELECT * FROM orders

WHERE paid_date IS NOT null

En una tabla creada previamente con la misma estructura que la tabla ordenes, insertamos las filas que devuelve el SELECT.

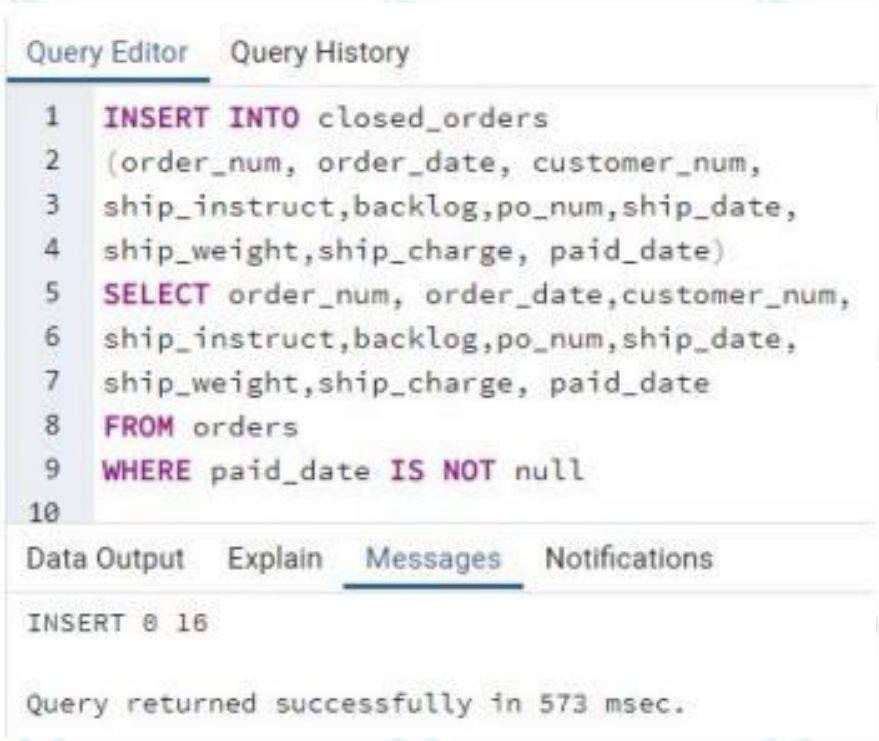
Es fundamental que el select devuelva las mismas filas y en el mismo orden las columnas que la tabla destino.

En este ejemplo el insert está acoplado a la definición de la tabla closed_orders y el SELECT está acoplado a la definición de la tabla orders.



SQL - Operador INSERT

```
INSERT INTO closed_orders (order_num, order_date, customer_num,
ship_instruct,backlog,po_num,ship_date, ship_weight,ship_charge, paid_date)
SELECT order_num, order_date,customer_num, ship_instruct,backlog,po_num,ship_date,
ship_weight,ship_charge, paid_date
FROM orders
WHERE paid_date IS NOT null
```



En una tabla creada previamente con la misma estructura que la tabla ordenes, insertamos las filas que devuelve el SELECT.

Con la lista de columnas en el INSERT y en el SELECT los desacoplamos de la definición de las tablas y viendo las instrucciones sabemos claramente que se está insertando y en dónde.

SQL - Operador INSERT

```
INSERT INTO manufact
(manu_code, manu_name, lead_time,state)
VALUES ('DBL','DBLANDIT',1,'CA')
```

Query Editor | Query History

```
1  ALTER TABLE manufact ALTER COLUMN f_alta_audit SET DEFAULT CURRENT_DATE;
2  ALTER TABLE manufact ALTER COLUMN d_usualta_audit SET DEFAULT USER;
3
4  INSERT INTO manufact (manu_code, manu_name, lead_time,state)
5  VALUES ('DBL','DBLANDIT',1,'CA');
6
7  SELECT * FROM manufact WHERE manu_code='DBL';
```

Data Output | Explain | Messages | Notifications

	manu_code [PK] character (3)	manu_name character varying (15)	lead_time smallint	state character (2)	f_alta_audit date	d_usualta_audit character varying (20)
1	DBL	DBLANDIT	1	CA	2020-10-17	postgres

DEFAULT CONSTRAINT

Cuando insertamos valores en una tabla y obviamos determinadas columnas, por DEFAULT tendrán valor NULL, salvo que con el constraint de DEFAULT les asignemos otro valor.

Alteramos la tabla manufact agregando dos valores DEFAULT. Se observa que en los atributos f_alta_audit y d_usualta_audit tienen los datos definidos como DEFAULT en la definición de la tabla.

SQL - Operador INSERT

INSERT INTO empleados (nombre, apellido, cuit)

VALUES ('Lisandro','Ayestaran',20235582487)

Query Editor | Query History

```
1  CREATE TABLE empleados
2  (empleadoid SERIAL,
3   nombre VARCHAR(60),
4   apellido VARCHAR(60),
5   cuit BIGINT
6  );
7
8  INSERT INTO empleados (nombre, apellido, cuit)
9  VALUES ('Lisandro','Ayestaran',20235582487);
10
11 SELECT * FROM empleados;
```

Data Output | Explain | Messages | Notifications

	empleadoid integer	nombre character varying (60)	apellido character varying (60)	cuit bigint
1	1	Lisandro	Ayestaran	20235582487

SERIAL / SECUENCIA

Al insertar una fila en una tabla con un atributo con un tipo de dato SERIAL, NO se debe incluir dicha columna en la lista de columnas, ni poner valor en la lista de valores.

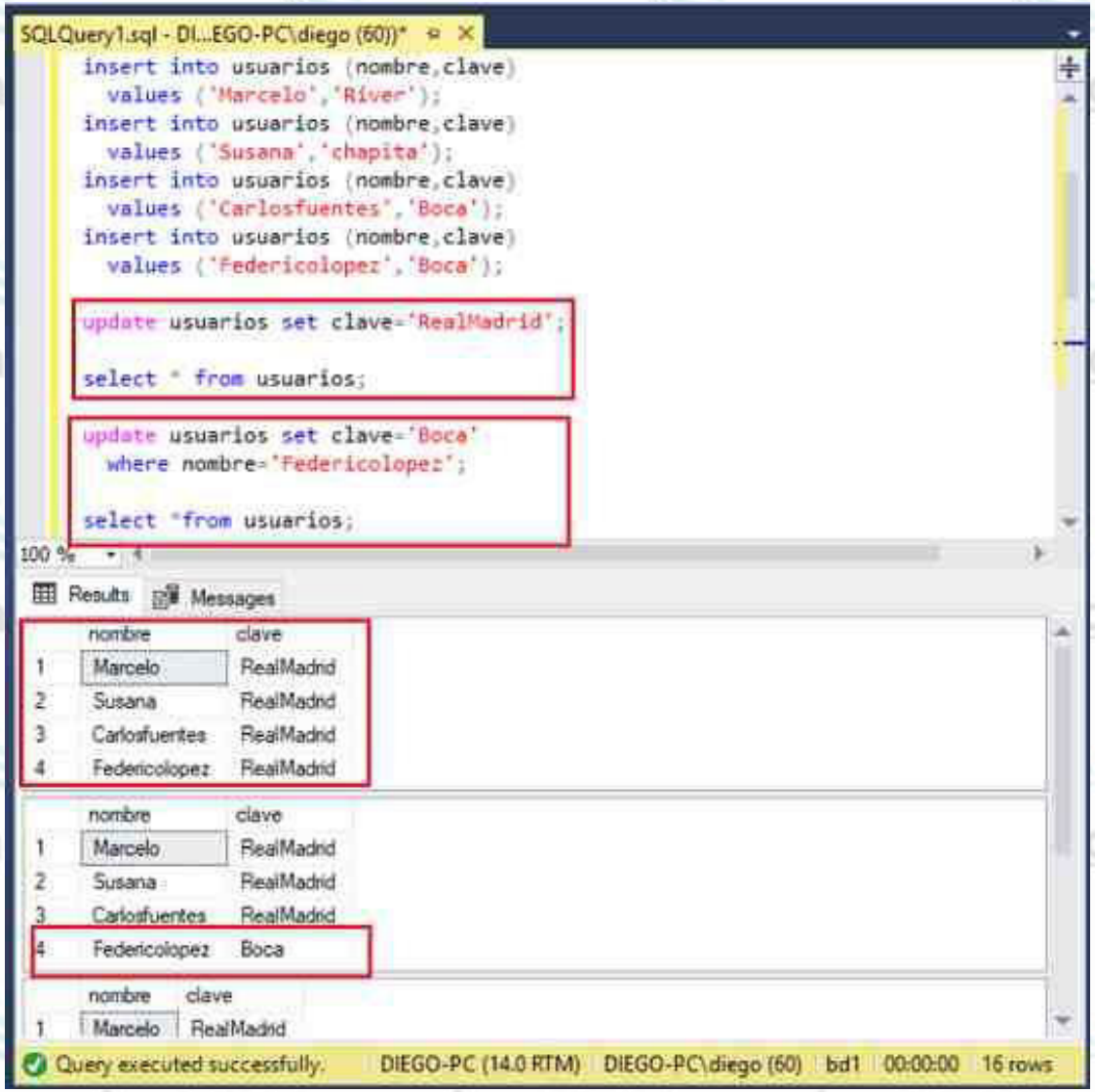
En el ejemplo vemos que la columna empleadoid no es incluida en la lista de columnas y en la lista de valores.

UPDATE

SQL - Operador UPDATE

Se modifica una o varias columnas de las filas que cumplan con la condición.

UPDATE nom_tabla
SET columna=valor[, columna=valor...])
[WHERE condiciones] (*)



(*) La cláusula WHERE es opcional, pero CUIDADO si no se pone esa condición el UPDATE se realizará sobre la TOTALIDAD DE LAS FILAS DE LA TABLA.

SQL - Operador UPDATE

```
UPDATE customer
SET company = 'ITBA', phone = '5555-5555'
WHERE customer_num = 112
```

Query Editor | Query History

```
1  UPDATE customer
2  SET company = 'ITBA',  phone = '5555-5555'
3  WHERE customer_num = 112;
4
5  SELECT customer_num,fname,lname,company,city,phone
6  FROM customer WHERE  customer_num = 112;
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	company character varying (20)	company character varying (20)	city character varying (15)	phone character varying (18)
1	112	Margaret	Lawson	ITBA	ITBA	Los Altos	5555-5555

SQL - Operador UPDATE

```
UPDATE empleados
SET apellido='García'
```

```
INSERT INTO empleados (nombre, apellido, cuit)
VALUES
('Carlos','Noseda',6),
('Marino','Cippitelli',5),
('Augusto','Vigil',4),
('Carlos','Vargaz',3),
('Jorge','Imach',2),
('Miguel','Ariagno',1);

SELECT * FROM empleados;
```

Data Output | Explain | Messages | Notifications

	empleadoid integer	nombre character varying (60)	apellido character varying (60)	cuit bigint
1	1	Lisandro	Ayestaran	20235582487
2	2	Carlos	Noseda	6
3	3	Marino	Cippitelli	5
4	4	Augusto	Vigil	4
5	5	Carlos	Vargaz	3
6	6	Jorge	Imach	2
7	7	Miguel	Ariagno	1

WARNING

Este UPDATE no tiene la cláusula WHERE, por lo que se modificarán todas las filas de la tabla asignándole al apellido 'Garcia'.

Query Editor | Query History

```
UPDATE empleados
SET apellido='García';

SELECT * FROM empleados;
```

Data Output | Explain | Messages | Notifications

	empleadoid integer	nombre character varying (60)	apellido character varying (60)	cuit bigint
1	1	Lisandro	García	20235582487
2	2	Carlos	García	6
3	3	Marino	García	5
4	4	Augusto	García	4
5	5	Carlos	García	3
6	6	Jorge	García	2
7	7	Miguel	García	1

SQL - Operador UPDATE

```
UPDATE products
SET unit_price= unit_price*1,20
WHERE manu_code='ANZ'
```

```
UPDATE products
SET unit_price= unit_price*1.20
WHERE  manu_code='ANZ'
```

00 %

Messages

```
(11 row(s) affected)
```

Query Editor | Query History

```
SELECT * FROM products WHERE  manu_code='ANZ'
```

Data Output | Explain | Messages | Notifications

	stock_num [PK] smallint	manu_code [PK] character (3)	unit_price numeric (6,2)	unit_code smallint
1	5	ANZ	19.80	19
2	6	ANZ	48.00	12
3	8	ANZ	840.00	13
4	9	ANZ	20.00	19
5	110	ANZ	244.00	16
6	201	ANZ	75.00	19
7	205	ANZ	312.00	13
8	301	ANZ	95.00	19
9	304	ANZ	170.00	1
10	310	ANZ	84.00	9
11	313	ANZ	60.00	2

Modificar ciertas filas cambiando el valor de una columna por el resultado de una fórmula.

Query Editor | Query History

```
UPDATE products
SET unit_price= unit_price*1.20
WHERE  manu_code='ANZ';

SELECT * FROM products WHERE  manu_code='ANZ'
```

Data Output | Explain | Messages | Notifications

	stock_num [PK] smallint	manu_code [PK] character (3)	unit_price numeric (6,2)	unit_code smallint
1	5	ANZ	23.76	19
2	6	ANZ	57.60	12
3	8	ANZ	1008.00	13
4	9	ANZ	24.00	19
5	110	ANZ	292.80	16
6	201	ANZ	90.00	19
7	205	ANZ	374.40	13
8	301	ANZ	114.00	19
9	304	ANZ	204.00	1
10	310	ANZ	100.80	9
11	313	ANZ	72.00	2

***Este
***This

CePETel

SQL DELETE:

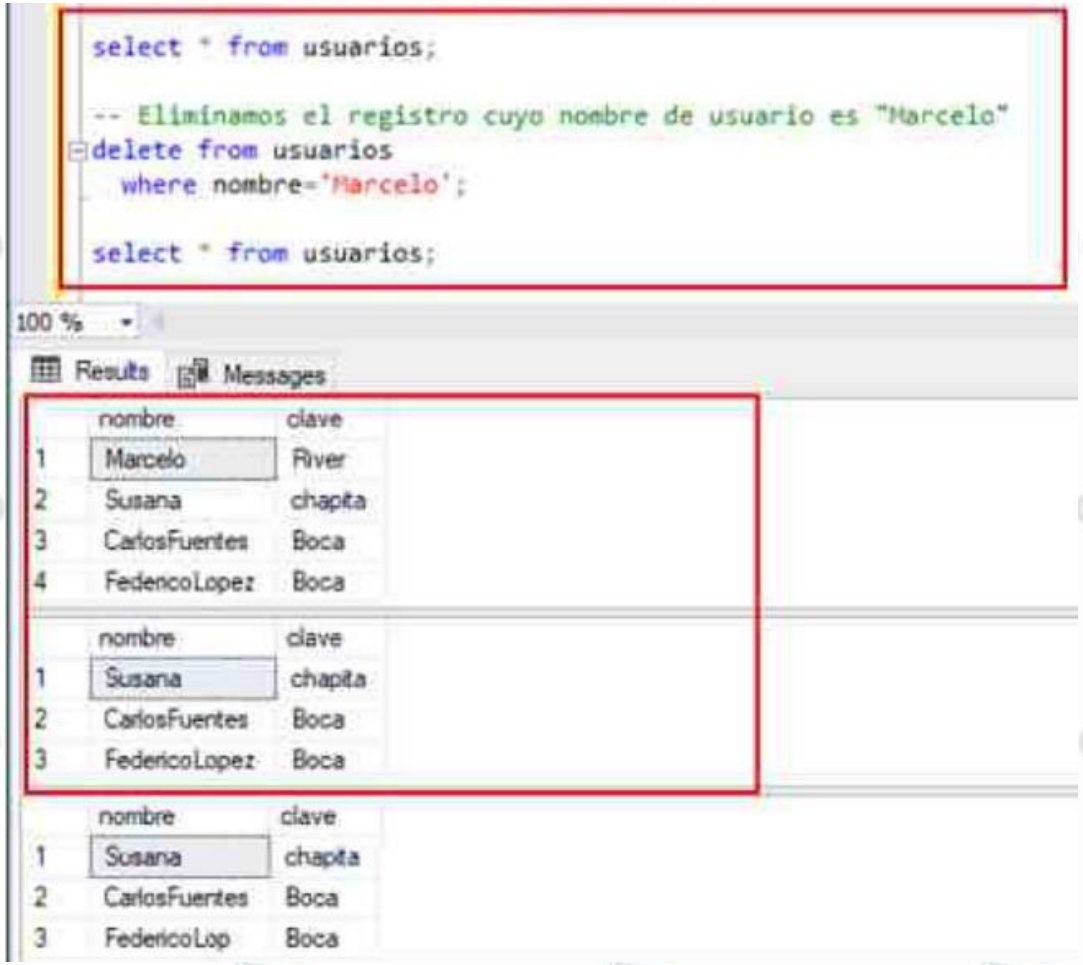
- The DELETE statement is used to delete existing records in a table.
- DELETE FROM *table_name*
  WHERE *condition*;
- DELETE FROM Customers
  WHERE CustomerName='junaid';
- It is possible to delete all rows in a table without deleting the table. This means that the table structure, attributes, and indexes will be intact:
- DELETE * FROM *table_name*;

SQL - Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE.

```
DELETE FROM nom_tabla
[WHERE condiciones] (*)
```

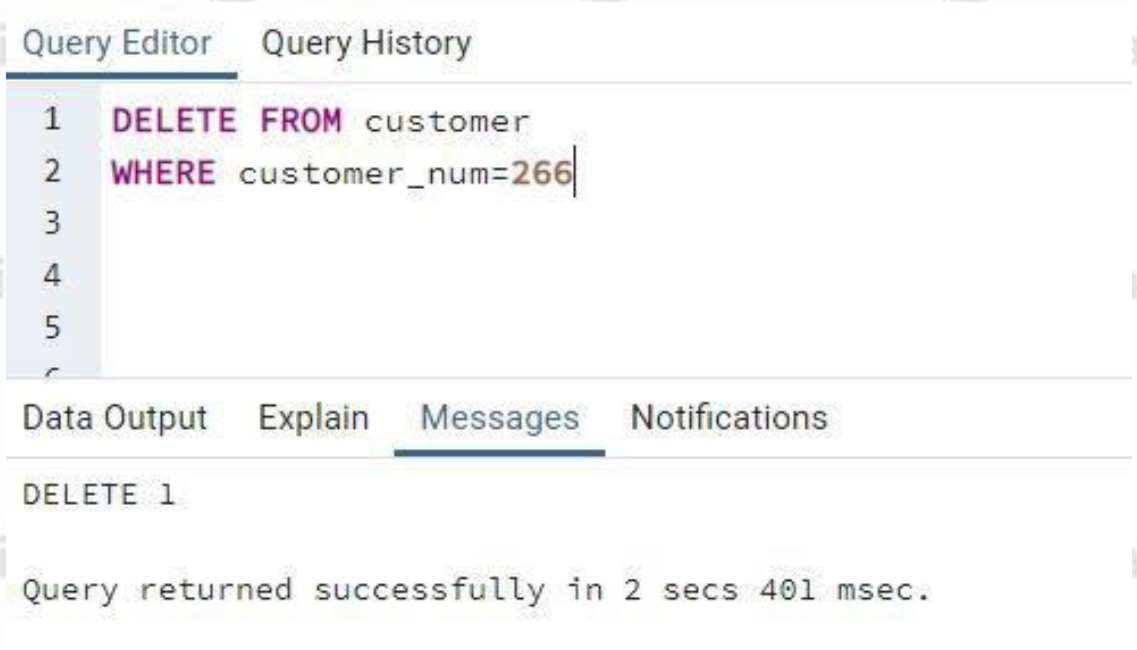
(*) La cláusula WHERE es opcional, pero CUIDADO si no se pone esa condición el DELETE borrara la TOTALIDAD DE LAS FILAS DE LA TABLA.



SQL - Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE, o sea cuyo customer_num sea 266

```
DELETE FROM customer
WHERE customer_num=266
```

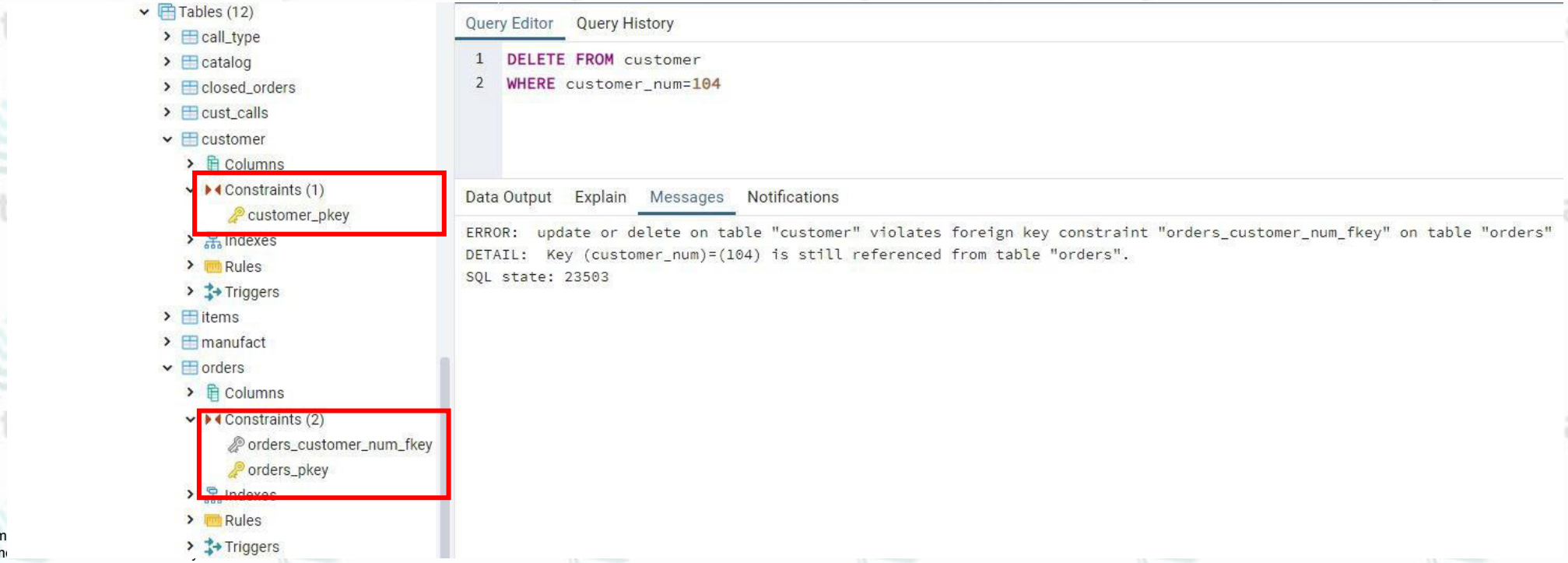


SQL - Operador DELETE

Se eliminan las filas que cumplan con la condición del DELETE, o sea cuyo customer_num sea 104

```
DELETE FROM customer
WHERE customer_num=104
```

WARNING
Cuidado con la Integridad Referencial resguardada por las PK y las FK.



***Este docum
***This docum

SQL – DELETE vs TRUNCATE

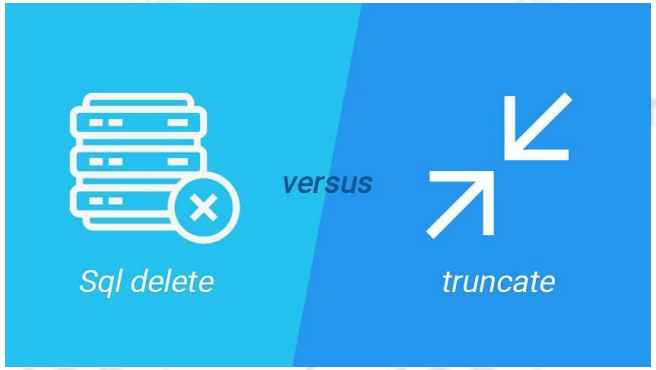
TRUNCATE

A diferencia de DELETE, TRUNCATE elimina todas las filas de la tabla sin borrar la tabla:

TRUNCATE TABLE nombre_tabla;

Truncate	Delete
• It removes all rows from a table. It is faster & does not use as much undo space as a Delete • It is a DDL command so this command change structure of table • You cannot rollback in Truncate • In SQL, the auto increment counter gets reset with truncate	• It is used to remove rows from table. A WHERE clause can be used to only remove some vows • It is a DML command. It only remove rows from a table, leaving the table structure untouched • In DELETE, you can rollback • The auto increment counter cannot get reset with delete

SQL – DELETE vs TRUNCATE



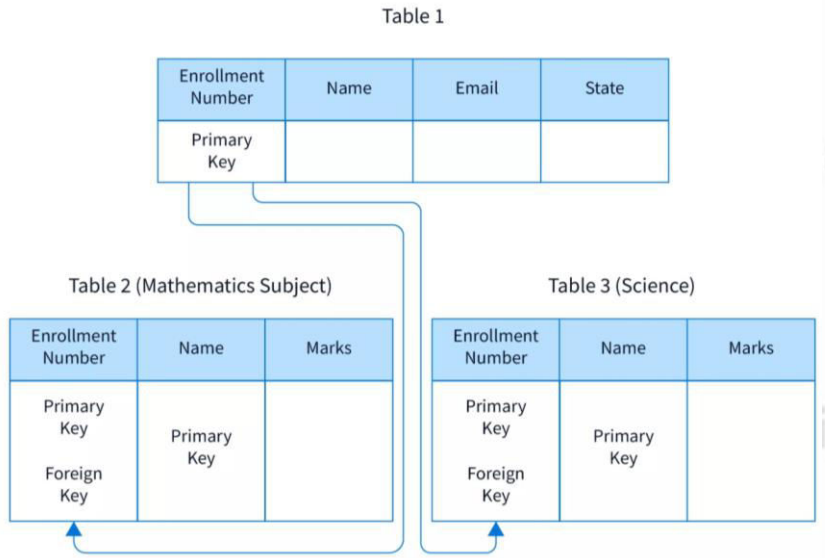
TRUNCATE VS DELETE

- Ambas eliminan los datos, no la estructura.
- Solo DELETE permite la eliminación condicional de los registros.
- DELETE es una operación registrada en el log de transacciones, basada en registrar cada eliminación individual.
- TRUNCATE es una operación registrada en el log de transacciones, pero como un todo, en conjunto, no por eliminación individual. TRUNCATE se registra como una liberación de las páginas de datos en las cuales existen los datos.
- TRUNCATE es más rápida que DELETE.
- Ambas se pueden deshacer con un ROLLBACK.
- TRUNCATE reiniciará el contador para una tabla que contenga una columna IDENTITY.
- DELETE mantendrá el contador de la tabla para una columna IDENTITY.
- TRUNCATE es un comando DDL(lenguaje de definición de datos) mientras que DELETE es un DML(lenguaje de manipulación de datos).
- TRUNCATE no desencadena un TRIGGER, DELETE sí.

SQL – CASCADE

ELIMINAR EN CASCADA: Cuando creamos una clave foránea utilizando esta opción, elimina las filas de referencia en la tabla secundaria cuando la fila referenciada se elimina en la tabla primaria que tiene una clave primaria.

ACTUALIZAR CASCADA: cuando creamos una clave externa utilizando ACTUALIZAR CASCADA, las filas de referencia se van a actualizar en la tabla secundaria cuando la fila referenciada se actualiza en la tabla principal que tiene una clave primaria.



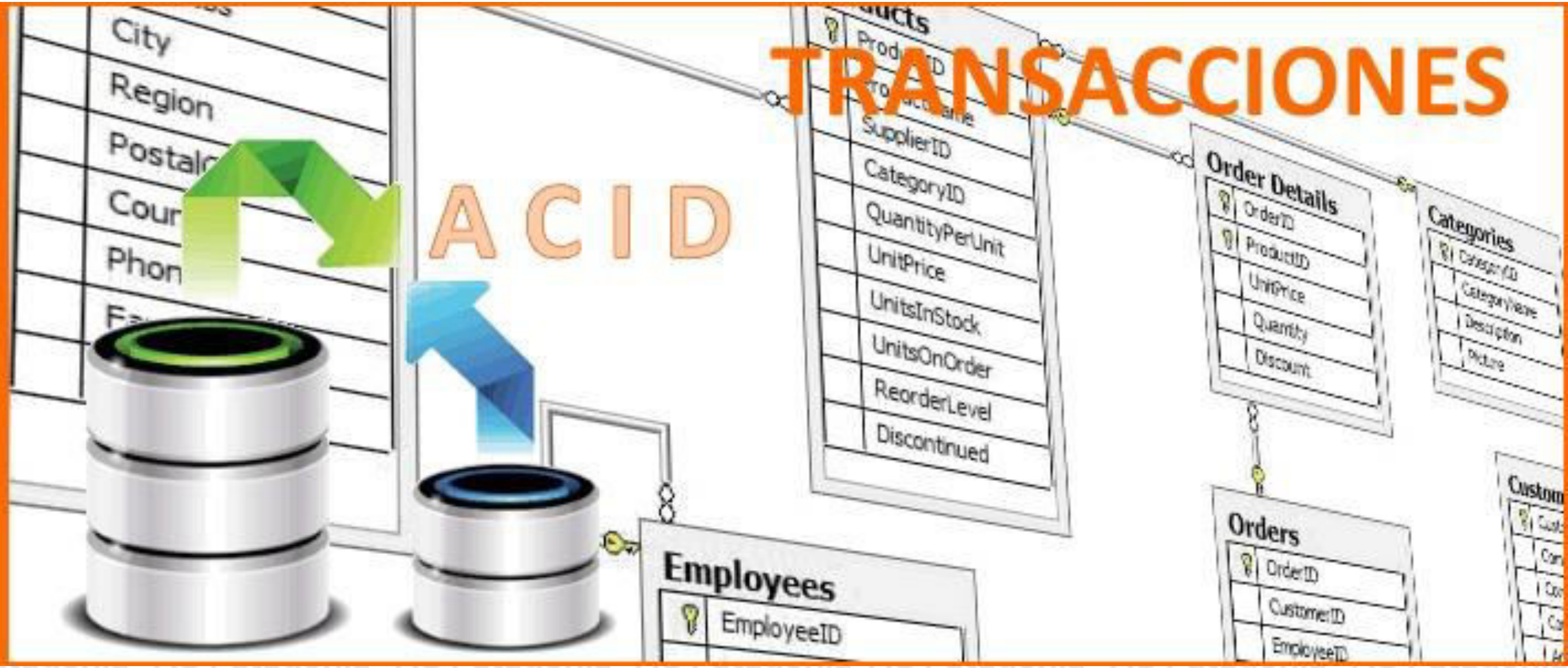
SQLQuery10.sql - L...RRE\mgledhill (52))* | SQLQuery9.sql - us...RRE\mgledhill (60))

```
USE [Northwind]
GO

ALTER TABLE [dbo].[Orders] DROP CONSTRAINT [FK_Orders_Customers]
GO

ALTER TABLE [dbo].[Orders]  WITH NOCHECK ADD  CONSTRAINT [FK_Orders_Customers] FOREIGN KEY([CustomerID])
REFERENCES [dbo].[Customers] ([CustomerID]) ON DELETE CASCADE
GO

ALTER TABLE [dbo].[Orders] CHECK CONSTRAINT [FK_Orders_Customers]
GO
```

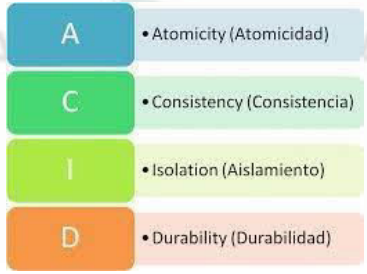
CePETel

Fundamentos de SQL

TRANSACCIONES



SQL - Transacciones

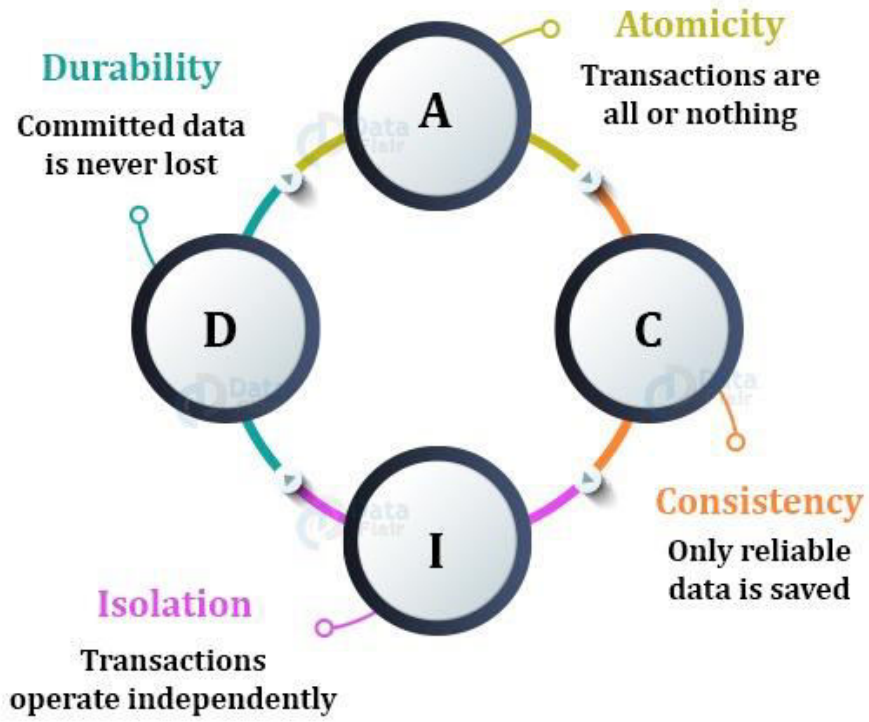


Propiedades ACID

Una transacción, para cumplir con su propósito y protegernos de todos los problemas que hemos visto, debe presentar las siguientes características:

- **Atomicidad**: las operaciones que componen una transacción deben considerarse como una sola.
- **Consistencia**: una operación nunca deberá dejar datos inconsistentes.
- **Aislamiento**: los datos "sucios" deben estar aislados, y evitar que los usuarios utilicen información que aún no está confirmada o validada. (por ejemplo: ¿sigue siendo válido el saldo mientras realizo la operación?)
- **Durabilidad**: una vez completada la transacción los datos actualizados ya serán permanentes y confirmados.

A estas propiedades se las suele conocer como propiedades ACID (de sus siglas en inglés: Atomicity, Consistency, Isolation y Durability).



SQL - Transacciones

Mecanismos para garantizar la consistencia de datos

El DBMS cuenta con distintos mecanismos para poder asegurar la consistencia de los datos que existentes en la/s Base/s de Datos.

Conceptos relacionados con la consistencia de datos

- ***TRANSACCIONES***
  - Es un conjunto de sentencias SQL que se ejecutan atómicamente en una unidad lógica de trabajo. Partiendo de que una transacción lleva la base de datos de un estado correcto a otro estado correcto, el motor posee mecanismos de manera de garantizar que la operación completa se ejecute o falle, no permitiendo que queden datos inconsistentes.
  - Cada sentencia de alteración de datos (insert, update o delete) es una transacción en sí misma (singleton transaction).

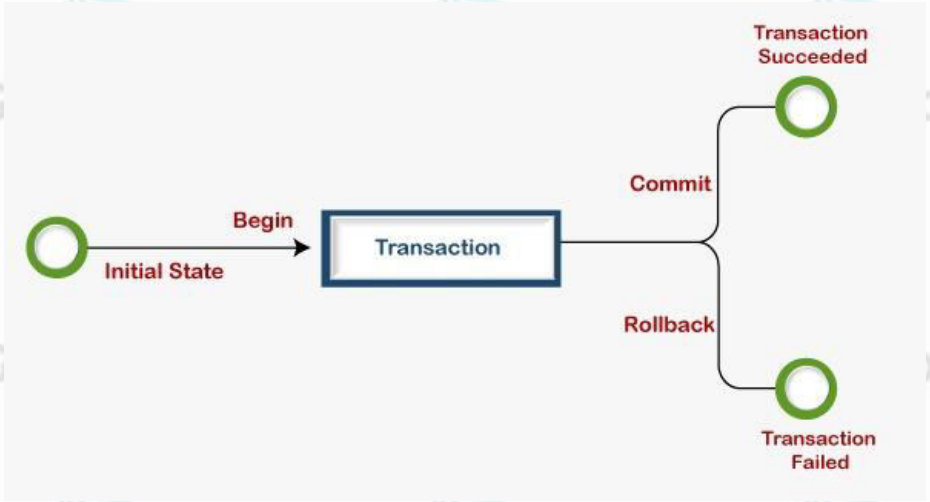
- ***LOGS TRANSACCIONALES***: Es un registro donde el motor almacena la información de cada operación llevada a cabo con los datos

- ***RECOVERY***: Método de recuperación ante caídas.

SQL - Transacciones

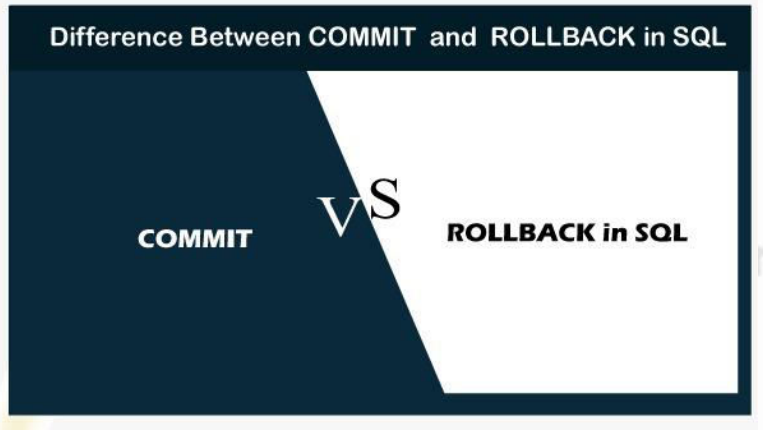
Transacciones

Para definir una transacción debemos definir un conjunto de instrucciones precedidas por la sentencia **BEGIN WORK**, de esta manera las sentencias a continuación se ejecutarán de forma atómica. Pero para lo que es el concepto de transacción, una transacción puede finalizar correctamente o puede fallar; en el caso de finalizar correctamente, todos los datos actualizados se confirmarían en la base de datos y en caso de fallar o por una regla de negocio definido, se deberían deshacer todos los cambios hasta el comienzo de la transacción.



Para manejar estas acciones contamos con la sentencia **COMMIT WORK** para actualizar los datos en la base de datos, y con la sentencia **ROLLBACK WORK** para deshacer nuestra transacción.

SQL - Transacciones



Transacciones (Cont.)

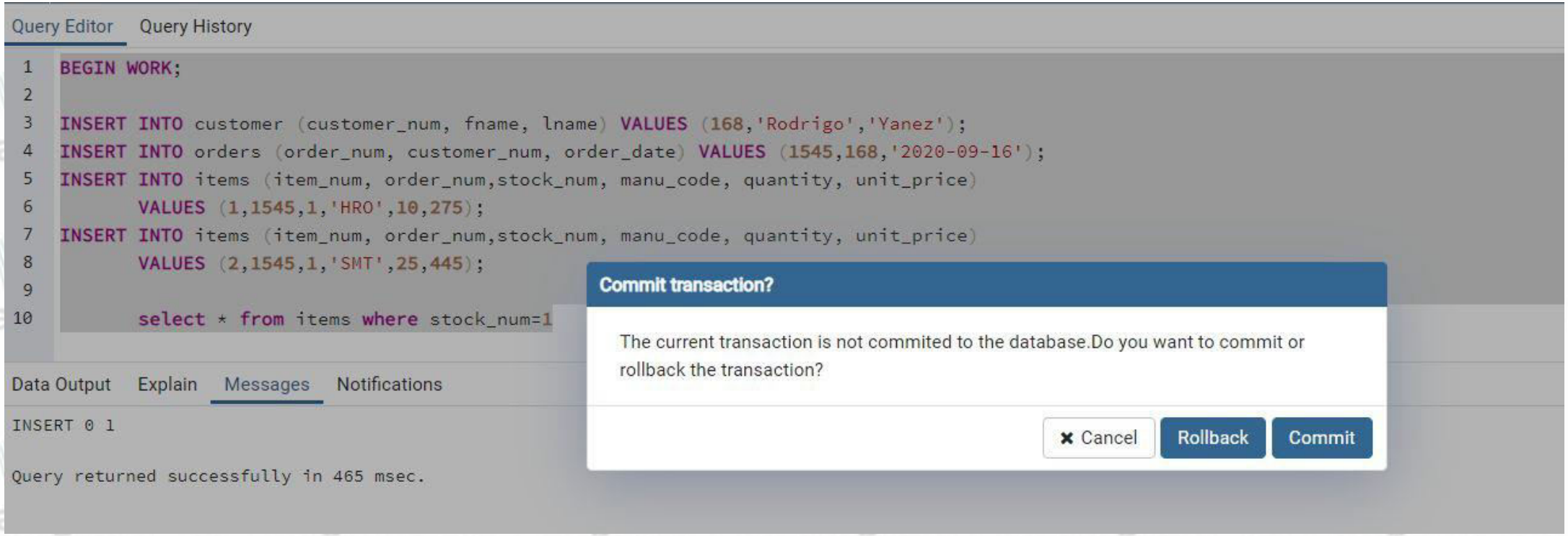
```
BEGIN WORK;

INSERT INTO customer (customer_num, fname, lname)
VALUES (168,'Rodrigo','Yanez');
INSERT INTO orders (order_num, customer_num, order_date)
VALUES (1545,168,'2020-09-16');
INSERT INTO items (item_num, order_num,stock_num, manu_code, quantity,
unit_price)
VALUES (1,1545,1,'HRO',10,275);
INSERT INTO items (item_num, order_num,stock_num, manu_code,
quantity,unit_price)
VALUES (2,1545,1,'SMT',25,445);
```

En el ejemplo observamos que se inició una transacción y se realizaron varios inserts que representan una única unidad, o sea si algo fallase se desharía toda la transacción.
Ahora bien, observamos que la transacción no está finalizada, ni por OK Commit, ni por Error Rollback.

SQL - Transacciones

Transacciones (Cont.)

Caso 1 - Queremos abandonar la sesión de PGAdmin cerrando la ventana del Query Tool.



SQL - Transacciones

Transacciones (Cont.)

Caso 2 - Realizamos un Commit a la Transacción

```
13  SELECT * FROM customer WHERE customer_num=168;
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	company character varying (20)
1	168	Rodrigo	Yanez	[null]

```
14  SELECT * FROM orders WHERE order_num=1545;
```

Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	ship_instruct character varying (40)
1	1545	2020-09-16	168	[null]

```
15  SELECT * FROM items WHERE order_num=1545;
```

Data Output | Explain | Messages | Notifications

	item_num smallint	order_num smallint	stock_num smallint	manu_code character (3)	quantity smallint	unit_price numeric (8,2)
1	1	1545	1	HRO	10	275.00
2	2	1545	1	SMT	25	445.00

Query Editor | Query History

```
1  BEGIN WORK;
2
3  INSERT INTO customer (customer_num, fname, lname) VALUES (168,'Rodrigo','Yanez');
4  INSERT INTO orders (order_num, customer_num, order_date) VALUES (1545,168,'2020-09-16');
5  INSERT INTO items (item_num, order_num,stock_num, manu_code, quantity, unit_price)
6         VALUES (1,1545,1,'HRO',10,275);
7  INSERT INTO items (item_num, order_num,stock_num, manu_code, quantity, unit_price)
8         VALUES (2,1545,1,'SMT',25,445);
9
10 COMMIT WORK;
```

Data Output | Explain | Messages | Notifications

```
COMMIT

Query returned successfully in 615 msec.
```

SQL - Transacciones

Transacciones (Cont.)

Caso 3 - Realizamos un Rollback a la Transacción

Consultas Durante la Transacción

```
13  SELECT * FROM customer WHERE customer_num=168;
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	company character varying (20)
1	168	Rodrigo	Yanez	[null]

```
14  SELECT * FROM orders WHERE order_num=1545;
```

Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	ship_instruct character varying (40)
1	1545	2020-09-16	168	[null]

```
15  SELECT * FROM items WHERE order_num=1545;
```

Data Output | Explain | Messages | Notifications

	item_num smallint	order_num smallint	stock_num smallint	manu_code character (3)	quantity smallint	unit_price numeric (8,2)
1	1	1545	1	HRO	10	275.00
2	2	1545	1	SMT	25	445.00

```
8           VALUES (2,1545,1,'SMT',25,445);
9
10  ROLLBACK WORK;
11
```

Data Output | Explain | Messages | Notifications

```
ROLLBACK

Query returned successfully in 501 msec.
```

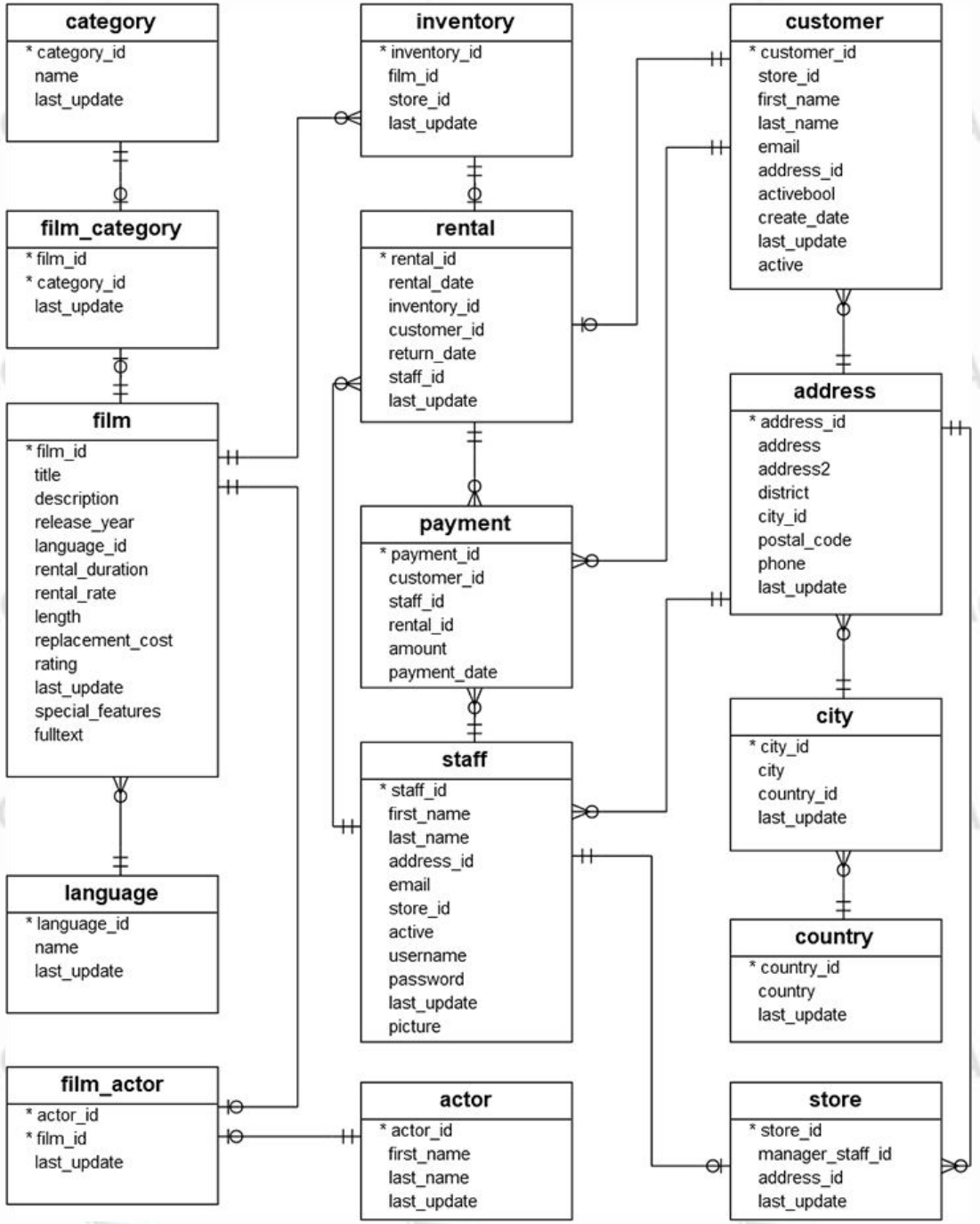
Consultas luego del Rollback

customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	company character varying (20)

order_num [PK] smallint	order_date date	customer_num smallint	ship_instruct character varying (40)	backl chara

item_num smallint	order_num smallint	stock_num smallint	manu_code character (3)	quantity smallint	unit_price numeric (8,2)

TP N° 5 : Update – Insert - Delete



CePETel

TP N° 5 : Update – Insert - Delete

1. Enumere todos los países.
2. Muestra el número de países.
3. Busque Estados Unidos en la tabla de países.
4. Enumere todos los pagos con un monto de 1,99, 2,99, 3,99 o 4,99
5. Inserte un nuevo registro denominado utopía en la tabla de países.
6. ¿Se puede ejecutar esta consulta con éxito? Insertar en los valores de país (country_id, país) (1, 'Utopía');
7. Vea que lenguajes hay en la tabla Language, inserte uno nuevo con todos los datos requeridos por la tabla, no puede haber campos nulos
8. Modifique el campo “last_update” a la fecha actual
9. Elimine el registro creado en el punto 7
10. Muestre todos los actores que se figuren en la tabla “actor”, creando una columna nueva concatenando el apellido (last_name) y el nombre (first_name), tener en consideración usar separador ‘,’ y un espacio, y asígnele un alias.
11. Inserte un actor nuevo con todos los datos necesarios según la definición de la tabla
12. Elimine el actor creado en el punto 11

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

Módulo 6:

DML (Secuencias – Vistas – Tablas temporales)

Data Manipulation Language

Data Manipulation Language

CePETel

DML - Secuencias

Los generadores de secuencias proveen una serie de números secuenciales, especialmente usados en entornos multiusuarios para generar una números secuenciales y únicos sin el overhead de I/O a disco ó lockeo transaccional.

Los motores de base de datos proveen diferentes formas de implementar secuencias a través de:

- Tipo de dato de una columna (Informix, PostgreSQL)
- Propiedades de una columna (SqlServer, Mysql, DB2)
- Objeto Sequence (Oracle, Informix, PostgreSQL, DB2, SqlServer)

DML - Secuencias

- Tipo de Dato de una columna
  - Motor BD Informix/PostgreSQL SERIAL, BIGSERIAL.
- Propiedades de una columna
  - Motor SqlServer, DB2 IDENTITY
  - Motor Mysql AUTO_INCREMENT
- Objeto Sequence
  - Motores Oracle, Informix, PostgreSQL, DB2, SqlServer.
  - CREATE SEQUENCE

DML - Secuencias

Tipo de Dato de una columna

El motor PostgreSQL posee varios tipos de datos SERIAL, BIGSERIAL que permiten realizar lo mismo que un objeto secuencia. Al insertar una fila en dicha tabla y asignarle un valor cero, el motor va a buscar el próximo nro. del más alto existente en la tabla.

```
CREATE TABLE tickets (
        ticketid BIGSERIAL,
        customer_num INTEGER REFERENCES customer,
        order_num INTEGER REFERENCES orders,
        ticket_date DATE,
        description VARCHAR(2000),
        d_alta_audit DATE DEFAULT CURRENT_DATE,
        d_usuario VARCHAR(20) DEFAULT USER);
INSERT INTO tickets (customer_num, order_num, ticket_date, description)
        VALUES (168,1545,'16/10/2020',
        'El cliente reclama la entrega de productos')
```

CePETel

DML - Secuencias

Propiedades de una columna

Existen motores que poseen propiedades de columna que permite realizar lo mismo que una secuencia. Al insertar una fila en dicha tabla, el motor va a buscar el próximo nro. del más alto existente en la tabla.

Ej. SQLServer IDENTITY

```
CREATE TABLE ordenes (
        N_orden int IDENTITY (1, 1),
        N_cliente int NULL ,
        F_orden datetime NULL);

INSERT INTO ordenes
(n_cliente, f_orden)

VALUES (114,'2020-03-03')
```

Ej. MySQL AUTO_INCREMENT

```
CREATE TABLE animales
        (animal_id INT AUTO_INCREMENT,
        nombre CHAR(30) NOT NULL);

INSERT INTO animales
('perro'),
('gato');
```

CePETel

DML - Secuencias

Objeto SEQUENCE

Una secuencia debe tener un nombre, debe ser ascendente o descendente, debe tener definido el intervalo entre números, tiene definidos métodos para obtener el próximo número ó el actual (entre otros).

Ej. SEQUENCE PostGreSQL

```
CREATE SEQUENCE sq_numerador
INCREMENT BY 1
NO MINVALUE
NO MAXVALUE
START WITH 3000
```

```
INSERT INTO orders (order_num, customer_num, order_date)
VALUES (NEXTVAL('sq_numerador'), 168, '2020-10-15');

INSERT INTO items (order_num, item_num, stock_num,
manu_code, quantity, unit_price)
VALUES (CURRVAL('sq_numerador'),1,1,'SMT',30, 250);
```

Ejemplo de actualización de secuencia con valor específico

```
SELECT SETVAL('sq_numerador', 4000) FROM orders LIMIT 1
```

Métodos asociados a una secuencia:

NEXTVAL(‘nombreSecuencia’) - Devuelve el próximo valor de la secuencia actualizándola.

CURRVAL(‘nombreSecuencia’) - Devuelve el valor actual de la secuencia.

SETVAL('nombreSecuencia', 4000) Actualiza la secuencia con un número específico.

CePETel

DML - Secuencias

Cuidado con las secuencias en cualquiera de sus implementaciones, cuando requerimos números consecutivos sin huecos.

CePETel

Data Manipulation Language

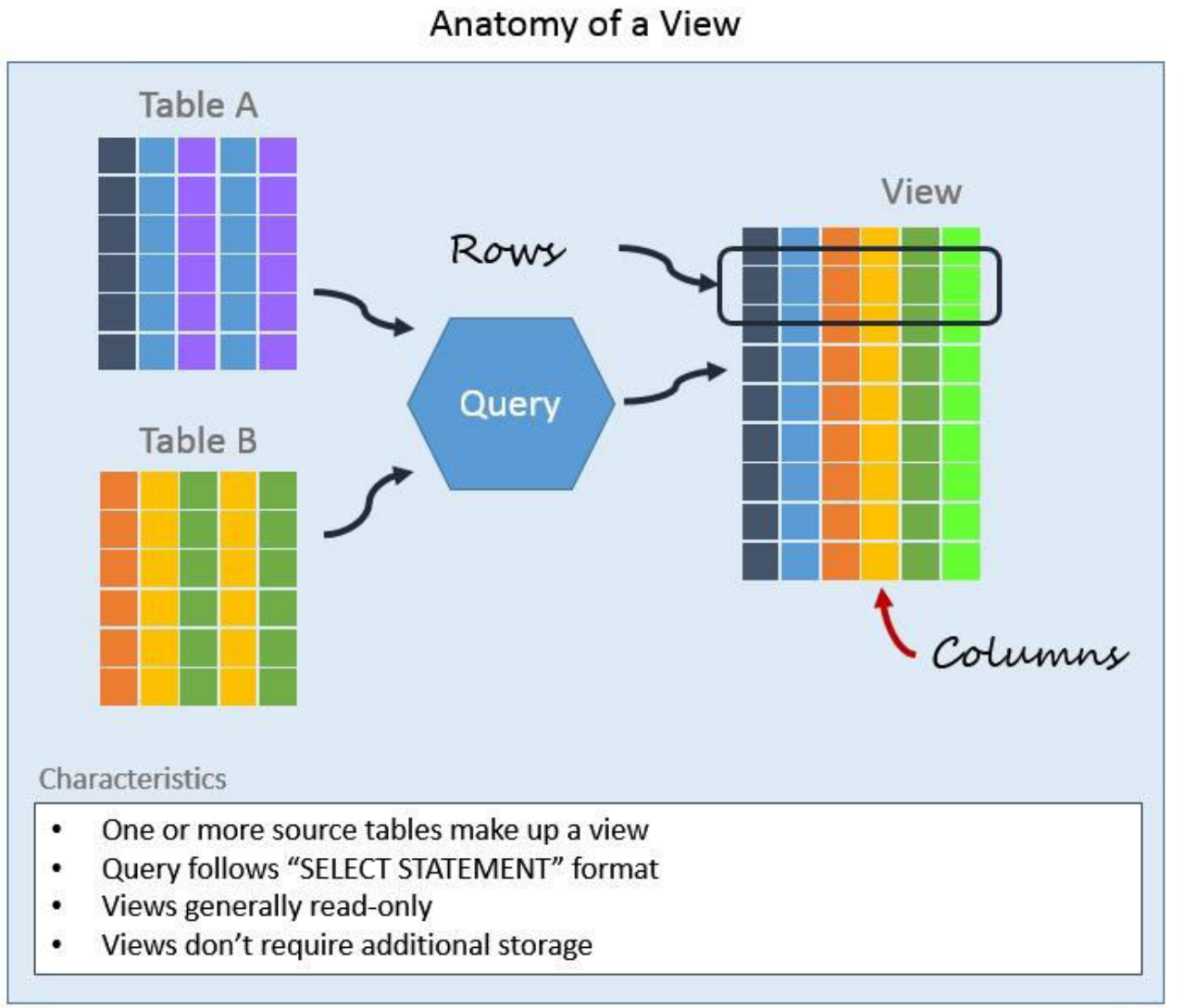
CePETel

DML – VIEWS (Vistas)

Una view es un conjunto de columnas, ya sea reales o virtuales, de una misma tabla o no, que puede contar con algún filtro para condicionar el resultado.

De esta forma, es una presentación adaptada de los datos contenidos en una o más tablas, o en otras vistas. Una vista toma la salida resultante de una consulta y la trata como una tabla.

Se pueden usar vistas en la mayoría de las situaciones en las que se pueden usar tablas.

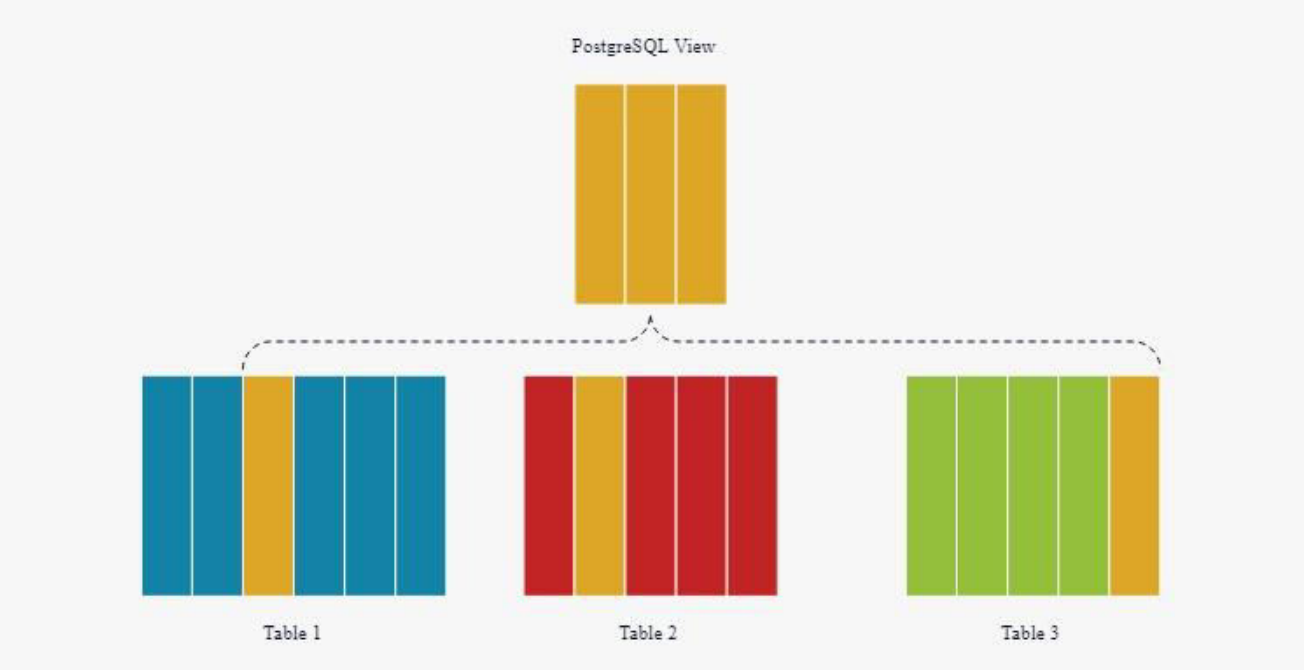


CePETel

DML – VIEWS (Vistas)

Tiene un nombre específico

- No contiene datos almacenados y no aloca espacio de almacenamiento.
- Está definida por una consulta sobre una o varias tablas.
- Solo se almacena la metadata de su definición.



DML – VIEWS (Vistas)

Las vistas se pueden utilizar para:

- Suministrar un **nivel adicional de seguridad** restringiendo el acceso a un conjunto predeterminado de filas o columnas de una tabla.
- **Ocultar la complejidad** de los datos.
- **Simplificar sentencias** al usuario.
- Presentar los datos desde una **perspectiva diferente**.
- **Aislar a las aplicaciones** de los cambios en la tabla base.

DML – VIEWS (Vistas)

RESTRICCIONES

- Tener en cuenta ciertas restricciones para el caso de Actualizaciones:
  - Si en la tabla existieran campos que no permiten nulos y en la view no aparecen, los inserts fallarían.
  - Si en la view no aparece la primary key los inserts podrían fallar.
  - Se puede borrar filas desde una view que tenga una columna virtual.
  - Con la opción WITH CHECK OPTION, se puede actualizar siempre y cuando el checkeo de la opción en el WHERE sea verdadero.

DML – VIEWS (Vistas)

Ventajas

De los puntos mencionados anteriormente, se puede apreciar lo siguiente:

- Siempre se muestran los datos actualizados.
- Simplifica el uso de consultas complejas.
- Simplifica la representación de los datos ofreciendo mas sentido lógico.
- Define un nivel mas de seguridad.
- Aísla las aplicaciones de la Base de Datos.
- Permite mayor flexibilidad.

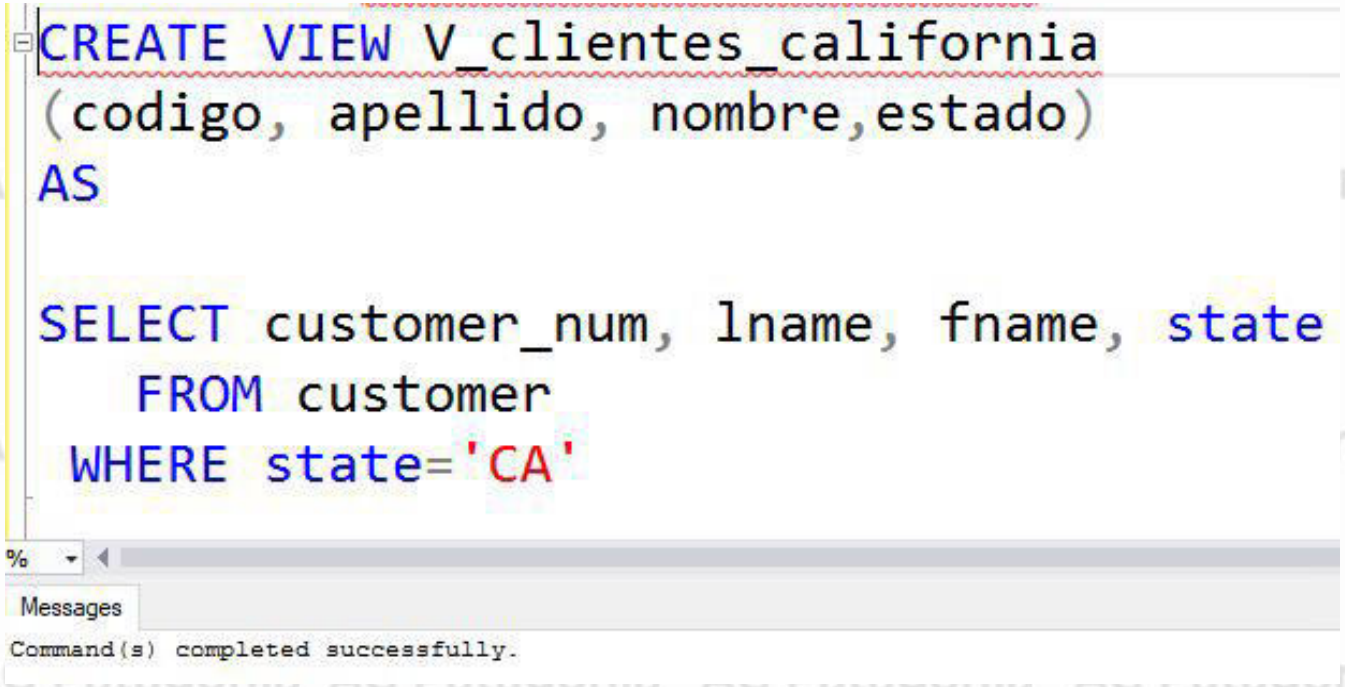
Desventajas

De los puntos mencionados anteriormente, se puede apreciar lo siguiente:

- No se pueden utilizar las sentencias **`INSERT`**, **`UPDATE`** y **`DELETE`** sobre la vista para alterar los datos.
- No mejora el rendimiento.

DML – VIEWS (Vistas)

```
CREATE VIEW V_clientes_california
(codigo, apellido, nombre)
AS

SELECT customer_num, lname, fname
       FROM customer
WHERE state='CA'
```



DML – VIEWS (Vistas)

Hacemos una consulta sobre la View v_clientes_california

```
select count(*) from v_clientes_california
```

100 %

Results | Messages

	(No column name)
1	18

Insertamos a través de la View una fila en la tabla original customer los siguientes valores (999,'Martin', 'Mihura','FL').

```
INSERT INTO v_clientes_california
(codigo,apellido,nombre,estado)
VALUES (999,'Martin', 'Mihura','FL')
```

100 %

Messages

```
(1 row(s) affected)
```

```
select count(*)
from v_clientes_california
```

100 %

Results | Messages

	(No column name)
1	18

¿Por Qué?

CePETel

DML – VIEWS (Vistas)

Hacemos una consulta sobre la View v_clientes_california

```
select *
 from v_clientes_california where codigo=101
```

100 %

Results | Messages

	codigo	apellido	nombre	estado
1	101	Pauli	Ludwig	CA

Hacemos un **Update** sobre el cliente 101 de California cambiando su estado a New York.

```
UPDATE v_clientes_california
set estado='NY'
WHERE codigo=101
```

100 %

Messages

(1 row(s) affected)

```
select *
 from v_clientes_california
 where codigo=101
```

100 %

Results | Messages

codigo	apellido	nombre	estado

¿Por Qué?

CePETel

DML – VIEWS (Vistas)

Ejecutamos un INSERT a través de una vista de una fila que la vista no puede mostrar por sus condiciones en el WHERE.

Ejecutamos un UPDATE en una fila a través de la vista cambiando un valor de un campo que es condición en el WHERE.

A través de la vista manipulamos datos que actualmente no pueden ser accedidos a través de la vista.

```
select customer_num,lname,fname,state
 from customer
 where customer_num in (101,999)
```

100 %

Results | Messages

	customer_num	lname	fname	state
1	101	Pauli	Ludwig	NY
2	999	Martin	Mihura	FL

```
select codigo,apellido,nombre,estado
from v_clientes_california
where codigo in (101,999)
```

100 %

Results | Messages

codigo	apellido	nombre	estado

¿Es esto correcto, no les hace ruido? Lo vemos en clase.-

CePETel

DML – VIEWS (Vistas)

CREATE VIEW V_clientes_california_WCK
(codigo, apellido, nombre,estado)
AS

SELECT customer_num, lname, fname,state
FROM customer
WHERE state='CA'

WITH CHECK OPTION

WITH CHECK OPTION
realiza un chequeo de integridad de los datos a insertar o modificar, los cuales deben cumplir con las condiciones del WHERE de la vista.

CePETel

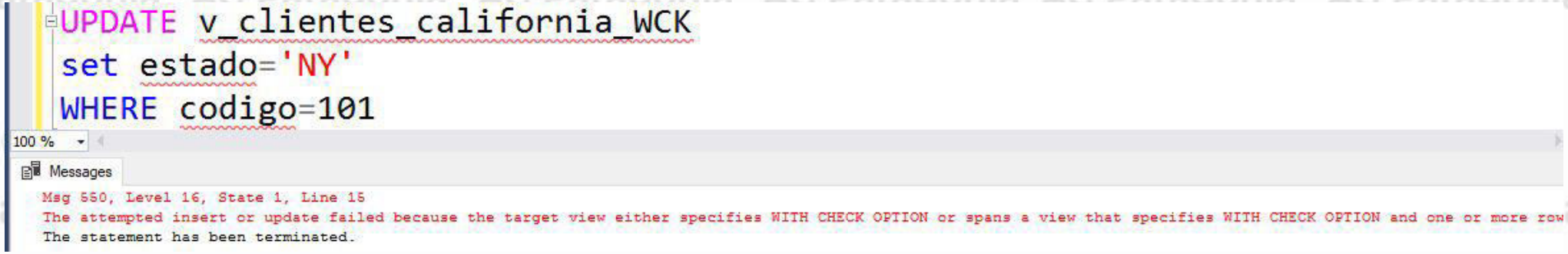
DML – VIEWS (Vistas)

CREATE VIEW V_clientes_california_WCK
(codigo, apellido, nombre,estado)
AS

SELECT customer_num, lname, fname,state
FROM customer
WHERE state='CA'

WITH CHECK OPTION

WITH CHECK OPTION
realiza un chequeo de integridad de los datos a insertar o modificar, los cuales deben cumplir con las condiciones del WHERE de la vista.



CePETel

Data Manipulation Language

CePETel

DML – Tablas Temporales

- Son tablas creadas cuyos datos son de existencia temporal.
- No son registradas en las tablas del diccionario de datos.
- No es posible alterar tablas temporarias. Si eliminarlas y crear los índices temporales que necesite una aplicación.
- Las actualizaciones a una tabla temporal podrían no generar ningún log transaccional si así se configurara.

Tipos de Tablas

- De Sesión (locales)
- Globales

Tipos de Creación

- Explícita
- Implícita

CePETel

DML – Tablas Temporales

Tipos de Tablas Temporales

De Sesión (locales)

Son visibles sólo para sus creadores durante la misma sesión (conexión) a una instancia del motor de BD.
Las tablas temporales locales se eliminan cuando el usuario se desconecta o cuando decide eliminar la tabla durante la sesión.

Globales

Las tablas temporales globales están visibles para cualquier usuario y sesión una vez creadas. Su eliminación depende del motor de base de datos que se utilice.
(En postgresql, la cláusula Global está deprecada en las tablas temporales.)

DML – Tablas Temporales

Tipos de Creación

Creación Explícita:

Este tipo de creación se realizar mediante la instrucción CREATE. De manera explícita se deberá crear la tabla indicando el nombre, sus campos, tipos de datos y restricciones.

Creación Implícita:

Se pueden crear tablas temporales a partir del resultado de una consulta SELECT.

DML – Tablas Temporales

¿Por qué utilizarlas?

1. Como almacenamiento intermedio de Consultas Muy Grandes:

Por ejemplo, se tiene una consulta SELECT que realiza **“JOINS”** con ocho tablas. Muchas veces las consultas con varios **“JOINS”** pueden funcionar de manera poco performante.

Una técnica para intentar es la de dividir una consulta grande en consultas más pequeñas. Si usamos tablas temporales, podemos crear tablas con resultados intermedios basados en consultas de menor tamaño, en lugar de intentar ejecutar una consulta única que sea demasiado grande y múltiples **“JOINS”.**

CePETel

DML – Tablas Temporales

¿Por qué utilizarlas?

2. Para optimizar accesos a una consulta varias veces en una aplicación:

Por ejemplo, usted está utilizando una consulta que tarda varios segundos en ejecutarse, pero sólo muestra un conjunto acotado de resultados, el cual desea utilizar en varias áreas de su procedimiento almacenado, pero cada vez que se llama se debe volver a ejecutar la consulta general.

Para resolver esto, puede ejecutar la consulta una sola vez en el procedimiento, llenando una tabla temporal, de esta manera se puede hacer referencia a la tabla temporal en varios lugares en su código, sin incurrir en una sobrecarga de resultados adicional.

DML – Tablas Temporales

¿Por qué utilizarlas?

3. Para almacenar resultados intermedios en una aplicación:

Por ejemplo, usted está necesita en un determinado proceso generar información que se ira actualizando y/o transformando en distintos momentos de la ejecución, sin querer actualizar o impactar a tablas reales de la BD hasta el final del procedimiento.

Para resolver esto, puede crear una tabla temporal de sesión durante la ejecución del procedimiento, realizando en ella inserciones, modificaciones, borrado y/o transformación de datos. Al llegar al final del procedimiento, con los datos existentes en la tabla temporal se actualizará la o las tablas físicas de la BD que corresponda.

DML – Tablas Temporales

Tablas Temporales de Sesión

Creación Explícita

```
CREATE TEMPORARY TABLE pending_orders(
        order_num smallint NOT NULL,
        order_date date,
        customer_num smallint NOT NULL,
        ship_instruct character varying(40),
        backlog character(1),
        po_num character varying(10),
        ship_date date,
        ship_weight numeric(8,2),
        ship_charge numeric(6,2),
        paid_date date );

INSERT INTO pending_orders
        SELECT * FROM orders WHERE paid_date IS NOT NULL
```

Creación Implícita

```
CREATE TEMPORARY TABLE pending_orders
AS
SELECT * FROM orders
WHERE paid_date IS NOT NULL
```

Tanto el ejemplo de creación explícita y el de implícita generarán la misma tabla temporal con los mismos datos.

CePETel

DML – Tablas Temporales

Tablas Temporales de Sesión

Sesión 1

stores777/postgres@PostgreSQL 12

Query Editor | Query History

```
1  CREATE TEMPORARY TABLE pending_orders(
2      order_num smallint NOT NULL,
3      order_date date,
4      customer_num smallint NOT NULL,
5      ship_instruct character varying(40),
6      backlog character(1),
7      po_num character varying(10),
8      ship_date date,
9      ship_weight numeric(8,2),
10     ship_charge numeric(6,2),
11     paid_date date );
12
13
14 INSERT INTO pending_orders
15  SELECT * FROM orders WHERE paid_date IS NOT NULL;
16
17  SELECT * FROM pending_orders
```

Data Output | Explain | Messages | Notifications

	order_num smallint	order_date date	customer_num smallint	ship_instruct character varying (40)	backlog character (1)	po_num characte
1	1001	2015-05-16	104	express	n	B77836
2	1002	2015-05-17	101	PO on box; deliver to back d...	n	9270
3	1003	2015-05-18	104	express	n	B77890
4	1005	2015-05-20	116	call before delivery	n	2865
5	1008	2015-06-03	110	closed Monday	y	LZ230
6	1009	2015-06-10	111	next door to grocery	n	4745
7	1010	2015-06-13	115	deliver 776 King St. if no ans...	n	429Q
8	1011	2015-06-14	104	express	n	B77897
9	1013	2015-06-18	104	express	n	B77930
10	1015	2015-06-23	110	closed Mondays	n	MA003
11	1018	2015-07-06	121	SW corner of Biltmore Mall	n	S22942

Sesión 2

stores777/postgres@PostgreSQL 12

Query Editor | Query History

```
1  SELECT * FROM pending_orders
```

Data Output | Explain | Messages | Notifications

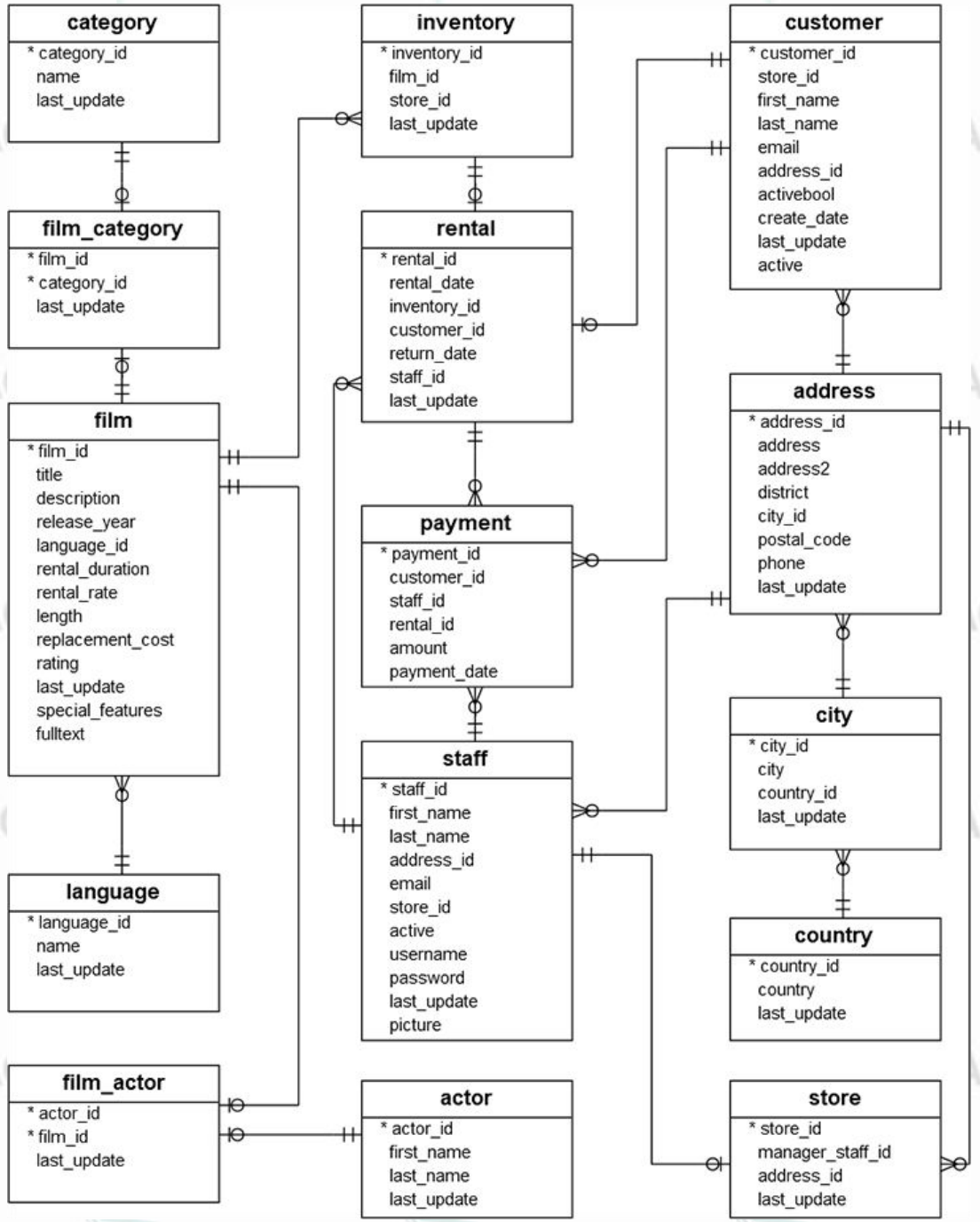
```
ERROR:  relation "pending_orders" does not exist
LINE 1:  SELECT * FROM pending_orders
                      ^
SQL state: 42P01
Character: 16
```

En este ejemplo creamos una tabla temporal de sesión en la **sesión 1**, le insertamos datos y luego consultamos la tabla temporal.

En una **sesión 2** se quiere consultar la tabla temporal creada y la misma no existe, esto ocurre porque las tablas de sesión solo viven dentro de la sesión en la que fueron creadas.

CePETel

TP N° 6: Secuencias, Vistas, Tablas Temporales



CePETel

TP N° 6: Secuencias, Vistas, Tablas Temporales

1. Crear una tabla temporal clientes a partir de la siguiente consulta: SELECT * FROM customer

2. Crear una tabla temporal clientesActivos con la misma estructura de la tabla customer. Realizar un insert masivo en la tabla #clientesActivos con todos los clientes de la tabla customer cuyo campo “active”=1.

3. Borrar de la tabla clientes (customer) todos los clientes cuyo apellido sea “Smith”. Cuantos clientes se eliminaron?

4. Modificar todos los clientes de la tabla #address, agregando un dígito 1 delante de cada número telefónico (phone), debido a un cambio de la compañía de teléfonos.

5. A partir de la tabla film_category crear una vista film_category_5 que obtenga todos los filmes cuya categoría sea igual a 5 y chequear la vista con un SELECT. Que representa el category_id=5? En que tabla esta esa información?

6. Elija una tabla del modelo, inserte una fila completa, verifque con el comando select que se haya insertado correctamente y luego elimela

CePETel
Sindicato de los Profesionales
de las Telecomunicaciones

SECRETARÍA TÉCNICA

IPEI
Instituto Profesional de Estudios e Investigación

Introducción a Bases de Datos y Programación SQL

TEMARIO

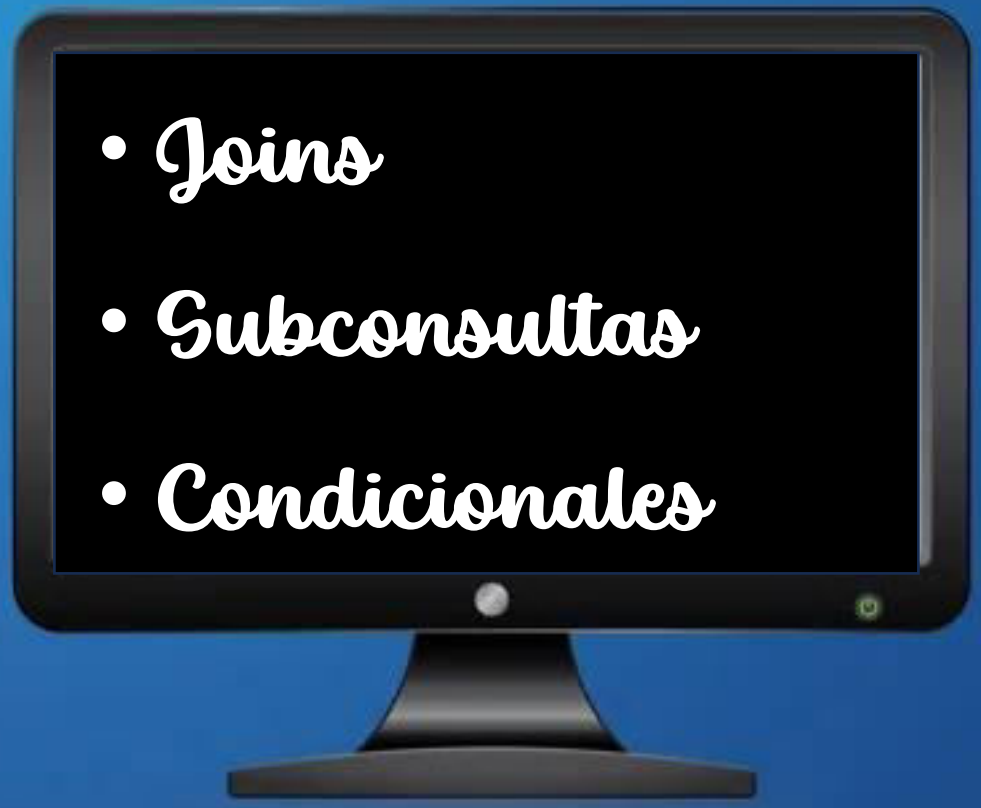
Módulo 1: Conceptos de Bases de datos y estructuras.
Módulo 2: Modelado de Datos. Normalización.
Módulo 3: DDL (Data Definition Language)
Módulo 4: DML (Data Manipulation Language) - Select
Módulo 5: DML (Update – Insert - Delete)
Módulo 6: DML (Secuencias – Vistas – Tablas temporales)
Módulo 7: DML (Joins – Subconsultas – Condicionales)
Módulo 8: DML (Funciones – Operadores)

Disertantes: Lic. Maria Trinidad Aquino – Ing. Raúl Alejandro Grassi

Introducción a Bases de Datos y Programación SQL

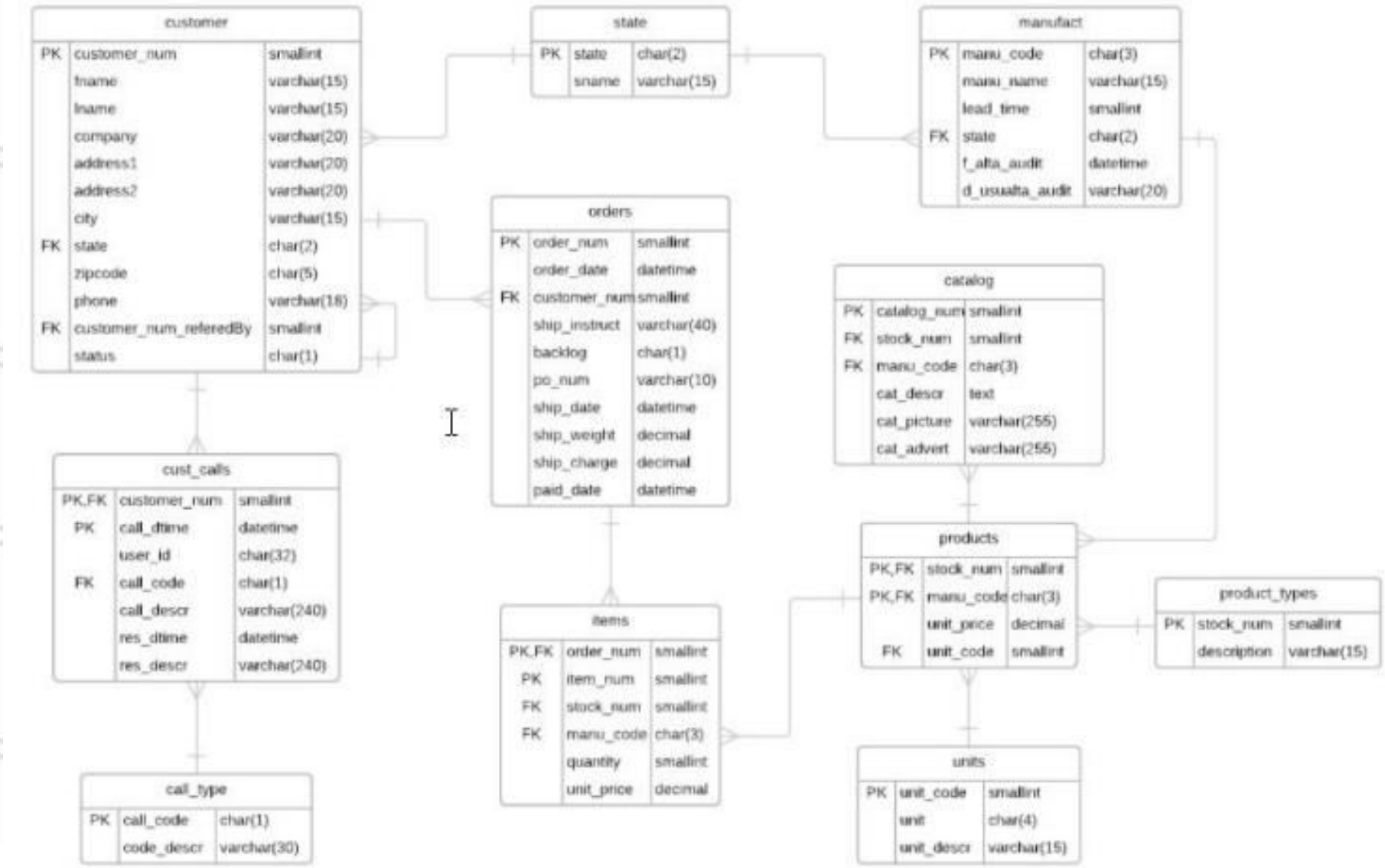
Módulo 7:
DML (Joins – Subconsultas – Condicionales)

Data Manipulation Language



Data Manipulation Language

Bases de Datos de Ejemplo



CePETel

DML - SQL

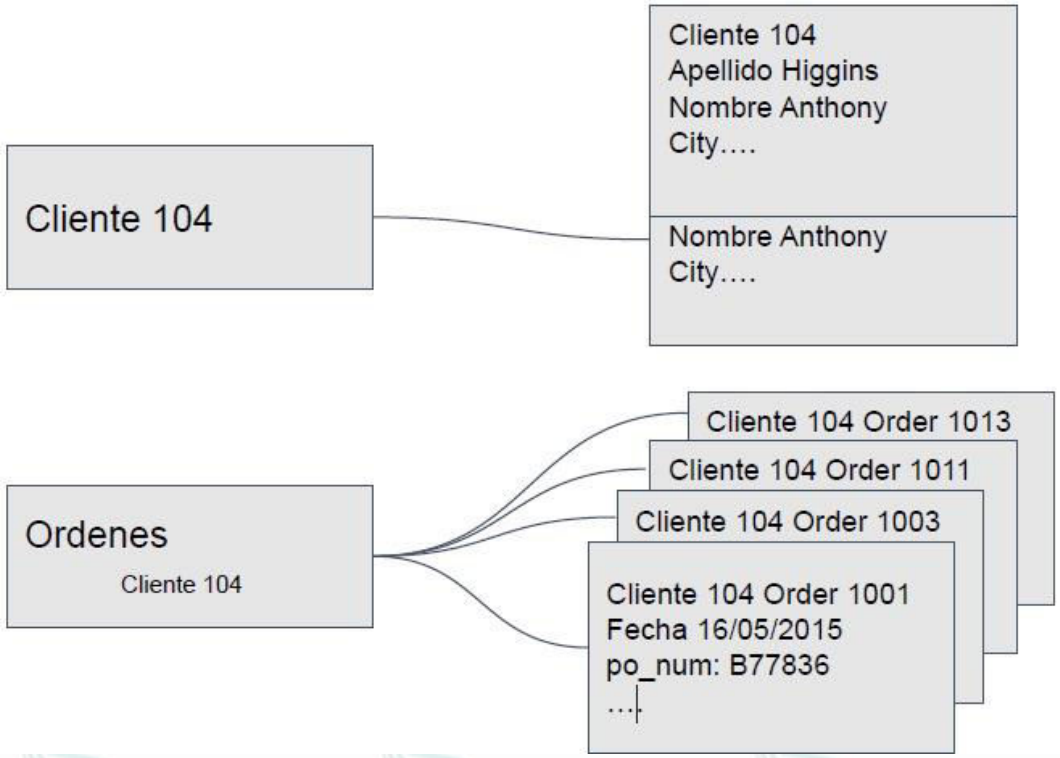
Video: Conociendo la historia SQL

DML – Data Manipulation Language

- SELECT
  - INNER JOIN
  - OUTER JOIN

Si queremos hacer coincidir filas de dos o más tablas a partir de un atributo con valores comunes, por ej.

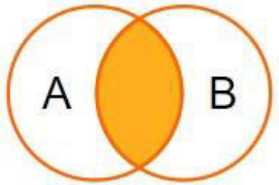
A partir del customer_num de la tabla orders obtener el lname y fname de cada cliente de la tabla customer.



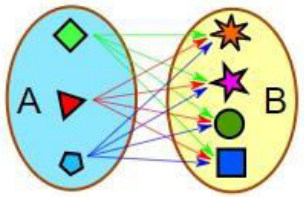
DML - SQL

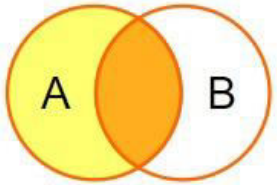
Joins del SQL

Select (campos)
From A Inner Join B
On A.Clave = B.Clave

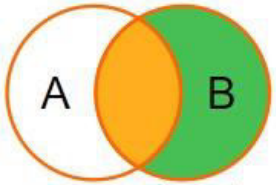


Select (campos)
From A Cross Join B

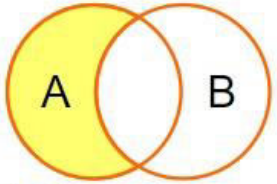




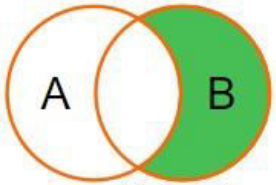
Select (campos)
From A Left Join B
On A.Clave = B.Clave



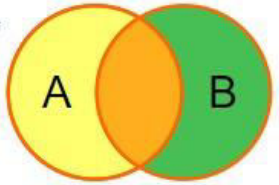
Select (campos)
From A Right Join B
On A.Clave = B.Clave



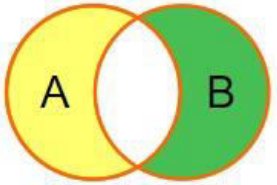
Select (campos)
From A Left Join B
On A.Clave = B.Clave
Where B.Clave is Null



Select (campos)
From A Right Join B
On A.Clave = B.Clave
Where A.Clave is Null



Select (campos)
From A Full Outer Join B
On A.Clave = B.Clave



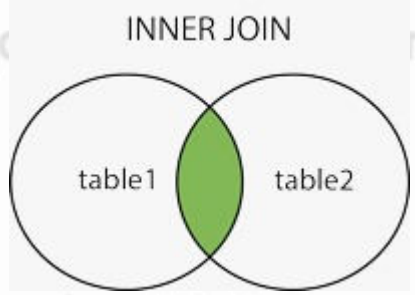
Select (campos)
From A Full Outer Join B
On A.Clave = B.Clave
Where (A.Clave is Null) Or (B.Clave is Null)

CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla

INNER JOIN (clave simple)



Se realiza un macheo de las filas de una tabla que coincidan a través de un atributo o combinación de atributos con filas de otras tablas.

El INNER JOIN solo devolverá las filas que coincidan.

```
SELECT c.customer_num, fname,lname,
order_num, order_date
FROM customer c INNER JOIN orders o
      ON (c.customer_num = o.customer_num)
```

Query Editor | Query History

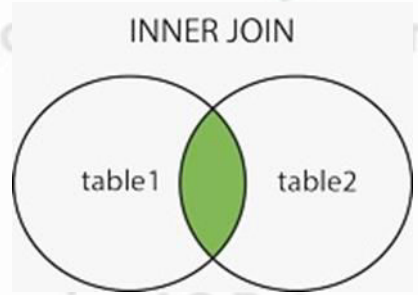
```
SELECT c.customer_num, fname,lname, order_num, order_date
FROM customer c INNER JOIN orders o
                        ON (c.customer_num = o.customer_num)
```

Data Output | Explain | Messages | Notifications

	customer_num smallint	fname character varying (15)	lname character varying (15)	order_num smallint	order_date date
1	104	Anthony	Higgins	1001	2015-05-16
2	101	Ludwig	Pauli	1002	2015-05-17
3	104	Anthony	Higgins	1003	2015-05-18
4	106	George	Watson	1004	2015-05-18
5	116	Jean	Parmelee	1005	2015-05-20
6	112	Margaret	Lawson	1006	2015-05-26
7	117	Arnold	Sipes	1007	2015-05-27
8	110	Roy	Jaeger	1008	2015-06-03
9	111	Frances	Keyes	1009	2015-06-10

DML - SQL

SQL – Operador SELECT de más de 1 tabla



INNER JOIN (clave compuesta)

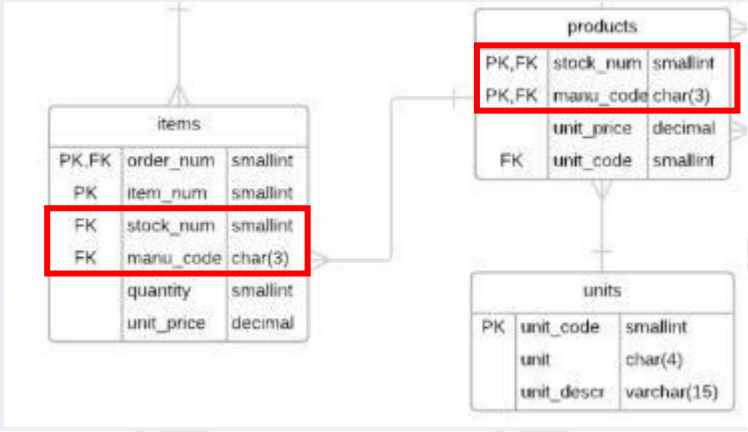
Queremos obtener el código de producto, total vendido y el código de unidad de medida.
Para esto deberemos tomar la unit_code de la tabla products.

```
1  SELECT i.stock_num, i.manu_code, SUM(i.unit_price*quantity) tot_producto, unit_code
2  FROM items i INNER JOIN products p
3                          ON (i.stock_num = p.stock_num and i.manu_code=p.manu_code)
4  GROUP BY i.stock_num, i.manu_code,unit_code
```

Data Output | Explain | Messages | Notifications

	stock_num smallint	manu_code character (3)	tot_producto numeric	unit_code smallint
1	110	SHM	228.00	16
2	6	SMT	144.00	12
3	1	SMT	19525.00	6
4	111	SHM	4499.91	18
5	2	HRO	252.00	13
6	205	ANZ	1248.00	13
7	5	ANZ	3484.80	19
8	114	PRC	120.00	7
9	4	HRO	960.00	13
10	307	PRC	1000.00	19
11	109	PRC	270.00	17
12	6	ANZ	384.00	12

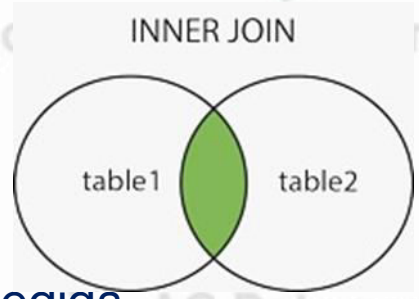
Por qué utilizamos alias de tabla en algunas columnas?



CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla



INNER JOIN (más de dos tablas)

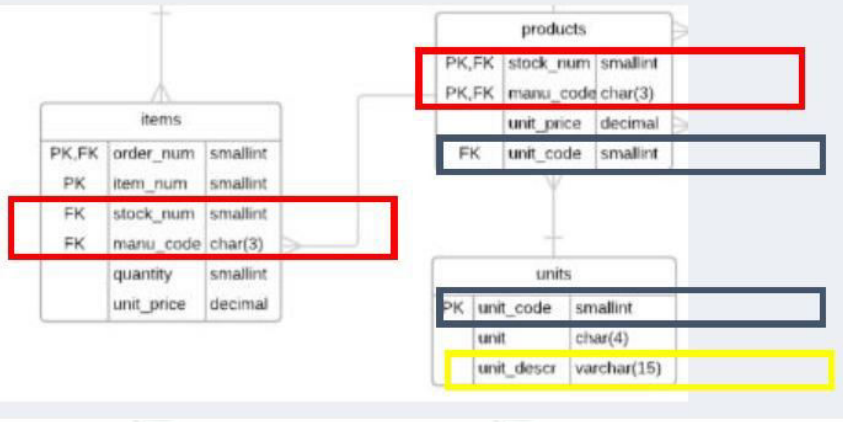
Obtener el código de producto, total vendido y la descripción de la unidad de medida.
Para esto deberemos tomar la unit_code de la tabla products y el unit_descr de la tabla units.

Query Editor | Query History

```
1  SELECT i.stock_num, i.manu_code, SUM(i.unit_price*quantity) tot_producto, unit_descr
2
3  FROM items i INNER JOIN products p ON (i.stock_num = p.stock_num and i.manu_code=p.manu_code)
4               INNER JOIN units u ON (p.unit_code = u.unit_code)
5
6  GROUP BY i.stock_num, i.manu_code,unit_descr
```

Data Output | Explain | Messages | Notifications

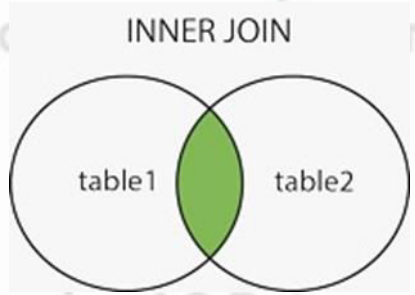
	stock_num smallint	manu_code character (3)	tot_producto numeric	unit_descr character varying (15)
1	110	HSK	308.00	4/case
2	103	PRC	80.00	each
3	306	SHM	190.00	each
4	101	SHM	272.00	4/box
5	301	KAR	1392.00	each
6	307	PRC	1000.00	each
7	5	ANZ	3484.80	each
8	304	HRO	280.00	10/box
9	301	SHM	408.00	each
10	5	NRG	3500.00	each
11	201	NKL	750.00	each
12	3	HSK	720.00	12/case
13	104	PRC	232.00	each
14	205	ANZ	1248.00	24/case
15	110	SHM	228.00	4/case



CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla



INNER JOIN (más de dos tablas)

Obtener el cod_cliente, apellido y nombre de cliente, orden de compra, fecha emisión, nro. de stock, código fabricante, nombre_fabricante, descripción tipo producto, nro de ítem, cantidad y precio unitario.

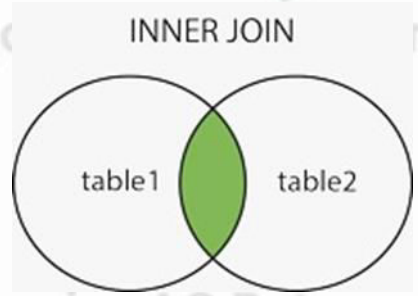
```
1  SELECT c.customer_num, fname,lname, o.order_num, order_date, item_num, i.stock_num, i.manu_code, manu_name, p.description,
2         quantity, unit_price, (quantity*unit_price) tot_item
3
4  FROM customer c INNER JOIN orders o ON (c.customer_num = o.customer_num)
5                  INNER JOIN items i  ON (o.order_num = i.order_num)
6                  INNER JOIN product_types p ON (i.stock_num = p.stock_num)
7                  INNER JOIN manufact m ON (i.manu_code = m.manu_code)
8
```

Data Output | Explain | Messages | Notifications

	customer_num smallint	fname character va	lname character varyin	order_num smallint	order_date date	item_num smallint	stock_num smallint	manu_code character (3)	manu_name character varying (	description character varying (15)	quantity smallint	unit_price numeric (8,2)	tot_item numeric
1	104	Anthony	Higgins	1001	2015-05-16	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
2	101	Ludwig	Pauli	1002	2015-05-17	2	3	HSK	Husky	baseball bat	1	240.00	240.00
3	101	Ludwig	Pauli	1002	2015-05-17	1	4	HSK	Husky	football	1	960.00	960.00
4	104	Anthony	Higgins	1003	2015-05-18	3	5	ANZ	Anza	tennis racquet	5	99.00	495.00
5	104	Anthony	Higgins	1003	2015-05-18	2	8	ANZ	Anza	volleyball	1	840.00	840.00
6	104	Anthony	Higgins	1003	2015-05-18	1	9	ANZ	Anza	volleyball net	1	20.00	20.00
7	106	George	Watson	1004	2015-05-18	4	1	HSK	Husky	baseball gloves	1	800.00	800.00
8	106	George	Watson	1004	2015-05-18	3	3	HSK	Husky	baseball bat	1	240.00	240.00
9	106	George	Watson	1004	2015-05-18	2	2	HRO	Hero	baseball	1	126.00	126.00
10	106	George	Watson	1004	2015-05-18	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
11	116	Jean	Parmelee	1005	2015-05-20	4	6	ANZ	Anza	tennis ball	1	48.00	48.00
12	116	Jean	Parmelee	1005	2015-05-20	3	6	SMT	Smith	tennis ball	1	36.00	36.00
13	116	Jean	Parmelee	1005	2015-05-20	2	5	ANZ	Anza	tennis racquet	10	198.00	1980.00
14	116	Jean	Parmelee	1005	2015-05-20	1	5	NRG	Norge	tennis racquet	10	280.00	2800.00
15	112	Margaret	Lawson	1006	2015-05-26	5	6	ANZ	Anza	tennis ball	1	48.00	48.00
16	112	Margaret	Lawson	1006	2015-05-26	4	6	SMT	Smith	tennis ball	1	36.00	36.00
17	112	Margaret	Lawson	1006	2015-05-26	3	5	ANZ	Anza	tennis racquet	5	99.00	495.00

DML - SQL

SQL – Operador SELECT de más de 1 tabla



INNER JOIN (más de dos tablas con condiciones adicionales en WHERE)

Mismo ejemplo anterior, aunque con condiciones: fabricante de nombre Hero, órdenes del mes de mayo del 2015.

```
1  SELECT c.customer_num, fname,lname, o.order_num, order_date, item_num, i.stock_num, i.manu_code, manu_name, p.description,
2         quantity, unit_price, (quantity*unit_price) tot_item
3
4  FROM customer c INNER JOIN orders o ON (c.customer_num = o.customer_num)
5                  INNER JOIN items i  ON (o.order_num = i.order_num)
6                  INNER JOIN product_types p ON (i.stock_num = p.stock_num)
7                  INNER JOIN manufact m ON (i.manu_code = m.manu_code)
8  WHERE manu_name='Hero' AND DATE_PART('month', order_date)=5 AND DATE_PART('year', order_date)=2015
9
```

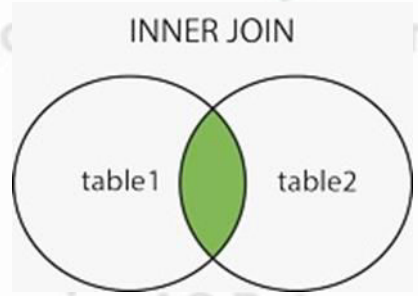
Data Output | Explain | Messages | Notifications

	customer_num smallint	fname character var	lname character var	order_num smallint	order_date date	item_num smallint	stock_num smallint	manu_code character (3)	manu_name character varyin	description character varying (15)	quantity smallint	unit_price numeric (8,2)	tot_item numeric
1	104	Anthony	Higgins	1001	2015-05-16	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
2	106	George	Watson	1004	2015-05-18	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
3	117	Arnold	Sipes	1007	2015-05-27	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
4	106	George	Watson	1004	2015-05-18	2	2	HRO	Hero	baseball	1	126.00	126.00
5	117	Arnold	Sipes	1007	2015-05-27	2	2	HRO	Hero	baseball	1	126.00	126.00
6	117	Arnold	Sipes	1007	2015-05-27	4	4	HRO	Hero	football	1	480.00	480.00
7	117	Arnold	Sipes	1007	2015-05-27	5	7	HRO	Hero	basketball	1	600.00	600.00

CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla



INNER JOIN (más de dos tablas con condiciones adicionales en WHERE)

Mismo ejemplo anterior, aunque con condiciones: cliente de nombre Husky, órdenes del mes de mayo del 2015 y precio unitario mayor que 126.

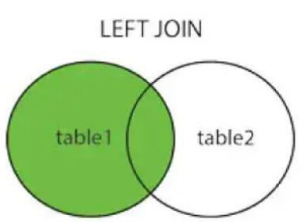
```
1  SELECT c.customer_num, fname,lname, o.order_num, order_date, item_num, i.stock_num, i.manu_code, manu_name, p.description,
2         quantity, unit_price, (quantity*unit_price) tot_item
3
4  FROM customer c INNER JOIN orders o ON (c.customer_num = o.customer_num)
5                  INNER JOIN items i  ON (o.order_num = i.order_num)
6                  INNER JOIN product_types p ON (i.stock_num = p.stock_num)
7                  INNER JOIN manufact m ON (i.manu_code = m.manu_code)
8  WHERE manu_name='Hero' AND DATE_PART('month', order_date)=5 AND DATE_PART('year', order_date)=2015 AND unit_price > 126
9
```

Data Output | Explain | Messages | Notifications

	customer_num smallint	fname character varyi	lname character var	order_num smallint	order_date date	item_num smallint	stock_num smallint	manu_code character (3)	manu_name character va	description character varying (15	quantity smallint	unit_price numeric (8,2)	tot_item numeric
1	104	Anthony	Higgins	1001	2015-05-16	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
2	106	George	Watson	1004	2015-05-18	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
3	117	Arnold	Sipes	1007	2015-05-27	1	1	HRO	Hero	baseball gloves	1	250.00	250.00
4	117	Arnold	Sipes	1007	2015-05-27	4	4	HRO	Hero	football	1	480.00	480.00
5	117	Arnold	Sipes	1007	2015-05-27	5	7	HRO	Hero	basketball	1	600.00	600.00

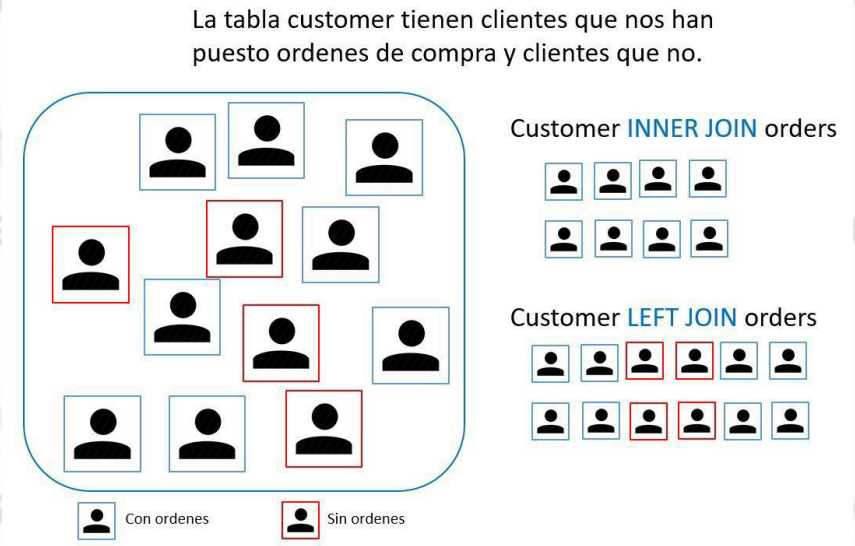
DML - SQL

SQL – Operador SELECT de más de 1 tabla



OUTER JOIN

El Outer join, mostrará todas las filas de la tabla dominante macheen o no con la otra tabla.
El Outer puede ser LEFT. (tabla izquierda dominante), RIGHT (tabla derecha dominante) o FULL (ambas tablas dominantes)




```
SELECT c.customer_num, fname,lname, COUNT(order_num) cantOrdenes
FROM customer c INNER JOIN orders o
                    ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

```
SELECT c.customer_num, fname,lname,COUNT(order_num)
cantOrdenes
FROM customer c LEFT JOIN orders o
                    ON (c.customer_num = o.customer_num)
GROUP BY c.customer_num, fname,lname
```

DML - SQL

SQL – Operador SELECT de más de 1 tabla

OUTER JOIN

Comparemos resultados.
Inner Join 17 clientes que compraron.

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM customer c JOIN orders o
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
16
```

Data Output | Explain | Messages | Notifications

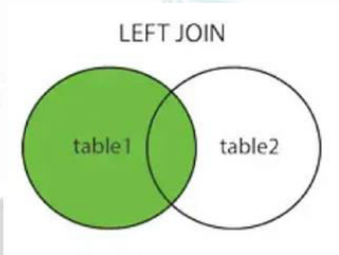
	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
1	104	Anthony	Higgins	4
2	117	Arnold	Sipes	2
3	106	George	Watson	2
4	110	Roy	Jaeger	2
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	116	Jean	Parmelee	1
9	120	Fred	Jewell	1
10	111	Frances	Keyes	1
11	123	Marvin	Hanlon	1
12	126	Eileen	Neelie	1
13	122	Cathy	O Brian	1
14	119	Bob	Shorter	1
15	101	Ludwig	Pauli	1
16	115	Alfred	Grant	1
17	112	Margaret	Lawson	1

Left Join 28 clientes, 17 con cantOrdenes mayor que 0 y 11 clientes con cantOrdenes=0

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM customer c LEFT JOIN orders o
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	120	Fred	Jewell	1
9	101	Ludwig	Pauli	1
10	115	Alfred	Grant	1
11	112	Margaret	Lawson	1
12	119	Bob	Shorter	1
13	116	Jean	Parmelee	1
14	111	Frances	Keyes	1
15	123	Marvin	Hanlon	1
16	126	Eileen	Neelie	1
17	122	Cathy	O Brian	1
18	107	Charles	Ream	0
19	102	Carole	Sadler	0
20	109	Jane	Miller	0
21	118	Dick	Baxter	0
22	108	Donald	Quinn	0
23	113	Lana	Beatty	0
24	125	James	Henry	0
25	114	Frank	Albertson	0
26	128	Frank	Lessor	0
27	103	Philip	Currie	0
28	105	Raymond	Vector	0



CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla

OUTER JOIN

Comparemos resultados con un RIGHT JOIN
Inner Join 17 clientes que compraron.

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM customer c JOIN orders o
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
16
```

Data Output | Explain | Messages | Notifications

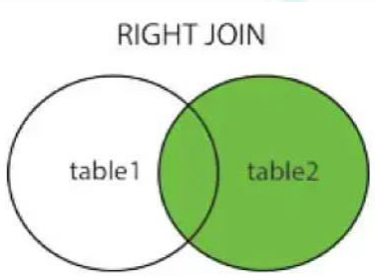
	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
1	104	Anthony	Higgins	4
2	117	Arnold	Sipes	2
3	106	George	Watson	2
4	110	Roy	Jaeger	2
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	116	Jean	Parmelee	1
9	120	Fred	Jewell	1
10	111	Frances	Keyes	1
11	123	Marvin	Hanlon	1
12	126	Eileen	Neelie	1
13	122	Cathy	O Brian	1
14	119	Bob	Shorter	1
15	101	Ludwig	Pauli	1
16	115	Alfred	Grant	1
17	112	Margaret	Lawson	1

Right Join 28 clientes, 17 con cantOrdenes mayor que 0 y 11 clientes con cantOrdenes=0

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM orders o RIGHT JOIN customer c
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	120	Fred	Jewell	1
9	101	Ludwig	Pauli	1
10	115	Alfred	Grant	1
11	112	Margaret	Lawson	1
12	119	Bob	Shorter	1
13	116	Jean	Parmelee	1
14	111	Frances	Keyes	1
15	123	Marvin	Hanlon	1
16	126	Eileen	Neelie	1
17	122	Cathy	O Brian	1
18	107	Charles	Ream	0
19	102	Carole	Sadler	0
20	109	Jane	Miller	0
21	118	Dick	Baxter	0
22	108	Donald	Quinn	0
23	113	Lana	Beatty	0
24	125	James	Henry	0
25	114	Frank	Albertson	0
26	128	Frank	Lessor	0
27	103	Philip	Currie	0
28	105	Raymond	Vector	0



Observamos que para utilizar un RIGHT JOIN y obtener el mismo resultado, lo único que hicimos fue invertir el orden de las tablas.

CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla

OUTER JOIN

Comparemos resultados con un RIGHT JOIN
Inner Join 17 clientes que compraron.

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM customer c JOIN orders o
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
16
```

Data Output | Explain | Messages | Notifications

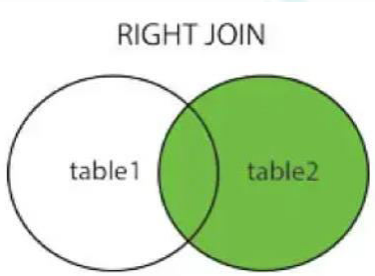
	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
1	104	Anthony	Higgins	4
2	117	Arnold	Sipes	2
3	106	George	Watson	2
4	110	Roy	Jaeger	2
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	116	Jean	Parmelee	1
9	120	Fred	Jewell	1
10	111	Frances	Keyes	1
11	123	Marvin	Hanlon	1
12	126	Eileen	Neelie	1
13	122	Cathy	O Brian	1
14	119	Bob	Shorter	1
15	101	Ludwig	Pauli	1
16	115	Alfred	Grant	1
17	112	Margaret	Lawson	1

Right Join 28 clientes, 17 con cantOrdenes mayor que 0 y 11 clientes con cantOrdenes=0

```
11 SELECT c.customer_num, fname,lname,COUNT(order_num) cant_ordenes
12 FROM orders o RIGHT JOIN customer c
13 ON (c.customer_num = o.customer_num)
14 GROUP BY c.customer_num, fname, lname
15 ORDER BY 4 DESC
```

Data Output | Explain | Messages | Notifications

	customer_num [PK] smallint	fname character varying (15)	lname character varying (15)	cant_ordenes bigint
5	127	Kim	Satifer	1
6	124	Chris	Putnum	1
7	121	Jason	Wallack	1
8	120	Fred	Jewell	1
9	101	Ludwig	Pauli	1
10	115	Alfred	Grant	1
11	112	Margaret	Lawson	1
12	119	Bob	Shorter	1
13	116	Jean	Parmelee	1
14	111	Frances	Keyes	1
15	123	Marvin	Hanlon	1
16	126	Eileen	Neelie	1
17	122	Cathy	O Brian	1
18	107	Charles	Ream	0
19	102	Carole	Sadler	0
20	109	Jane	Miller	0
21	118	Dick	Baxter	0
22	108	Donald	Quinn	0
23	113	Lana	Beatty	0
24	125	James	Henry	0
25	114	Frank	Albertson	0
26	128	Frank	Lessor	0
27	103	Philip	Currie	0
28	105	Raymond	Vector	0



Observamos que para utilizar un RIGHT JOIN y obtener el mismo resultado, lo único que hicimos fue invertir el orden de las tablas.

CePETel

DML - SQL

SQL – Operador SELECT de más de 1 tabla

JOIN AUTO REFERENCIADO (SELF REFERENCING JOIN)

La tabla customer tiene un atributo customer_num_referedBy, que nos indica quién fue el cliente que lo referencio.

Podríamos armar una consulta que nos diga nombre y apellido del referido y de quien lo referencio.

```
27  SELECT c1.fname nombreReferido, c1.lname apellidoReferido, c2.fname nombreReferente, c2.lname apellidoReferente
28  FROM customer c1 JOIN customer c2 ON (c1.customer_num_referedby=c2.customer_num)
29
```

Data Output | Explain | Messages | Notifications

	nombrereferido character varying (15)	apellidoreferido character varying (15)	nombrereferente character varying (15)	apellidoreferente character varying (15)
1	Carole	Sadler	Ludwig	Pauli
2	Philip	Currie	Ludwig	Pauli
3	Anthony	Higgins	Philip	Currie
4	Raymond	Vector	Philip	Currie
5	George	Watson	Philip	Currie
6	Donald	Quinn	Charles	Ream
7	Jane	Miller	Charles	Ream
8	Lana	Beatty	Frances	Keyes
9	Frank	Albertson	Frances	Keyes
10	Jean	Parmelee	Alfred	Grant
11	Arnold	Sipes	Alfred	Grant
12	Dick	Baxter	Alfred	Grant

DML - SQL

SQL – Operador SELECT de más de 1 tabla

JOIN (COLUMNAS AMBIGUAS)

Cuando un atributo existe en más de una tabla del SELECT, es obligatorio identificar de qué tabla lo estamos tomando, utilizando el punto como separador (dot notation) tabla.columna o alias.columna.

```
23  SELECT customer_num, fname, lname, order_num
24  FROM customer c LEFT JOIN orders o
25  ON (c.customer_num = o.customer_num)
```

Data Output | Explain | Messages | Notifications

```
ERROR:  column reference "customer_num" is ambiguous
LINE 1: SELECT customer_num, fname, lname, order_num
               ^
SQL state: 42702
Character: 8
```

```
23  SELECT c.customer_num, fname, lname, order_num
24  FROM customer c LEFT JOIN orders o
25  ON (c.customer_num = o.customer_num)
```

Data Output | Explain | Messages | Notifications

	customer_num smallint	fname character varying (15)	lname character varying (15)	order_num smallint
11	104	Anthony	Higgins	1011
12	117	Arnold	Sipes	1012
13	104	Anthony	Higgins	1013
14	106	George	Watson	1014
15	110	Roy	Jaeger	1015
16	119	Bob	Shorter	1016
17	120	Fred	Jewell	1017
18	121	Jason	Wallack	1018
19	122	Cathy	O Brian	1019
20	123	Marvin	Hanlon	1020
21	124	Chris	Putnum	1021
22	126	Eileen	Neelie	1022
23	127	Kim	Satifer	1023
24	109	Jane	Miller	[null]
25	102	Carole	Sadler	[null]

DML - SQL

SQL – Operador SELECT de más de 1 tabla

PRODUCTO CARTESIANO

El producto cartesiano es muy costoso para el motor de base de datos. En el caso que deban realizar alguno, deben tratar de achicar el working set lo máximo que puedan, proyectando solo las columnas que necesiten y condicionando con WHERE lo que pudiesen. RECOMENDACIÓN NO LO UTILICEN.

```
31
32  SELECT *
33  FROM customer c, orders o
```

Data Output | Explain | Messages | Notifications

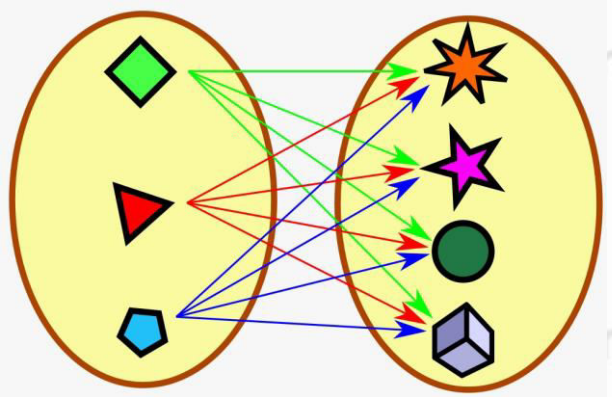
	customer_num smallint	fname character varying (15)	lname character varying (15)	company character varying (20)	address1 character varying (20)	address2 character varying (20)	city character varying (15)	state character (2)	zipcode character (5)	phone
1	101	Ludwig	Pauli	All Sports Supplies	213 Erstwild Court		Sunnyvale	CA	94086	1408-
2	102	Carole	Sadler	Sports Spot	785 Geary St		San Francisco	CA	94117	1415-
3	103	Philip	Currie	Phil s Sports	654 Poplar	P. O. Box 3498	Palo Alto	CA	94303	1415-
4	104	Anthony	Higgins	Play Ball!	East Shopping Cntr.	422 Bay Road	Redwood City	CA	94026	1415-
5	105	Raymond	Vector	Los Altos Sports	1899 La Loma Drive		Los Altos	CA	94022	1415-
6	106	George	Watson	Watson & Son	1143 Carver Place		Mountain View	CA	94063	1415-
7	107	Charles	Ream	Athletic Supplies	41 Jordan Avenue		Palo Alto	CA	94304	1415-
8	108	Donald	Quinn	Quinn s Sports	587 Alvarado		Redwood City	CA	94063	1415-
9	109	Jane	Miller	Sport Stuff	Mayfair Mart	7345 Ross Blvd.	Sunnyvale	CA	94086	1408-
10	110	Roy	Jaeger	AA Athletics	520 Topaz Way		Redwood City	CA	94062	1415-
11	111	Frances	Keyes	Sports Center	3199 Sterling Court		Sunnyvale	CA	94085	1408-
12	113	Lana	Beatty	Sportstown	654 Oak Grove		Menlo Park	CA	94025	1415-
13	114	Frank	Albertson	Sporting Place	947 Waverly Place		Redwood City	CA	94062	1415-
14	115	Alfred	Grant	Gold Medal Sports	776 Gary Avenue		Menlo Park	CA	94025	1415-
15	116	Jean	Parmelee	Olympic City	1104 Spinosa Drive		Mountain View	CA	94040	1415-
16	117	Arnold	Sipes	Kids Korner	850 Lytton Court		Redwood City	CA	94063	1415-
17	118	Dick	Baxter	Blue Ribbon Sports	5427 College		Oakland	CA	94609	1415-
18	119	Bob	Shorter	The Triathletes Club	2405 Kings Highway		Cherry Hill	NJ	8002	1609-
19	120	Fred	Jewell	Century Pro Shop	6627 N. 17th Way		Phoenix	AZ	85016	1602-
20	121	Jason	Wallack	City Sports	Lake Biltmore Mall	350 W. 23rd Street	Wilmington	DE	19898	1302-
21	122	Cathy	O Brian	The Sporting Life	543 Nassau Street		Princeton	NJ	8540	1609-
22	123	Marvin	Hanlon	Bay Sports	10100 Bay Meadows Ro	Suite 1020	Jacksonville	FL	32256	1904-
23	124	Chris	Putnum	Putnum s Putters	4715 S.E. Adams Blvd	Suite 909C				

✔ Successfully run. Total query runtime: 2 secs 209 msec. 644 rows affected.

Clientes (12 cols)+Ordenes(10 cols)= 22 cols

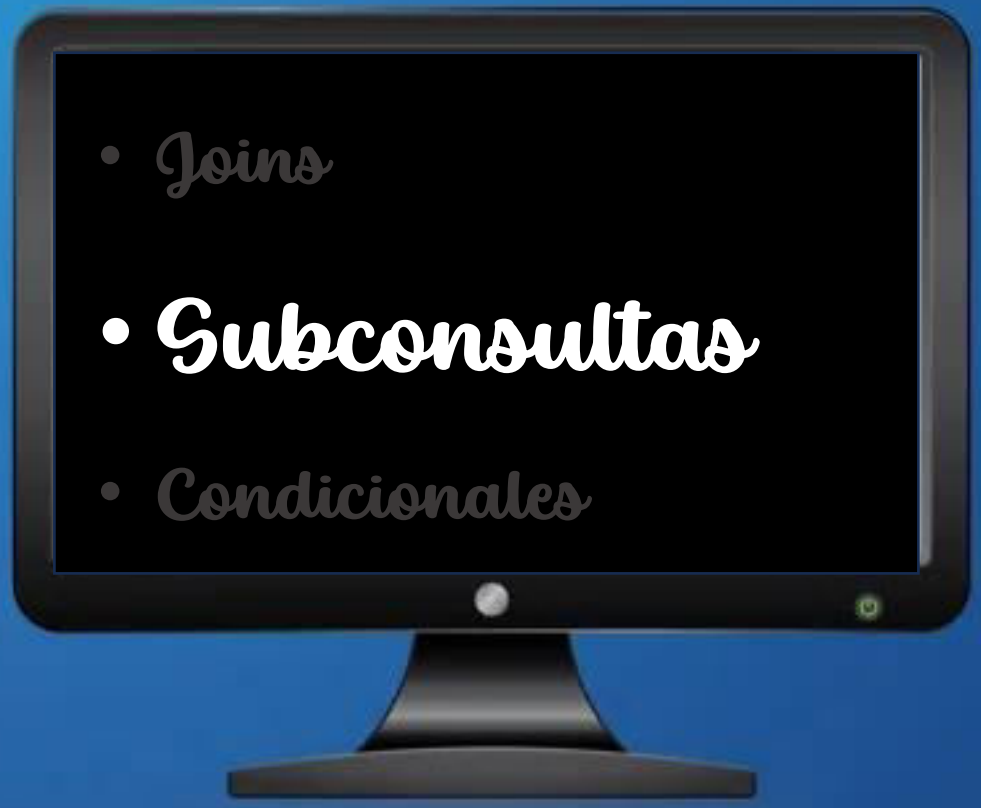
28 clientes * 23 ordenes =644 filas

¿Qué pasaría si fuesen 1000 clientes con 100000 ordenes?



CePETel

Data Manipulation Language



DML - SQL

SQL – Subquery en INSERT

```
40  CREATE TABLE closed_orders
41  (
42      order_num smallint NOT NULL,
43      order_date date,
44      customer_num smallint NOT NULL,
45      ship_instruct character varying(40) ,
46      backlog character(1) ,
47      po_num character varying(10) ,
48      ship_date date,
49      ship_weight numeric(8,2),
50      ship_charge numeric(6,2),
51      paid_date date,
52      CONSTRAINT orders_pkey2 PRIMARY KEY (order_num),
53      CONSTRAINT orders_customer_num_fkey2 FOREIGN KEY (customer_num)
54          REFERENCES customer (customer_num)
55  );
56
57  INSERT INTO closed_orders
58  SELECT * FROM orders
59  WHERE paid_date IS NOT null;
60
61  SELECT * FROM closed_orders;
```

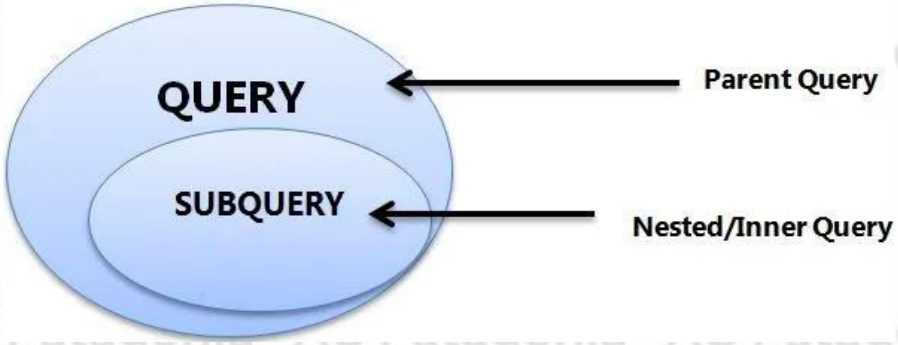
Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	order_date date	customer_num smallint	ship_instruct character varying (40)	backlog character (1)	po_num character varying (10)	ship_date date	ship_weight numeric (8,2)	ship_charge numeric (6,2)	paid_date date
1	1001	2015-05-16	104	express	n	B77836	2015-05-28	20.40	10.00	2015-07-18
2	1002	2015-05-17	101	PO on box; deliver to back d...	n	9270	2015-05-22	50.60	15.30	2015-05-30
3	1003	2015-05-18	104	express	n	B77890	2015-05-19	35.60	10.80	2015-06-10
4	1005	2015-05-20	116	call before delivery	n	2865	2015-06-05	80.80	16.20	2015-06-17
5	1008	2015-06-03	110	closed Monday	y	LZ230	2015-07-02	45.60	13.80	2015-07-17
6	1009	2015-06-10	111	next door to grocery	n	4745	2015-06-17	20.40	10.00	2015-08-17
7	1010	2015-06-13	115	deliver 776 King St. if no ans...	n	429Q	2015-06-25	40.60	12.30	2015-08-18
8	1011	2015-06-14	104	express	n	B77897	2015-06-29	10.40	5.00	2015-08-25
9	1013	2015-06-18	104	express	n	B77930	2015-07-06	60.80	12.20	2015-07-27
10	1015	2015-06-23	110	closed Mondays	n	MA003	2015-07-12	20.60	6.30	2015-08-27
11	1018	2015-07-06	121	SW corner of Biltmore Mall	n	S22942	2015-07-09	70.50	20.00	2015-08-02

```
INSERT INTO closed_orders
SELECT * FROM orders
WHERE paid_date IS NOT null
```

En una tabla creada previamente con la misma estructura que la tabla ordenes, insertamos las filas que devuelve el SELECT.

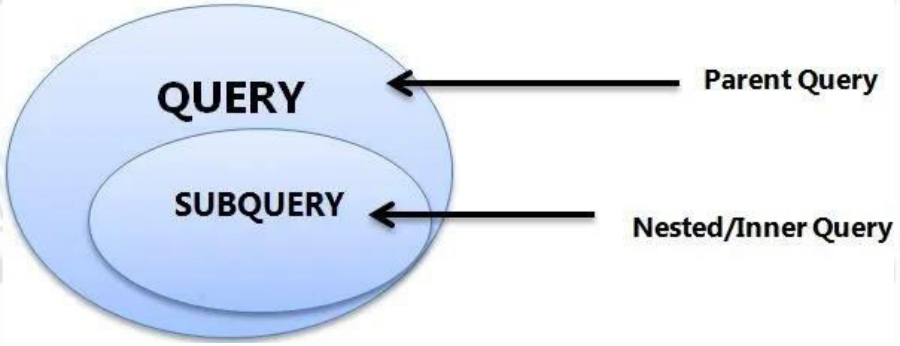
Es fundamental que el select devuelva las mismas filas y en el mismo orden las columnas que la tabla destino.



CePETel

DML - SQL

SQL – Subquery en DELETE




```
DELETE FROM customer
        WHERE customer_num NOT IN
        (SELECT DISTINCT customer_num FROM cust_calls)
        AND customer_num NOT IN
        (SELECT DISTINCT customer_num FROM orders)
        AND customer_num NOT IN
        (SELECT DISTINCT customer_num_referedBy FROM customer c2
          WHERE customer_num_referedBy IS NOT NULL)
```

1

2

3

Utilizando tres subqueries como condiciones del DELETE, podríamos borrar todos los clientes de la Tabla customer, que no posean órdenes de compra asociadas y que no hayan referenciado a otro cliente y que no tengan llamados telefónicos.

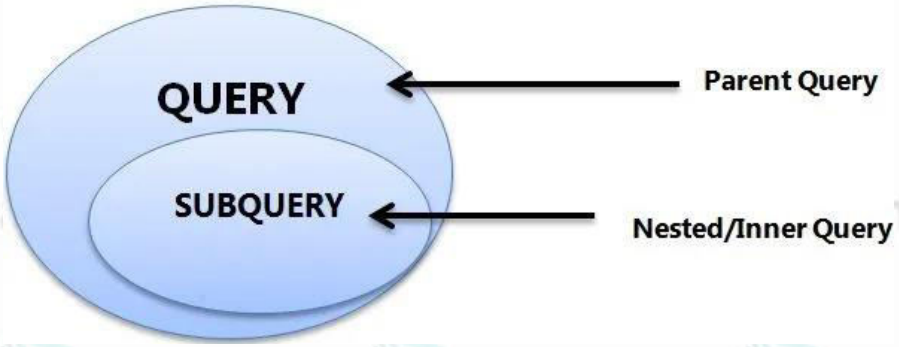
DML - SQL

SQL – Subquery en DELETE

```
64  CREATE TEMP TABLE clientesParaBorrar AS SELECT * FROM customer;
65
66  DELETE FROM clientesParaBorrar
67      WHERE customer_num NOT IN
68        (SELECT DISTINCT customer_num FROM cust_calls)
69        AND customer_num NOT IN
70        (SELECT DISTINCT customer_num FROM orders)
71        AND customer_num NOT IN
72        (SELECT DISTINCT customer_num_referedBy FROM customer c2
73         WHERE customer_num_referedBy IS NOT NULL)
74
```

Data Output | Explain | Messages | Notifications

```
DELETE 8

Query returned successfully in 847 msec.
```



Para hacer la prueba y no borrar datos de nuestra tabla real corrimos el ejemplo en la tabla temporal creada clientesParaBorrar.

CePETel

DML - SQL

SQL – Subquery en DELETE

```
76  SELECT customer_num FROM customer
77  EXCEPT
78  SELECT customer_num FROM clientesParaBorrar
79  ORDER BY 1
```

VEREMOS ESTE OPERADOR LUEGO

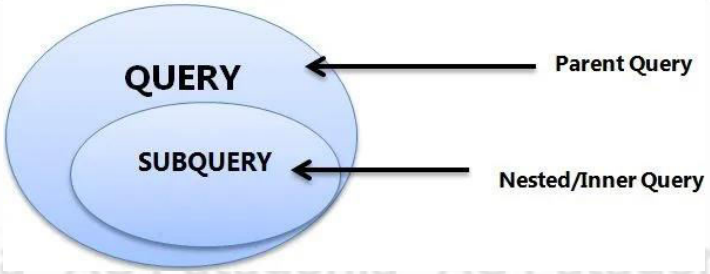
Data Output | Explain | Messages | Notifications

	customer_num smallint
1	102
2	105
3	108
4	109
5	113
6	114
7	118
8	128

En este ejemplo hacemos la comparación entre las tablas para ver qué filas se borraron.

DML - SQL

SQL – Subquery en UPDATE




```
UPDATE FROM customer
   SET columna =
    WHERE columna (=/IN/NOT IN)

          (EXISTS / NOT EXISTS )
```

Subquery

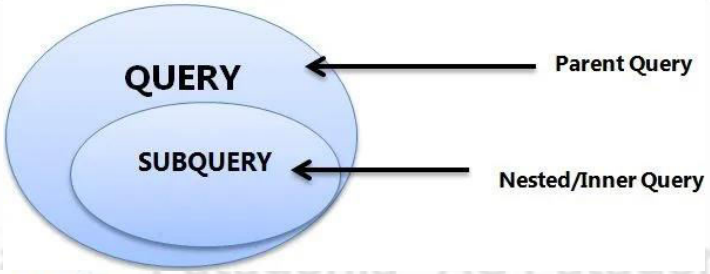
Subquery

Subquery

En un update vemos distintos lugares y formas de ejecutar un subquery.

DML - SQL

SQL – Subquery en CLAUSULA SET




```
1  ALTER TABLE state RENAME COLUMN code TO state; -- Corregido para que quede igual que el modelo pasado.
2
3  CREATE TEMP TABLE clientesParaBorrar AS SELECT * FROM customer;
4
5  UPDATE clientesParaBorrar
6  SET state=(SELECT state from state WHERE sname= 'Florida')
7  WHERE customer_num=101;
8
9  SELECT cpb.lname, cpb.fname, cpb.state NuevoState, c.state OrigState
10 FROM clientesParaBorrar cpb JOIN customer c ON (cpb.customer_num = c.customer_num)
11 WHERE cpb.state <> c.state
```

Data Output | Explain | Messages | Notifications

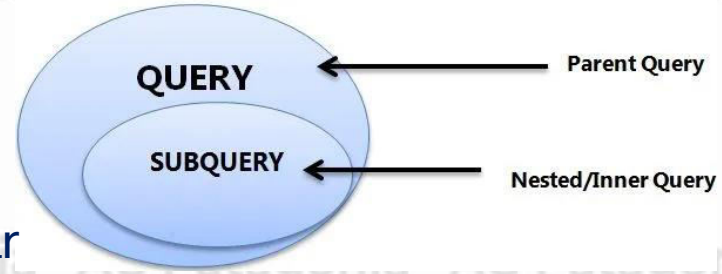
	lname character varying (15)	fname character varying (15)	nuevostate character (2)	origstate character (2)
1	Pauli	Ludwig	FL	CA

El subquery puede retornar solo una única columna y fila. O sea, un valor escalar, debido a que está combinado con un operador de igualdad (=).

CePETel

DML - SQL

SQL – Subquery en UPDATE



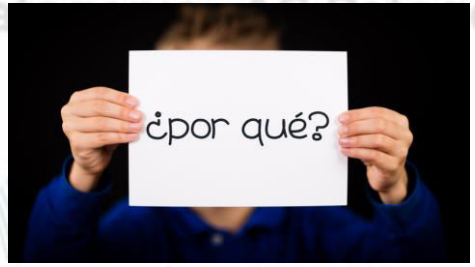
Subquery en CLAUSULA WHERE con Subquery que devuelve valor escalar

```
13  UPDATE clientesParaBorrar
14  SET state='CA'
15  WHERE customer_num = (SELECT customer_num from customer WHERE lname= 'Pauli');
16
17  SELECT lname, fname, state
18  FROM  customer c
19  WHERE customer_num = 101
20
```

Subquery

Data Output | Explain | Messages | Notifications

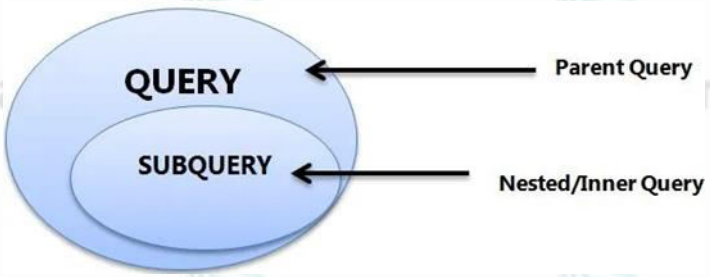
	lname character varying (15)	fname character varying (15)	state character (2)
1	Pauli	Ludwig	CA



El subquery puede retornar solo una única columna y fila. O sea, un valor escalar.

CePETel

DML - SQL



SQL – Subquery en UPDATE

Subquery en CLAUSULA WHERE con Subquery que devuelve una columna con múltiples filas.

```
22  UPDATE manufact SET lead_time=15
23  WHERE manu_code IN (SELECT DISTINCT manu_code FROM items);
24
25
26
```

Subquery

Data Output | Explain | Messages | Notifications

```
UPDATE 9

Query returned successfully in 660 msec.
```

```
26  SELECT DISTINCT i.manu_code, lead_time
27  FROM items i JOIN manufact m ON (i.manu_code=m.manu_code)
```

Data Output | Explain | Messages | Notifications

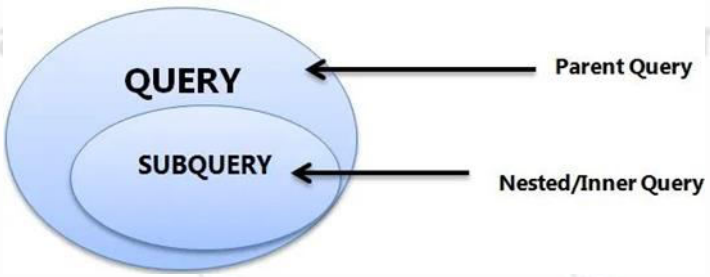
	manu_code character (3)	lead_time smallint
1	ANZ	15
2	KAR	15
3	SMT	15
4	HSK	15
5	PRC	15
6	HRO	15
7	SHM	15
8	NKL	15
9	NRG	15

Comprobación

Modificamos el lead_time (tiempo de entrega) a 15 días para todos los proveedores de los cuales ya hayamos entregado productos.

DML - SQL

SQL – Subquery en UPDATE



¿Qué sucede si corremos este UPDATE?

```
UPDATE manufact SET lead_time=15
WHERE manu_code = (SELECT DISTINCT
manu_code
FROM items)
```

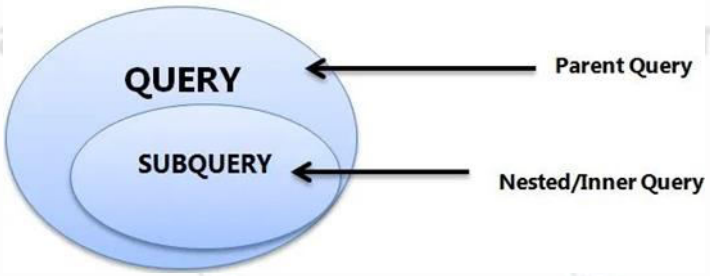
```
 99  UPDATE manufact SET lead_time=15
100  WHERE manu_code = (SELECT DISTINCT manu_code FROM items)
```

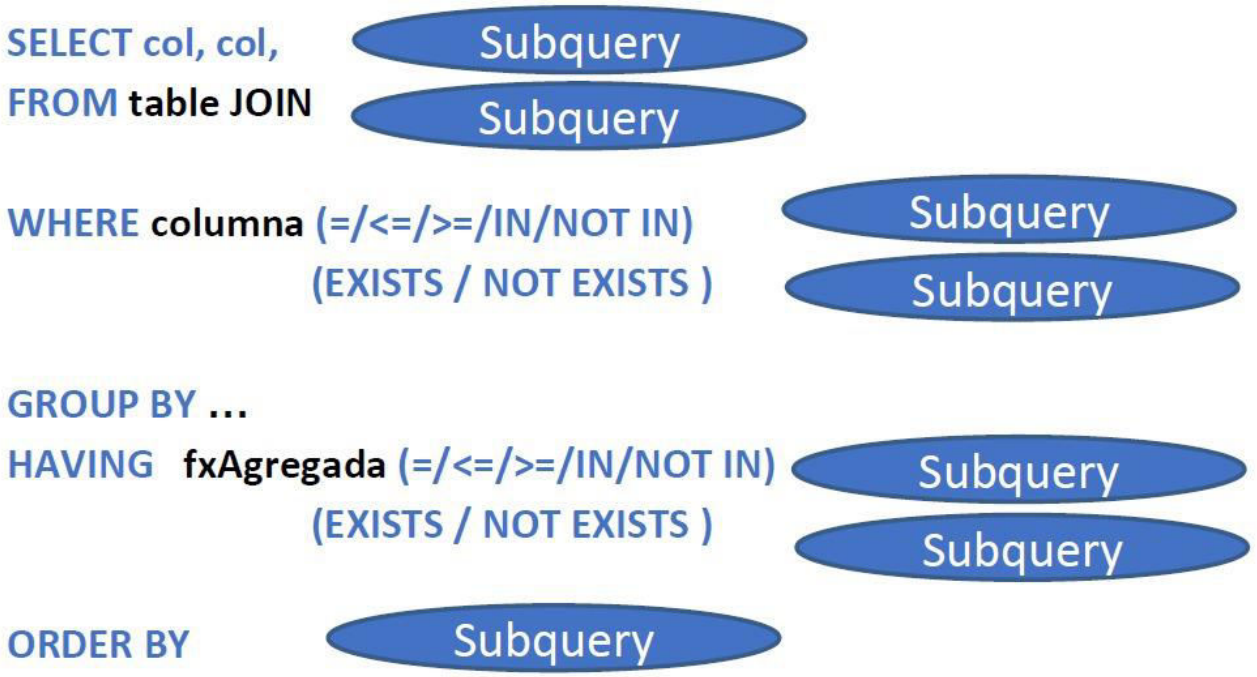
Data Output | Explain | Messages | Notifications

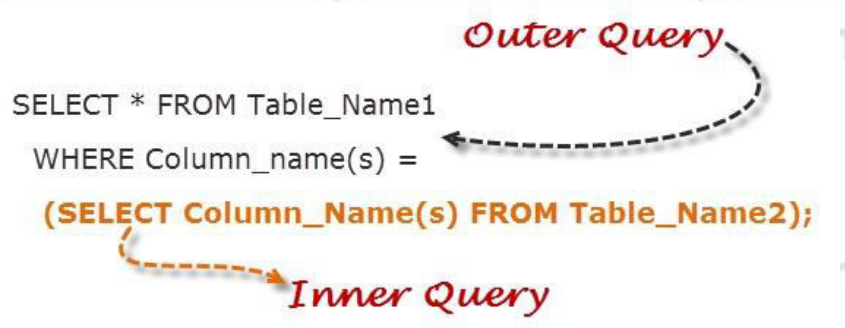
```
ERROR:  more than one row returned by a subquery used as an expression
SQL state: 21000
```

DML - SQL

SQL – Subquery en SELECT







En un SELECT vemos distintos lugares y formas de ejecutar un subquery.

CePETel

DML - SQL

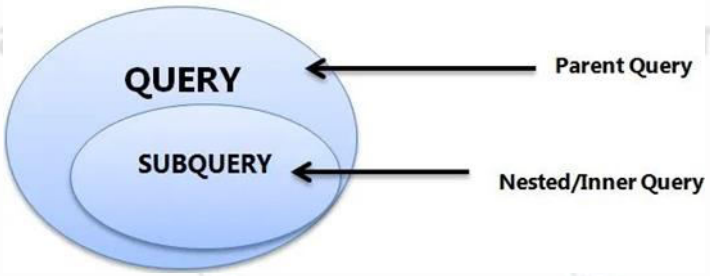
SQL – Subquery en SELECT

Subquery en Lista de columnas

```
110  SELECT customer_num, COUNT(order_num) cantOCCliente,
111         (SELECT count(*) FROM orders) CantTotalOC
112  FROM orders
113  GROUP BY customer_num
114
```

Data Output | Explain | Messages | Notifications

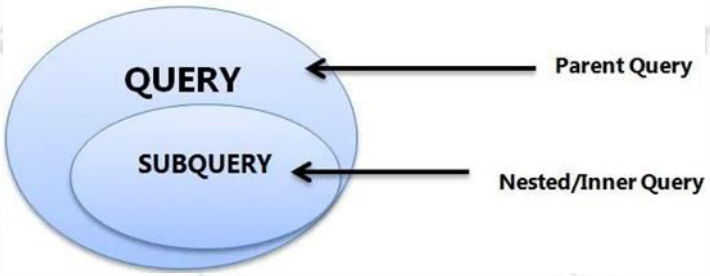
	customer_num smallint	cantoccliente bigint	canttotaloc bigint
1	101	1	23
2	116	1	23
3	122	1	23
4	121	1	23
5	117	2	23
6	115	1	23
7	119	1	23
8	112	1	23
9	106	2	23
10	104	4	23



El subquery devuelve la cantidad total de órdenes que existen y como si fuese un valor constante lo muestra en cada fila

DML - SQL

SQL – Subquery en SELECT



Subquery en FROM

```
115  SELECT lname apellido, fname nombre, cliente, cantidad
116  FROM customer c1 JOIN
117                  (SELECT customer_num cliente, count(order_num) cantidad
118                   FROM orders GROUP BY customer_num) c2
119                   ON (c1.customer_num = c2.cliente)
120  WHERE cantidad>3
```

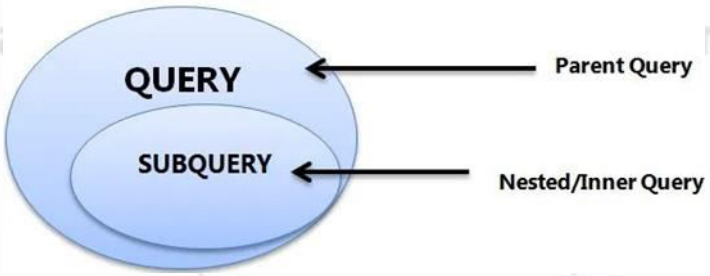
Data Output | Explain | Messages | Notifications

	apellido character varying (15)	nombre character varying (15)	cliente smallint	cantidad bigint
1	Higgins	Anthony	104	4

El subquery devuelve un conjunto de filas se asocian al alias c2 como si fuese el result set de una tabla y luego estos datos son JOINEADOS con la tabla customer c1.

DML - SQL

SQL – Subquery en SELECT



Subquery en WHERE

```
101  SELECT CONCAT (lname,', ',fname) customer_num
102  FROM customer
103  WHERE customer_num IN (SELECT customer_num FROM cust_calls
104                         GROUP BY customer_num HAVING COUNT(*)>1)
105
```

Data Output | Explain | Messages | Notifications

	customer_num text
1	Parmelee, Jean

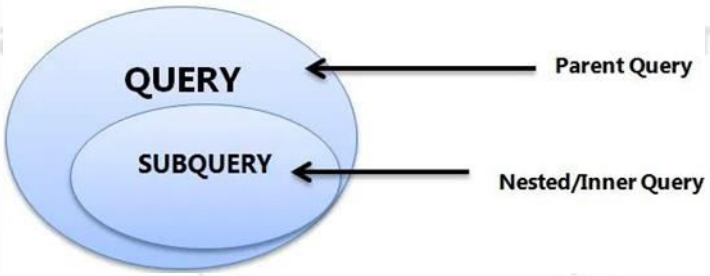
La consulta muestra los clientes que posean más de una llamada.
Muchas veces este tipo de subqueries se pueden resolver mediante joins.

Anímate e inténtalo

CePETel

DML - SQL

SQL – Subquery en SELECT



Subquery en WHERE con un valor escalar

```
106  SELECT COUNT(*) FROM customer
107  WHERE city = (SELECT city FROM customer WHERE lname='Higgins')
108
```

Data Output | Explain | Messages | Notifications

	count bigint
1	5

La consulta devuelve la cantidad de clientes que viven en la misma ciudad en la que vive Higgins, incluso el mismo.

Muchas veces este tipo de subqueries se pueden resolver mediante joins.

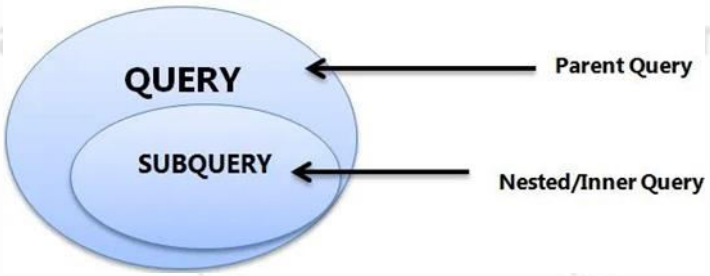
Anímate e inténtalo

CePETel

DML - SQL

SQL – Subquery en SELECT

Subquery CORRELACIONADO




```
86  SELECT customer_num, fname
87  FROM customer c
88  WHERE NOT EXISTS (SELECT order_num FROM orders o
89                    WHERE o.customer_num = c.customer_num)
```

Data Output | Explain | Messages | Notifications

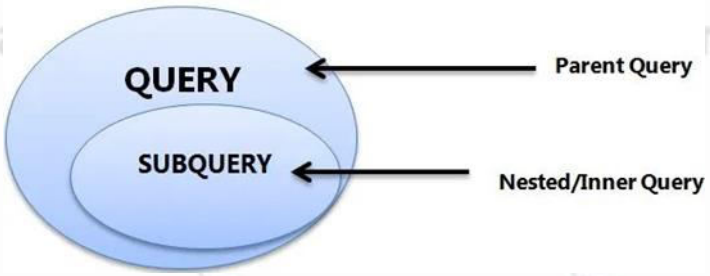
	customer_num [PK] smallint	fname character varying (15)
1	102	Carole
2	103	Philip
3	105	Raymond
4	107	Charles
5	108	Donald
6	109	Jane
7	113	Lana
8	114	Frank
9	118	Dick
10	125	James
11	128	Frank

La consulta devuelve los clientes que para los que no existen órdenes de compra.
Muchas veces es necesario utilizarlo en subqueries en el WHERE de comandos UPDATE o DELETE.

Existe una correlación entre la ejecución del subquery y la fila en la que se está en el Query original.
En general no es muy performante!!

DML - SQL

SQL – Subquery en SELECT



Listar las órdenes de los clientes con su total comprado, pero sólo para los que el total de la orden sea mayor que el promedio comprado de dicho cliente.

```
77  SELECT o.order_num, o.customer_num, SUM(quantity*unit_price) totalOrden, promedio
78  FROM orders o JOIN items i ON o.order_num=i.order_num
79  JOIN (SELECT customer_num,  ROUND(SUM(quantity*unit_price)/COUNT(DISTINCT o2.order_num),2) promedio
80        FROM orders o2 JOIN items i2 ON (o2.order_num=i2.order_num)
81        GROUP BY customer_num) tabSQ  ON (o.customer_num= tabSQ.customer_num)
82  GROUP BY o.order_num, o.customer_num, tabSQ.promedio
83  HAVING SUM(i.quantity*unit_price) > tabSQ.promedio
84
```

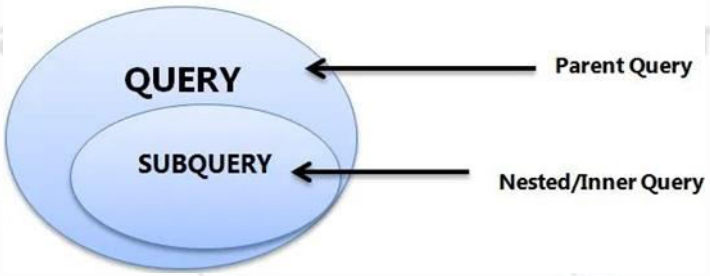
Resolución con subquery en FROM

Data Output | Explain | Messages | Notifications

	order_num [PK] smallint	customer_num smallint	totalorden numeric	promedio numeric
1	1008	110	1340.00	895.00
2	1003	104	1355.00	570.95
3	1014	106	1440.00	1428.00
4	1012	117	2840.00	2268.00

DML - SQL

SQL – Subquery en SELECT




```
SELECT stock_num,manu_code
FROM products
WHERE unit_price < 25
UNION
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
ORDER BY 1,2
```

La cantidad de campos en cada consulta debe corresponderse.

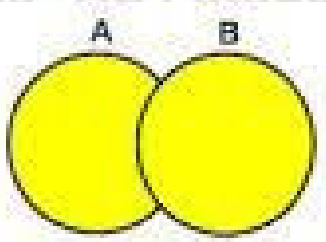
Los campos en igual posición deben tener el mismo tipo de datos.

La cláusula ORDER BY va al final del último SELECT.

Solo se puede Ordenar referenciando por posición. En el caso de filas con idénticos valores solo deja una de las dos.

DML - SQL

SQL – Operador UNION




```
SELECT stock_num,manu_code
FROM products
WHERE unit_price < 25
```

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

```
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
```

	stock_num	manu_code
1	5	NRG
2	5	SMT
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	NRG
7	5	ANZ
8	5	ANZ
9	5	SMT

```
68  SELECT stock_num,manu_code
69  FROM products WHERE unit_price < 25
70  UNION
71  SELECT stock_num,manu_code
72  FROM items WHERE stock_num=5
73  ORDER BY 1,2
```

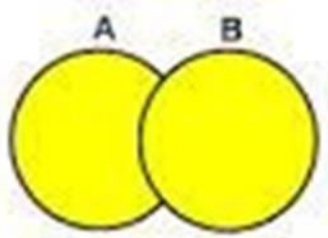
Data Output | Explain | Messages | Notifications

	stock_num smallint	manu_code character (3)
1	5	ANZ
2	5	NRG
3	5	SMT
4	9	ANZ
5	103	PRC
6	106	PRC
7	302	HRO
8	302	KAR

CePETel

DML - SQL

SQL – Operador UNION




```
SELECT stock_num PROD,manu_code FAB FROM products
WHERE unit_price < 25
UNION
SELECT stock_num,manu_code FROM items WHERE stock_num=5
ORDER BY 1,2
```

```
SELECT stock_num PROD,manu_code FAB FROM products
WHERE unit_price < 25
UNION
SELECT stock_num,manu_code FROM items WHERE stock_num=5
ORDER BY 1,2
```

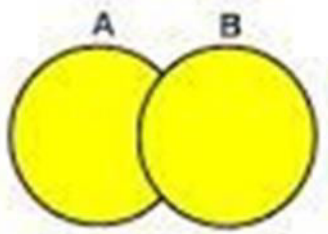
Results | Messages

PROD	FAB
5	ANZ
5	NRG
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

La tabla de salida toma los nombres de columnas del primer SELECT.

DML - SQL

SQL – Operador UNION




```
SELECT 1 orden,customer_num,order_num, order_date
FROM orders WHERE customer_num=127
UNION
SELECT 3 orden,customer_num,order_num, order_date
FROM orders WHERE customer_num BETWEEN 111 AND 126
UNION
SELECT 2 orden,customer_num,order_num, order_date
FROM orders WHERE customer_num BETWEEN 1 AND 110
ORDER BY 1 ASC, 2 DESC
```

En este caso particular vemos que utilizamos un ordenamiento de filas determinado por una constante en cada SELECT que dice que filas irán primero en la salida.

DML - SQL

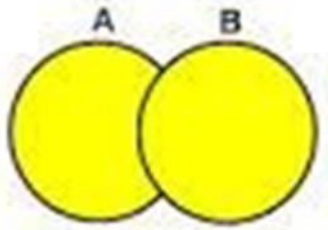
SQL – Operador UNION

Ordenamiento particular:
primero el 127, luego del 110 al 1, y finalmente del 128 al 111

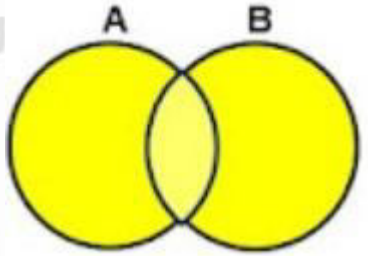
```
57  SELECT 1 orden,customer_num,order_num, order_date
58  FROM orders WHERE customer_num=127
59  UNION
60  SELECT 3 orden,customer_num,order_num, order_date
61  FROM orders WHERE customer_num BETWEEN 111 AND 126
62  UNION
63  SELECT 2 orden,customer_num,order_num, order_date
64  FROM orders WHERE customer_num BETWEEN 1 AND 110
65  ORDER BY 1 ASC, 2 DESC
```

Data Output | Explain | Messages | Notifications

	orden integer	customer_num smallint	order_num smallint	order_date date
1	1	127	1023	2015-07-20
2	2	110	1008	2015-06-03
3	2	110	1015	2015-06-23
4	2	106	1004	2015-05-18
5	2	106	1014	2015-06-21
6	2	104	1013	2015-06-18
7	2	104	1003	2015-05-18
8	2	104	1011	2015-06-14
9	2	104	1001	2015-05-16
10	2	101	1002	2015-05-17
11	3	126	1022	2015-07-20
12	3	124	1021	2015-07-19
13	3	123	1020	2015-07-07
14	3	122	1019	2015-07-07
15	3	121	1018	2015-07-06
16	3	120	1017	2015-07-05
17	3	119	1016	2015-06-25
18	3	117	1007	2015-05-27
19	3	117	1012	2015-06-14
20	3	116	1005	2015-05-20



DML - SQL



SQL – Operador UNION ALL

```
SELECT stock_num,manu_code FROM products WHERE unit_price < 25
UNION ALL
SELECT stock_num,manu_code FROM items WHERE stock_num=5
ORDER BY 1,2
```

```
51  SELECT stock_num,manu_code FROM products WHERE unit_price < 25
52  UNION ALL
53  SELECT stock_num,manu_code FROM items WHERE stock_num=5
54  ORDER BY 1,2
55
```

Data Output | Explain | Messages | Notifications

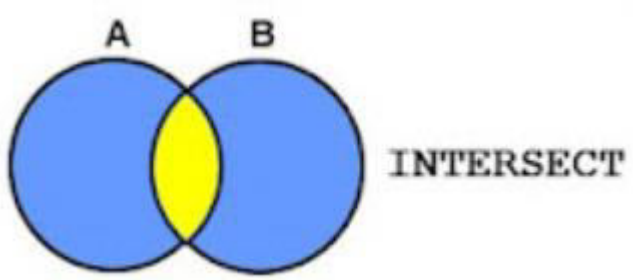
	stock_num smallint	manu_code character (3)
1	5	ANZ
2	5	ANZ
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	ANZ
7	5	NRG
8	5	NRG
9	5	SMT
10	5	SMT
11	9	ANZ
12	103	PRC
13	106	PRC
14	302	HRO
15	302	KAR

El Operador UNION con la cláusula ALL, repite en la estructura de salida todas las filas de las tablas involucradas, sean o no iguales.

CePETel

DML - SQL

SQL – Operador INTERSECT




```
SELECT stock_num,manu_code
FROM products
WHERE unit_price < 25
INTERSECT
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
ORDER BY 1,2
```

Mismas restricciones del UNION.
Devuelve las filas que están en ambas consultas.

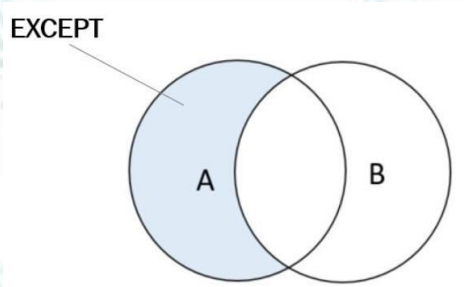
```
41  SELECT stock_num,manu_code
42  FROM products
43  WHERE unit_price < 25
44  INTERSECT
45  SELECT stock_num,manu_code
46  FROM items
47  WHERE stock_num=5
48  ORDER BY 1,2
49
```

Data Output | Explain | Messages | Notifications

	stock_num smallint	manu_code character (3)
1	5	ANZ

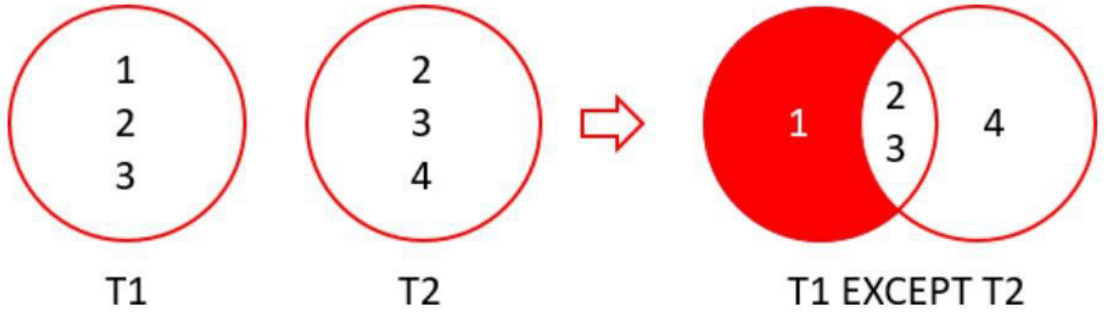
DML - SQL

SQL – Operador EXCEPT




```
SELECT stock_num,manu_code
FROM products
WHERE unit_price < 25
EXCEPT
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
ORDER BY 1,2
```

Mismas restricciones del UNION.
Devuelve las filas del Primer SELECT que no están en la intersección con el Segundo SELECT.




```
31  SELECT stock_num,manu_code
32  FROM products
33  WHERE unit_price < 25
34  EXCEPT
35  SELECT stock_num,manu_code
36  FROM items
37  WHERE stock_num=5
38  ORDER BY 1,2
```

Data Output | Explain | Messages | Not

	stock_num smallint	manu_code character (3)
1	9	ANZ
2	103	PRC
3	106	PRC
4	302	HRO
5	302	KAR

CePETel

DML - SQL

SQL – Comparación U - Ua - I - E

	stock_num	manu_code
1	5	ANZ
2	9	ANZ
3	103	PRC
4	106	PRC
5	302	HRO
6	302	KAR

```
SELECT
stock_num,manu_code
FROM products
WHERE unit_price < 25
```

```
SELECT stock_num,manu_code
FROM items
WHERE stock_num=5
```

	stock_num	manu_code
1	5	NRG
2	5	SMT
3	5	ANZ
4	5	ANZ
5	5	ANZ
6	5	NRG
7	5	ANZ
8	5	ANZ
9	5	SMT

UNION

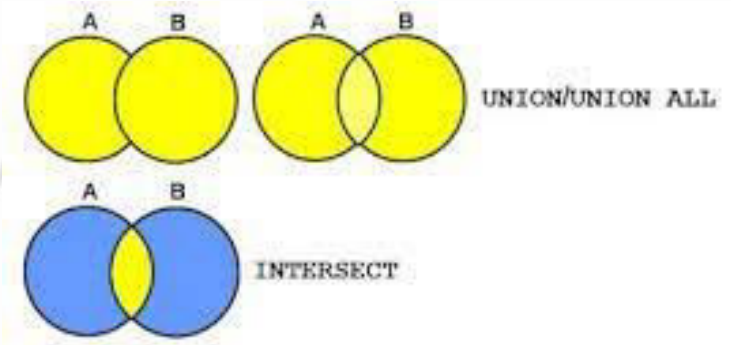
stock_num	manu_code
5	ANZ
5	NRG
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

UNION

stock_num	manu_code
5	ANZ
5	ANZ
5	ANZ
5	ANZ
5	ANZ
5	ANZ
5	NRG
5	NRG
5	SMT
5	SMT
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

ALL INTERSECT

stock_num	manu_code
5	ANZ

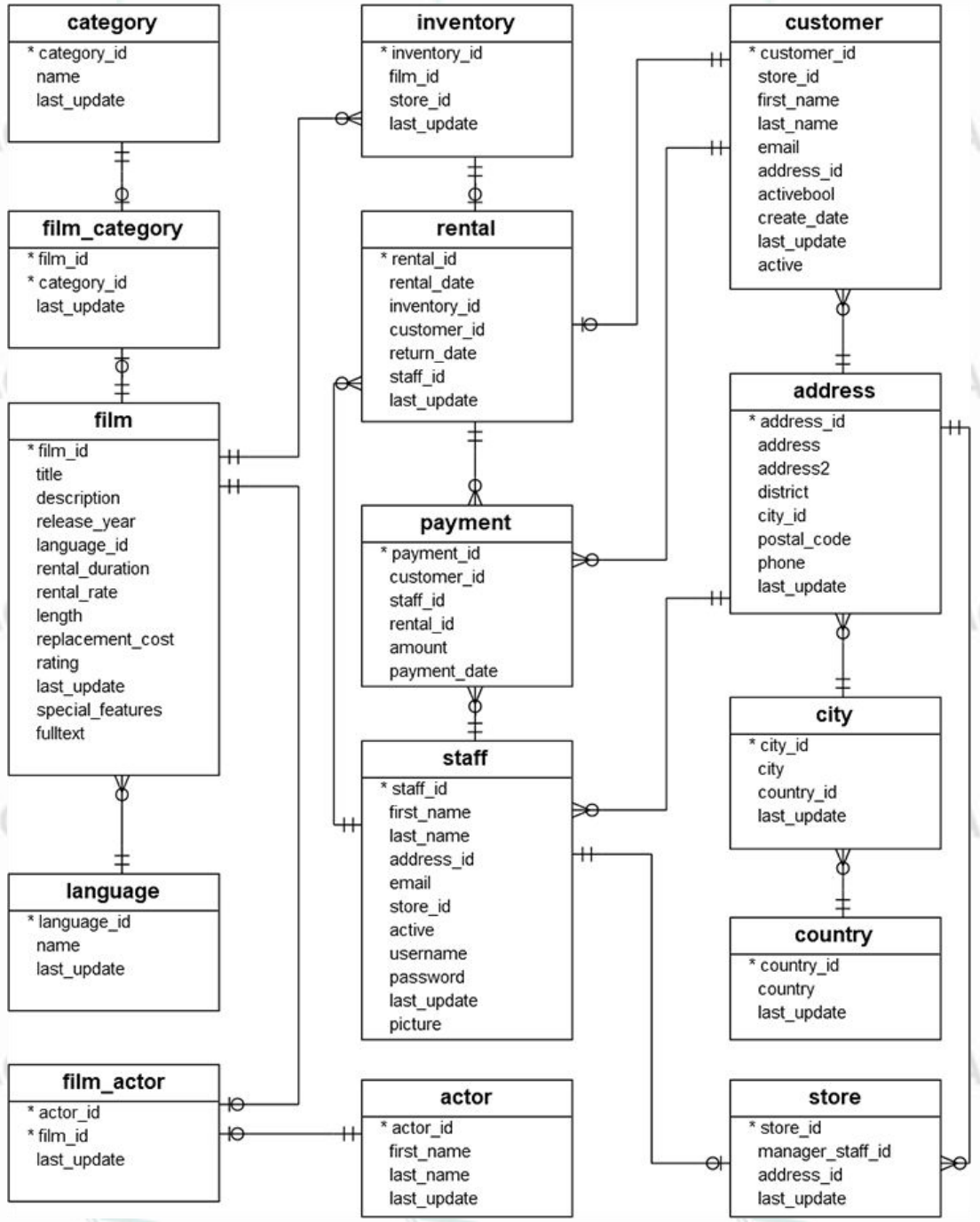


EXCEPT

stock_num	manu_code
9	ANZ
103	PRC
106	PRC
302	HRO
302	KAR

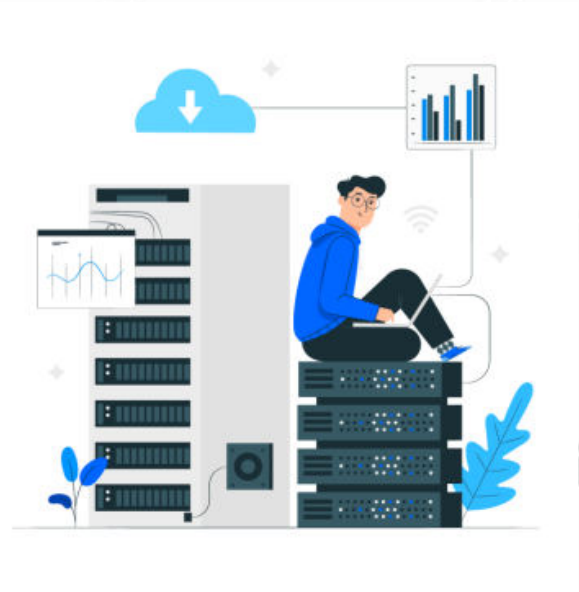
CePETel

TP N° 7: Joins, Subconsultas y Condicionales



CePETel

TP N° 7: Joins, Subconsultas y Condicionales



1) Ejercicios: Inner Joins

Selecciona first_name, last_name, amount, y payment_date uniéndo las tablas de clientes y pagos (customer y payment).

Seleccione film_id, Category_id y name de la unión de las tablas film_category y category, solo cuando Category_id sea menor que 10.

2) Ejercicio: Joining y agrupación

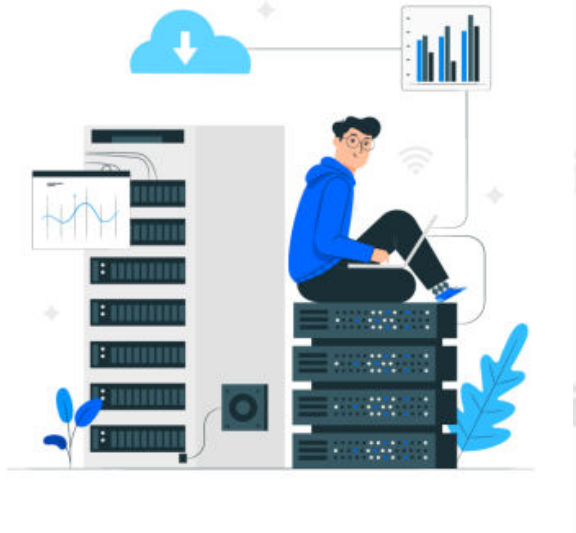
Obtenga una lista de los nombres de los clientes que han gastado más de $150, junto con su gasto total. (incluir nombre y apellido del cliente)

¿Quién es el cliente con el monto de pago promedio más alto? (incluir nombre y apellido del cliente)

3) Ejercicio: Joining las tablas Customers, Payments y Staff

Unir las tablas de clientes y de pago con un inner join; seleccione customer_id, el nombre, el monto y la fecha y ordene por customer_id. Luego, haga un join también a la staff table para agregar el nombre del personal (staff name)

TP N° 7: Joins, Subconsultas y Condicionales



4) Joining para obtener mejores direcciones

Crea una lista de direcciones que incluya el nombre de la ciudad en lugar del ID number y también el nombre del país.

5) Ejercicio Left Join

Tenemos nuestra tabla de películas que enumera todas las películas, nuestra tabla de inventario, y necesitábamos saber qué películas no están en el inventario.

5) Ejercicio: Subqueries

¿En qué películas aparecen juntos los actores con identificadores 129 y 195?

Desafío: ¿Cuántos actores hay en más películas que el actor id 47? Sugerencia: esto requiere 2 subconsultas (una anidada en la otra). Trabaja de adentro hacia afuera: 1) en cuántas películas participa el actor 47; 2) ¿Qué actores aparecen en más películas que esta? 3) Cuente esos actores.

CePETel