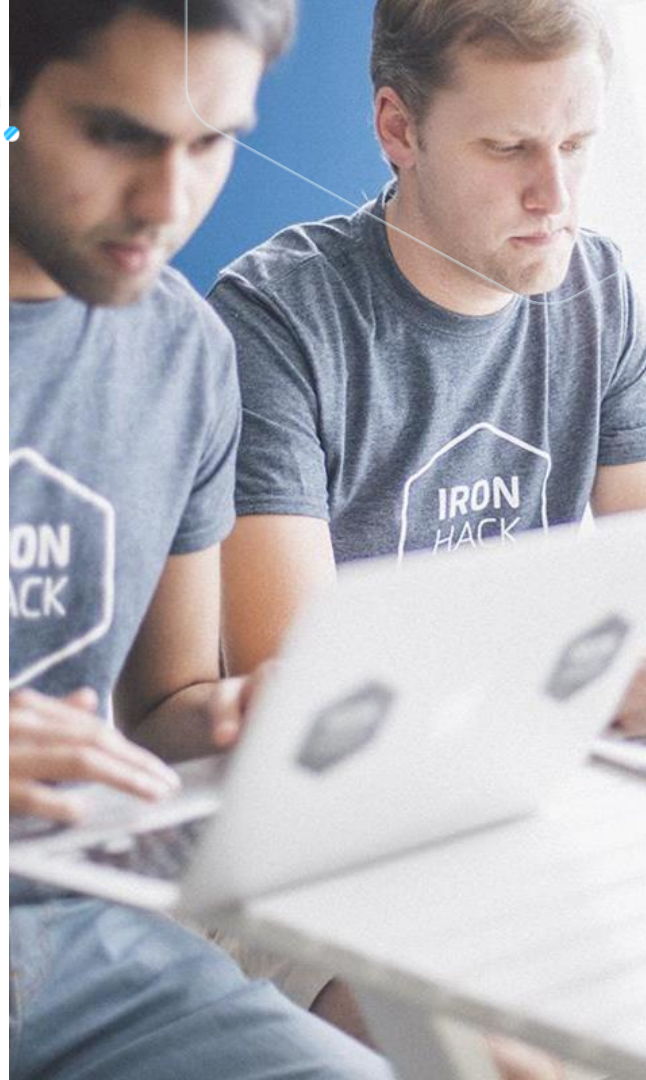
Intro to SQL

Hear, forget. See, remember.

Do, Understand

/ Objetivos

- Aprender conceptos fundamentales de base de datos como entidades y relaciones.
- Entender dónde situar este lenguaje en el panorama actual.
- Conocer su sintaxis básica, así como funciones más avanzadas.
- Familiarizarse con la forma de trabajo en proyectos reales.



Contexto

SQL | Contexto

- Todas las empresas quieren ser hoy en día **data-driven.**
- Y a lo largo de los años han ido creando **bases de datos** para poder operar en su día a día, así como para analizar históricos.
- Esos sistemas, muy diversos entre sí, comparten sin embargo una forma de trabajar: **Structured Query Language (SQL)**.
- Aunque tiene connotaciones negativas (antiguo, *verboso*, limitado, etc.) al final siempre termina apareciendo en los lugares más insospechados.
- Razones? (ahí va mi apuesta):
    - Facilidad de aprendizaje.
    - Ser declarativo.
    - Ubícuo.

SQL | Modelado

- Las bases de datos se modelan a partir de **entidades** (pensadlos como nombres: personas, lugares, cosas o eventos)
    - En un dataset de libros, nos encontraremos entidades como *libros, autores, editores*...
    - En un dataset de deportes, tendremos *jugadores, posiciones, equipos, partidos*...

- Estas entidades tienen conexión con otras entidades a través de distintos tipos de **relaciones**
    - **1:1**, cuando una entidad puede tener o pertenecer a otra entidad únicamente (*1 libro* es publicado por *1 editorial*)
    - **1:M**, cuando una entidad puede tener o pertenecer a múltiples entidades (*1 equipo* se compone de *muchos jugadores*)
    - **M:M**, cuando una entidad puede tener o pertenecer a múltiples entidades, y viceversa (*1 libro* es creado por *múltiples autores* y *1 autor* puede crear *múltiples libros*)

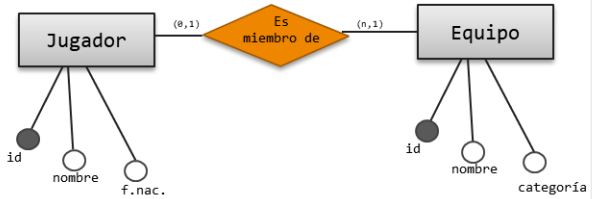
SQL | Normalización

- La información en una base de datos relacional se organiza de una manera *normalizada*.
- Existen seis reglas (**formas normales**) que permiten evitar las anomalías e inconsistencias de un modelo, que se pueden resumir en:
    - Cada entidad debe estar representada como una **tabla** donde cada representación individual (registro) debe tener un identificador único (**clave primaria**)
    - Las relaciones entre entidades se mantienen mediante **claves foráneas**, donde un campo de una tabla hace referencia al identificador de la tabla relacionada.
- El objetivo de este diseño normalizado es evitar la **redundancia** y garantizar la **coherencia** de los datos.
- Pero... hay más formas de modelar una base de datos (*dimensional modeling*, *data vault*, *anchor modeling*, etc.) cuando nuestros objetivos son otros (**analytics**).

SQL | Diseño

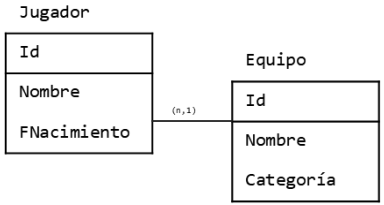
- La forma de diseñar las bases de datos es a través de **diagramas Entidad Relación** (E/R).
- A través de ellos podemos ver las entidades de las que se compone nuestro modelo, así como las relaciones existentes entre ellas.

Modelo Conceptual



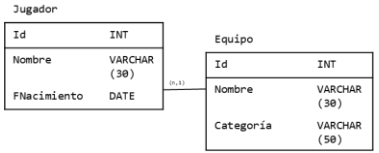
- Descripción a alto nivel para que técnico y usuario lleguen a un acuerdo.
- Solo se incluye la realidad que tenga sentido almacenar en la base de datos.

Modelo Lógico



- Diseño teórico de las tablas, sin tener en consideración criterios específicos del motor de base de datos

Modelo Físico

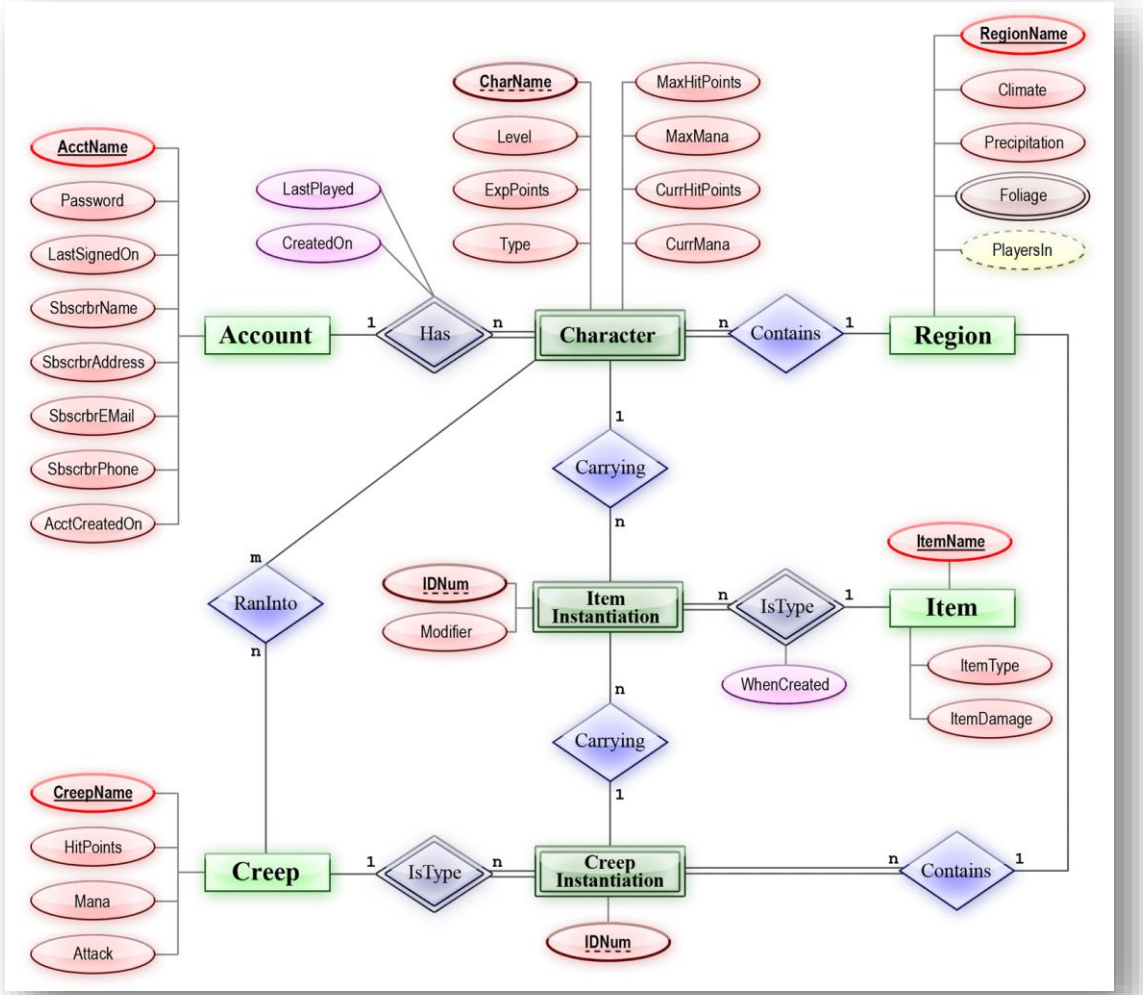


- Diseño final de las tablas considerando ya el motor donde se implementarán
- Se tienen en cuenta consideraciones de rendimiento

SQL | Diseño

Modelo Conceptual

- Descripción a alto nivel para que técnico y usuario lleguen a un acuerdo.
- Solo se incluye la realidad que tenga sentido almacenar en la base de datos.

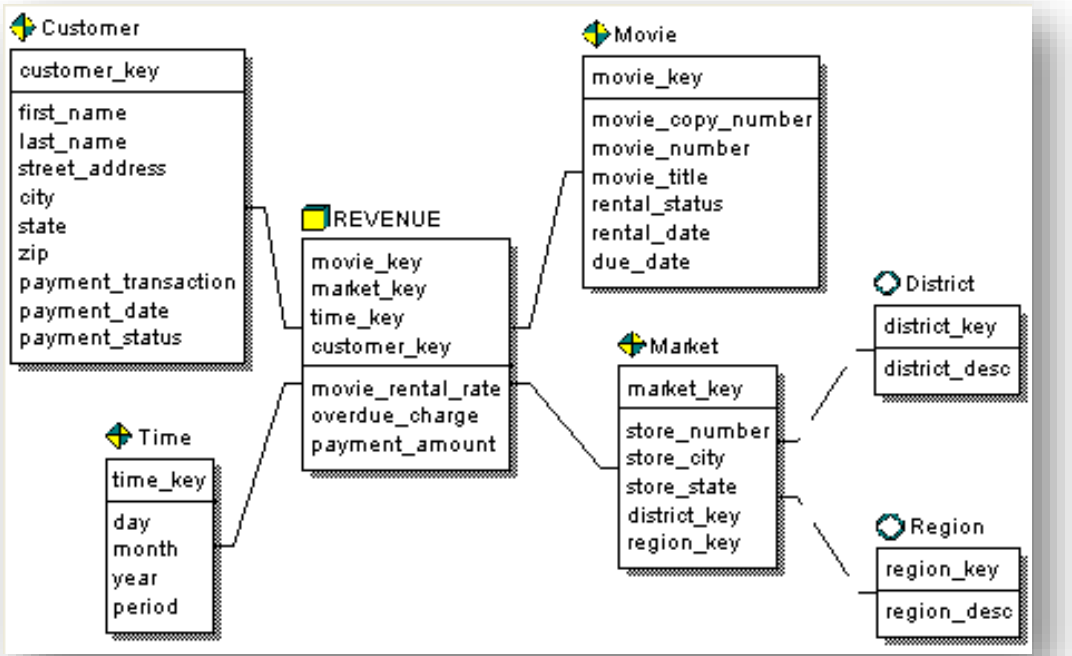


Fuente: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model

SQL | Diseño

Modelo Lógico

- Diseño teórico de las tablas, sin tener en consideración criterios específicos del motor de base de datos

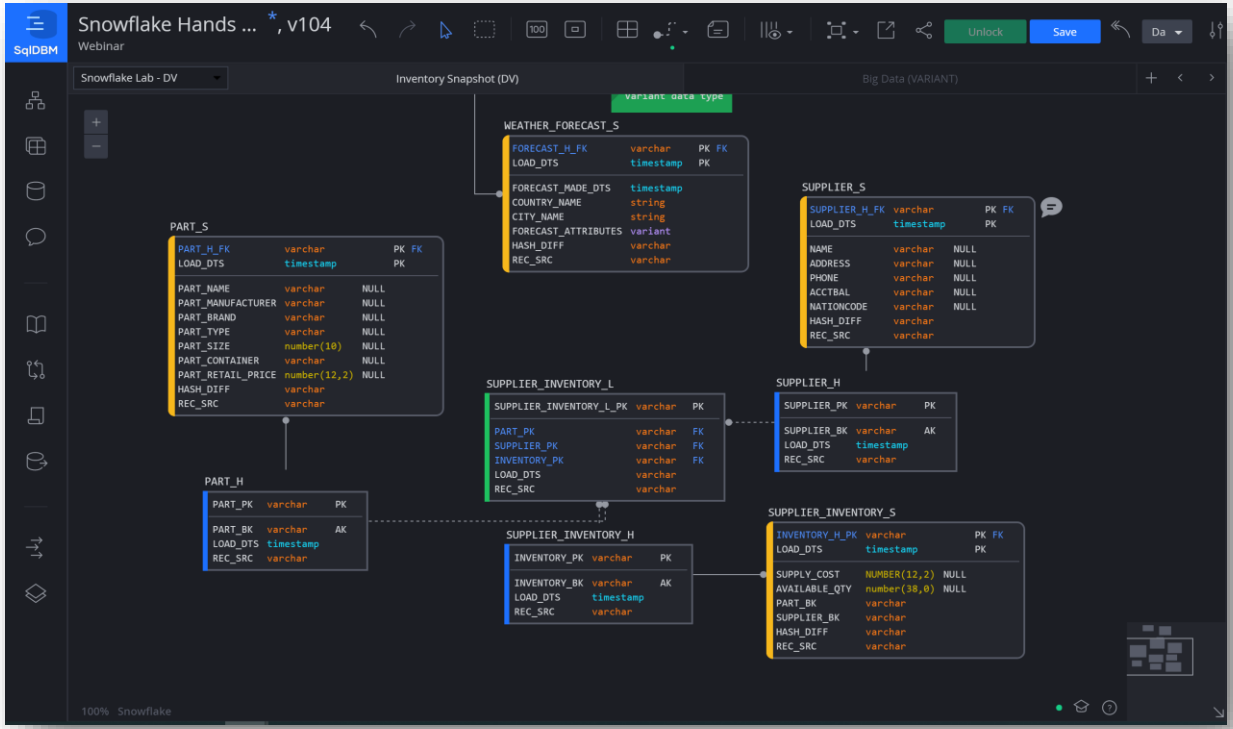


Fuente:
https://bookshelf.erwin.com/bookshelf/9.8.00/Bookshelf_Files/HTML/erwin%20Help/index.htm?toc.htm?Star_Schema_Design.html

SQL | Diseño

Modelo Físico

- Diseño final de las tablas considerando ya el motor donde se implementarán
- Se tienen en cuenta consideraciones de rendimiento



Fuente: https://galavan.com/data-modeling-sqldbm-the-snowflake-data-cloud-the-lowdown/

SQL | Summary

#1: Hemos retrocedido un poco para entender el lugar que ocupa SQL hoy en día.

#2: Para pasar a comentar brevemente los conceptos fundamentales del diseño de base de datos.

#3: Así que ya estamos preparados para empezar a jugar.

Lenguaje SQL

SQL

- ***S**tructured **Q**uery **L**anguage*.
- Diseñado, principalmente, para SGBDR (***S**istemas **G**estores de **B**ase de **D**atos **R**elacionales*).
- Aunque es un lenguaje estándar, existen implementaciones **particulares** (según el motor).
- Lenguaje **declarativo**: se dice el *qué*, no el *cómo*.
- El lenguaje no es sensible a mayúsculas/minúsculas, pero los objetos creados sí pueden serlo.
- No es sólo para consultas
  - DDL – Data **Definition** Language (`CREATE`, `ALTER`, etc.)
  - DML – Data **Manipulation** Language (`UPDATE`, `INSERT`, etc.)
  - DCL – Data **Control** Language (`GRANT`, `REVOKE`, etc.)
  - DQL – Data **Query** Language (`SELECT`)

Credit: https://commons.wikimedia.org/wiki/File:Sql_data_base_with_logo.png

SQL | DDL

- Nos permite **crear, modificar y eliminar** objetos dentro de la base de datos
- Los nombres deben ser **únicos** dentro del nivel en el que se encuentran
  - Bases de datos dentro del servidor.
  - Tablas / vistas dentro de la base de datos.
  - Columnas dentro de la tabla / vista

CREATE

Define un nuevo objeto (tabla, índice, vista, etc.)

```
CREATE TABLE { tabla } (
[ opciones creación de tabla ]
[ definiciones de columnas]
[ restricciones a nivel de tabla ]
[ definiciones de índices ]
) [ ; ]
```

```
CREATE TABLE dbo.myTable
(
    ID INT NOT NULL REFERENCES otherTable.ID(ID),
    colB SMALINT NOT NULL,
    colC BIGINT NOT NULL,
    ColD FLOAT NULL,
    ColE DATETIME NOT NULL CONSTRAINT df DEFAULT (GETDATE()),
    myTable_PK PRIMARY KEY CLUSTERED (ID, ColB)
);
```

DROP

Elimina un objeto existente

```
DROP TABLE { tabla } [ ; ]
```

```
DROP TABLE dbo.myTable;
```

ALTER

Modifica un objeto existente

```
ALTER TABLE { tabla } (
[ modificación de tabla ]
[ modificación de columna ]
[ modificación de restricción ]
[ modificación de índices ]
) [ ; ]
```

```
ALTER TABLE dbo.myTable ADD ColF INT NOT NULL;
ALTER TABLE dbo.myTable ALTER COLUMN ColF FLOAT NOT NULL;
ALTER TABLE dbo.myTable DROP COLUMN ColF;
ALTER TABLE dbo.myTable DROP CONSTRAINT myTable_PK;
```

SQL | DML

Tal como su nombre indica, nos permite manipular los datos.

INSERT

Introduce registro(s) en una tabla

```
INSERT [INTO] { table } [(col1[,…n])]
{
  { VALUES ({DEFAULT | NULL | <expression>] [,…n] })
  |derived_table }
} [;]
```

```
INSERT INTO dbo.myTable (ID, colB, colC, colD, colE)
VALUES (1, 2, 3, 0.5, '20100101');
```

DELETE

Elimina registro(s)

```
DELETE [FROM] { table } [WHERE <search_condition>] [ ; ]
```

```
DELETE dbo.myTable WHERE ID = 1;
```

UPDATE

Modifica registro(s)

```
UPDATE { table } SET {col_name = {<expression> |
DEFAULT | NULL}}
[FROM { table_source} ]
[WHERE {search_condition}] [ ; ]
```

```
UPDATE dbo.myTable SET colB = 10 WHERE ID = 1;
```

SQL | DML

Además de las clásicas, existe una que nos permite ejecutar todas ellas en la misma instrucción.

MERGE

Inserta, actualiza o ejecuta operaciones de borrado en la tabla destino a partir del resultado de una combinación con una tabla de origen

```
MERGE { target_table }
USING { source_table }
ON <merge_search_condition>
WHEN MATCHED THEN <merge_matched>
WHEN NOT MATCHED THEN <merge_not_matched>
WHEN NOT MATCHED BY SOURCE <merge_not_matched>
[;]
```

```
MERGE dbo.myTable
USING dbo.myOtherTable
ON myTable.ID = myOtherTable.ID
WHEN MATCHED THEN UPDATE SET colB = 20
WHEN NOT MATCHED THEN INSERT (colB) VALUES (myOtherTable.colZ)
WHEN NOT MATCHED BY SOURCE THEN DELETE;
```

Lab 01

DDL

Lab 01

15:00 Stop

- Basado en https://github.com/ih-datapt-mad/dataptmad0223_labs/tree/main/module-1/mysql http://bit.ly/3Zd50Tw
- Pero usando un entorno de trabajo virtual: https://dbfiddle.uk/ (Chrome)
- Vamos a tratar de crear las instrucciones CREATE TABLE a partir de los datos de ejemplo

Cars

ID	VIN	Manufacturer	Model	Year	Color
0	3K096I98581DHSNUP	Volkswagen	Tiguan	2019	Blue
1	ZM8G7BEUQZ97IH46V	Peugeot	Rifter	2019	Red
2	RKXVNNIHLVVZOUB4M	Ford	Fusion	2018	White
3	HKNDGS7CU31E9Z7JW	Toyota	RAV4	2018	Silver
4	DAM41UDN3CHU2WVF6	Volvo	V60	2019	Gray
5	DAM41UDN3CHU2WVF6	Volvo	V60 Cross Country	2019	Gray

Salespersons

ID	Staff ID	Name	Store
0	00001	Petey Cruiser	Madrid
1	00002	Anna Sthesia	Barcelona
2	00003	Paul Molive	Berlin
3	00004	Gail Forcewind	Paris
4	00005	Paige Turner	Mimia
5	00006	Bob Frapples	Mexico City
6	00007	Walter Melon	Amsterdam
7	00008	Shonda Leer	São Paulo

Customers

ID	Customer ID	Name	Phone	Email	Address	City	State/Province	Country	Postal
0	10001	Pablo Picasso	+34 636 17 63 82	-	Paseo de la Chopera, 14	Madrid	Madrid	Spain	28045
1	20001	Abraham Lincoln	+1 305 907 7086	-	120 SW 8th St	Miami	Florida	United States	33130
2	30001	Napoléon Bonaparte	+33 1 79 75 40 00	-	40 Rue du Colisée	Paris	Île-de-France	France	75008

Invoices

ID	Invoice Number	Date	Car	Customer	Sales Person
0	852399038	22-08-2018	0	1	3
1	731166526	31-12-2018	3	0	5
2	271135104	22-01-2019	2	2	7

- Una vez hecho, el siguiente paso es tratar de crear las instrucciones INSERT INTO
- Y por último (bonus time), tratemos de crear las instrucciones UPDATE y DELETE propuestas

SQL | DML

Pero sin duda, el protagonista del lenguaje SQL es la posibilidad que nos ofrece para consultar y extraer los datos almacenados tal como necesitamos.

SELECT

Devuelve registros de la base de datos y permite la selección de uno o varios registros y columnas de una o varios conjuntos de datos de origen.

```
SELECT [ TOP (top_expression) ] [ ALL | DISTINCT ]
[ * | column_name | <expression>] [,…n]
[ FROM { source_table } [, …n] ]
[ WHERE <search_condition> ]
[ GROUP BY <group_by_clause> ]
[ HAVING  <search_condition> ]
[ ORDER BY  <order_by_expression> ]
[;]
```

```
SELECT * FROM dbo.myTable;
```

SELECT | Logical order

¿Sabías, sin embargo, que el orden de ejecución de las cláusulas es distinto?

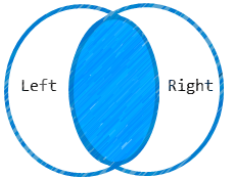
1. **`FROM`**. Esta fase identifica las tablas de origen y las combina según los criterios establecidos.
2. **`WHERE`**. A partir del conjunto de resultados anterior, le aplica los filtros establecidos.
3. **`GROUP BY`**. Esta fase organiza el conjunto de resultados en grupos que cumplan las condiciones establecidas.
4. **`HAVING`**. Filtra los grupos de la fase anterior que cumplan las condiciones establecidas.
5. **`SELECT`**. Esta fase procesa los elementos de esta cláusula, resolviendo las expresiones que existan y aplicando el criterio de unicidad en caso de que exista la cláusula DISTINCT.
6. **`ORDER BY`**. Se ordena el conjunto de datos de la fase anterior según la lista indicada.
7. **`TOP`**. Filtra registros del conjunto de datos anterior.

SELECT | Joins

- Una de las capacidades más potentes de SQL es el de combinar de múltiples maneras distintos conjuntos de datos: por ejemplo, en Lab01 saber el nombre de los clientes (*Customers*) y sus compras (*Invoices*).
- Al combinar conjuntos de datos, el primero que se declara se le considera la parte izquierda (*left*) de la combinación y al segundo la parte derecha (*right*). Dependiendo del tipo de combinación, tener claro este dato es importante.

INNER JOIN

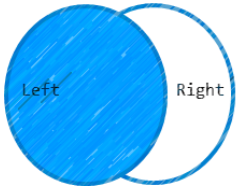
Registros que existen en ambos conjuntos de datos.
Es el predeterminado si no se especifica.



"Ventas realizadas"

LEFT JOIN

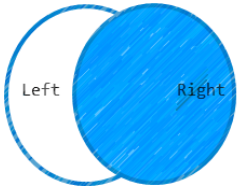
Todos los registros de la tabla izquierda de la combinación.
Los registros no existentes se ponen a NULL.



"Clientes registrados (con o sin facturas)"

RIGHT JOIN

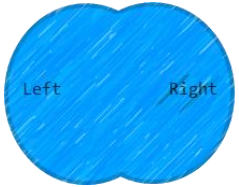
Todos los registros de la tabla derecha de la combinación
Los registros no existentes se ponen a NULL.



"Facturas con datos de clientes (o no)"

FULL JOIN

Todos los registros de ambas tablas.
Los registros no existentes se ponen a NULL.



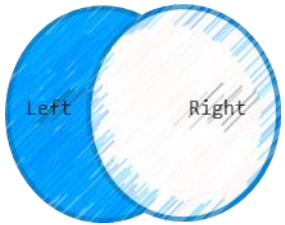
"Clientes con y sin ventas y facturas con y sin cliente"

SELECT | Joins

- Existen combinaciones más avanzadas

EXCLUSIVE LEFT JOIN

Registros que solo se encuentran en la parte izquierda de la combinación.

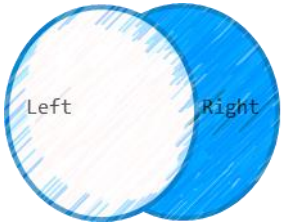


Remember

"Clientes sin ventas (sin facturas)"

EXCLUSIVE RIGHT JOIN

Registros que solo se encuentran en la parte derecha de la combinación.

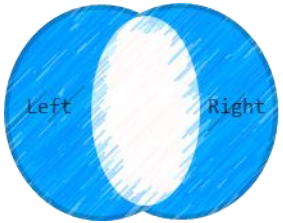


Remember

"Facturas de clientes inexistentes (tal vez erróneas)"

EXCLUSIVE FULL JOIN

Registros que solo se encuentran en la parte izquierda o que solo se encuentran en la parte derecha.
Los registros no existentes se ponen a NULL.



Remember

"Clientes sin ventas y facturas sin clientes"

SELECT | Joins

- Es posible combinar tablas con una sintaxis basada (ANSI-89) en la cláusula WHERE

```
SELECT *
FROM Customers, Invoices
WHERE Customers.ID = Invoices.Customer -- condición de combinación
 AND Customer.City = 'Madrid' -- condición adicional
```

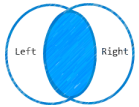
- Pero hoy en día (ANSI-92) es más normal (y recomendable) usar la cláusula `[ INNER | LEFT | RIGHT | FULL ] JOIN`

```
SELECT *
FROM Customers INNER JOIN Invoices ON Customers.ID = Invoices. Customer -- condición de combinación
WHERE  Customer.City = 'Madrid' -- condición adicional
```

SELECT | Joins

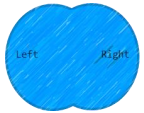
INNER JOIN

```
SELECT *
FROM Customers INNER JOIN Invoices
ON Customers.ID = Invoices.Customer
```



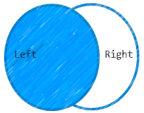
FULL JOIN

```
SELECT *
FROM Customers FULL JOIN Invoices
ON Customers.ID = Invoices.Customer
```



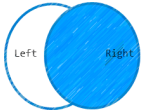
LEFT JOIN

```
SELECT *
FROM Customers LEFT JOIN Invoices
ON Customers.ID = Invoices.Customer
```



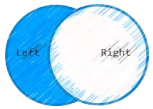
RIGHT JOIN

```
SELECT *
FROM Customers RIGHT JOIN Invoices
ON Customers.ID = Invoices.Customer
```



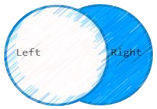
EXCLUSIVE LEFT JOIN

```
SELECT *
FROM Customers LEFT JOIN Invoices
ON Customers.ID = Invoices.Customer
WHERE Invoices.ID IS NULL
```



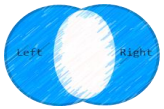
EXCLUSIVE RIGHT JOIN

```
SELECT *
FROM Customers RIGHT JOIN Invoices
ON Customers.ID = Invoices.Customer
WHERE Customers.ID IS NULL
```



EXCLUSIVE FULL JOIN

```
SELECT *
FROM Customers FULL JOIN Invoices
ON Customers.ID = Invoices.Customer
WHERE Customers.ID IS NULL OR
Invoices.ID IS NULL
```



SELECT | Joins

- Existe una combinación adicional que permite hacer **productos cartesianos**: `CROSS JOIN`
- Cada registro de la tabla izquierda se combina con un registro de la tabla derecha, obteniendo un conjunto que es el resultado de ***n* x *m* filas** de los conjuntos originales.
- No existe la cláusula `ON` porque **no hay condición de combinación** como tal.
- Su uso es más limitado, pero es muy práctico para generar conjuntos de datos de ejemplo.

```
SELECT *
FROM Customers CROSS JOIN Invoices
```

SELECT | Joins

- Es posible usar **varias** instrucciones JOIN para combinar varias tablas.
- Es posible usar los **diferentes tipos** de JOIN en la misma instrucción.
- Es posible combinar una tabla **consigo misma** (por ejemplo en relaciones de jerarquía).
- En cualquier caso, es importante prestar atención a lo que indicamos en la **cláusula `ON`** para evitar combinaciones incorrectas que arrojen resultados inesperados.

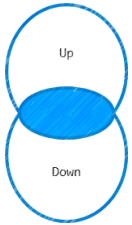
```
SELECT *
FROM Cars INNER JOIN Invoices ON Cars.ID = Invoices.Car -- condición de combinación
INNER JOIN Customers ON Invoices.Customer = Customers.ID -- condición de combinación
```

SELECT | Set Operators

- Existen otras formas de operar con varias tablas además de combinaciones.
- A través de los operadores de conjuntos podemos **sumar o excluir** varios conjuntos de resultados en uno solo.
- Es cierto que al final combinamos datasets, pero en un caso (JOINs) lo ensanchamos (añadimos columnas) y en otro (Set Operators) lo **alargamos** (añadimos registros).
- La condición es que los datasets deben tener **igual forma** (número y tipo de datos de las columnas).

INTERSECT

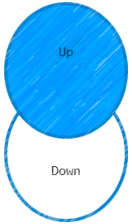
Devuelve las filas únicas que se encuentran en el los dos datasets.




```
SELECT ID
FROM table1
INTERSECT
SELECT ID
FROM table2
```

EXCEPT

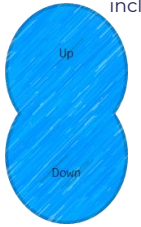
Devuelve los registros únicos que sólo están en el dataset inicial.




```
SELECT ID, c1
FROM table1
EXCEPT
SELECT ID, c2
FROM table2
```

UNION [ALL]

Concatena los resultados de dos consultas en un único dataset. ALL incluye duplicados




```
SELECT c1, c4
FROM table1
UNION
SELECT c2, c6
FROM table2
```

SELECT | ORDER

- El conjunto de resultados se puede **ordenar** en base a una condición específica.
- Es importante tener en cuenta que si no se especifica esta cláusula, **no hay garantía** de obtener un orden concreto en el resultado.
- En el criterio se puede usar el nombre de la columna, el alias o la posición que ocupa en la cláusula `SELECT` (aunque esto último no está recomendado) .
- Puede ser un criterio de ordenación compuesto por varias expresiones

ORDER BY

Ordena el resultado en base a los criterios establecidos

[ORDER BY {<order_by_expression> [ASC | DESC] } [,…]]

```
SELECT * FROM dbo.myTable ORDER BY c1, c2 DESC;
```

Lab 02

JOIN

Lab 02

- Basado en la bbdd creada en Lab 01
- Usando el entorno de trabajo virtual: https://dbfiddle.uk/ (Chrome)
- Probar los diferentes tipos de JOIN mostrados en la teoría combinando las tablas que consideréis (siempre que tengan sentido)
- `INNER JOIN, FULL JOIN, [INCLUSIVE | EXCLUSIVE] LEFT JOIN, [INCLUSIVE | EXCLUSIVE] RIGHT JOIN, CROSS JOIN`

SELECT | WHERE

- Se usa para **filtrar** registros del conjunto de datos.
- No sólo aplica a la instrucción `SELECT`, se puede usar también en `UPDATE` e `INSERT`.
- Existen diferentes expresiones lógicas, pudiéndose **combinar** con operadores lógicos `AND` y `OR`.

`=`	Igualdad
`>`	Mayor que
`<`	Menor que
`>=`	Mayor o igual
`<=`	Menor o igual
`<>`	Distinto. En algunos casos también se puede usar !=
`[NOT] BETWEEN … AND`	Entre
`[NOT] LIKE`	Patrón. Pueden usarse expresiones regulares como % (cualquier carácter), _ (un carácter), [] (cualquier carácter dentro del rango), [^] (cualquier carácter que no esté en el rango)
`[NOT] IN`	Valores posibles

SELECT | Subconsultas

- Son consultas **dentro** de consultas.
- Se usan a menudo para filtrar datos de una forma más avanzada, y a veces por legibilidad.
- Pueden ser **escalares** (solo devuelven un único valor), **multivalor** o como **tabla** (múltiples campos, múltiples registros).
- Cuando la subconsulta tiene una **dependencia** con la consulta principal se denominan *consultas correlacionadas*.

SUBCONSULTA ESCALAR

Devuelve un único registro

```
SELECT *
FROM Invoices
WHERE “Date” =
(SELECT MAX(“Date”) FROM Invoices)
```

SUBCONSULTA MULTIVALOR

Devuelve múltiples registros

```
SELECT *
FROM SalesPerson
WHERE ID IN
(SELECT “Sales Person” FROM Invoices)
```

SUBCONSULTA CORRELACIONADA

Depende de la consulta exterior.

```
SELECT *
FROM Orders AS o1
WHERE OrderDate =
(SELECT MAX(OrderDate) FROM Orders AS o2
WHERE o1.Customer = o2.customer)
```

SELECT | Subconsultas

- Siempre van encerradas entre **paréntesis**.
- Los datos que devuelve **no se incluyen** en el conjunto de datos final.
- Pueden ir precedidas de distintas **cláusulas**
    - [NOT] ALL, devuelve TRUE cuando todos los valores retornados por la subconsulta satisfacen la comparación establecida.
    - [NOT] [SOME | ANY], devuelve TRUE cuando cualquier valor devuelto por la subconsulta satisface la comparación establecida.
    - [NOT] EXISTS, para comprobar si la subconsulta devuelve algún registro.

SELECT | Alias

- Nombre **alternativo** asignado a una tabla, subconsulta o columna, añadiendo la cláusula **AS** después del nombre de dicho objeto junto con el nombre del alias asignado.
- Necesario cuando la misma entidad aparece **varias veces** en la instrucción.
- Pero también es muy útil para **legibilidad**.

```
SELECT o1.OrderDate AS oDate
FROM Orders AS o1
WHERE OrderDate =
(SELECT MAX(OrderDate) FROM Orders AS o2 WHERE o1.Customer = o2.customer)
```

SELECT | Vistas

- Una forma de **reutilizar** una subconsulta (no correlacionada) es creando una vista.
- Una vista no es más que un **alias** dado a una consulta cuya definición (**no los datos**) se persiste.
- Tiene varias ventajas:
    - Permite **reutilizar** una consulta en varios sitios.
    - Permite **simplificar** una consulta compleja.
    - Permite ofrecer un **interfaz de acceso** a datos más amigable y seguro para consumidores finales.

VISTA

Almacena la definición de una consulta

```
[ CREATE | ALTER ] VIEW {view_name} AS <select_statement>
```

SELECT | CTE

- Acrónimo de **Common Table Expression**, es un nombre temporal dado a un resultset.
- Como si definiéramos una **vista temporal** con alcance dentro de la instrucción SELECT.
- Pueden crearse definiciones de **varias instrucciones**.
- Tiene varias ventajas:
    - Permite **reutilizar** una consulta en varios sitios.
    - Permite **simplificar** una consulta compleja.
    - Hace más **legible** el código

CTE

Especifica un nombre temporal a un resulset

WITH <cte_name> AS (<cte_query_definition>)

```
WITH max_values AS (SELECT MAX('Date') AS d FROM Invoices)
SELECT *
FROM Invoices
WHERE 'Date' = (SELECT d FROM max_values)
```

SQL | Summary

#1: Hemos visto los diferentes *dialectos* que tiene SQL.

#2: Hemos aprendido diferentes formas de combinar datasets.

#3: Y hemos empezado a filtrar conjuntos de datos.

Las bbdd y el cloud computing

Cloud Computing

- Aunque no específico a las bbdd, el cloud ha permitido usar (despliegue, escalado, testing, etc.) más fácilmente los SGBDR.
- Conocer sus fundamentos nos va a ofrecer otro punto de vista para trabajar con estos sistemas.

Definición

Qué es el cloud computing ?

*“Disponibilidad de servicios de **computación a demanda***

*y con pago por uso a través de **Internet***

*sin **mantenimiento directo** por parte del usuario”*

Cloud Computing | Beneficios

Ahorro

No solo evita la necesidad de comprar, poseer y mantener data centers físicos y pagar por lo que realmente se usa, sino que ese gasto es mucho menor que si fuera propio (economía de escala).

Agilidad

Fácil acceso a muchos tipos de servicios y tecnologías.
Agilidad tanto para desplegarlos como para probar nuevas ideas.

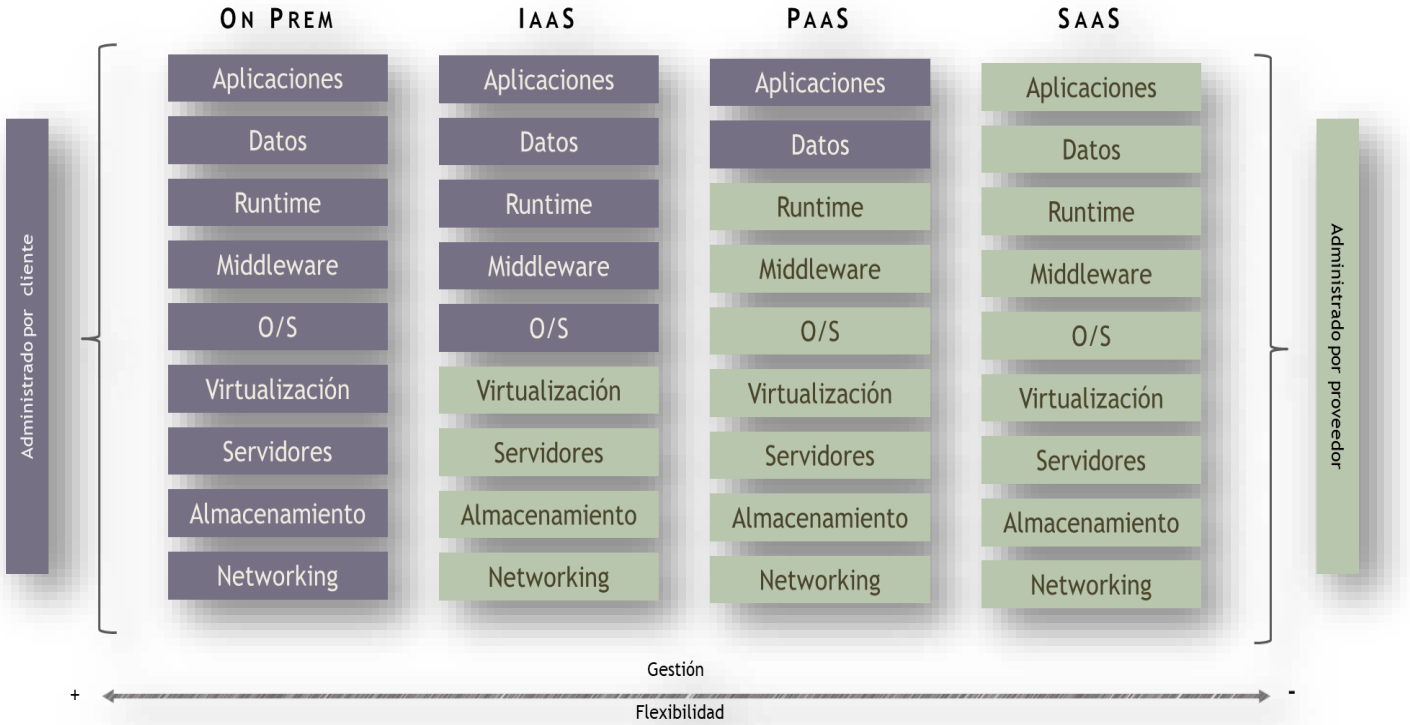
Global

Posibilidad de expandirse a nuevas regiones a lo largo del globo a un golpe de clic, acercando las aplicaciones a los usuarios finales y de ese modo reducir latencias.

Elasticidad

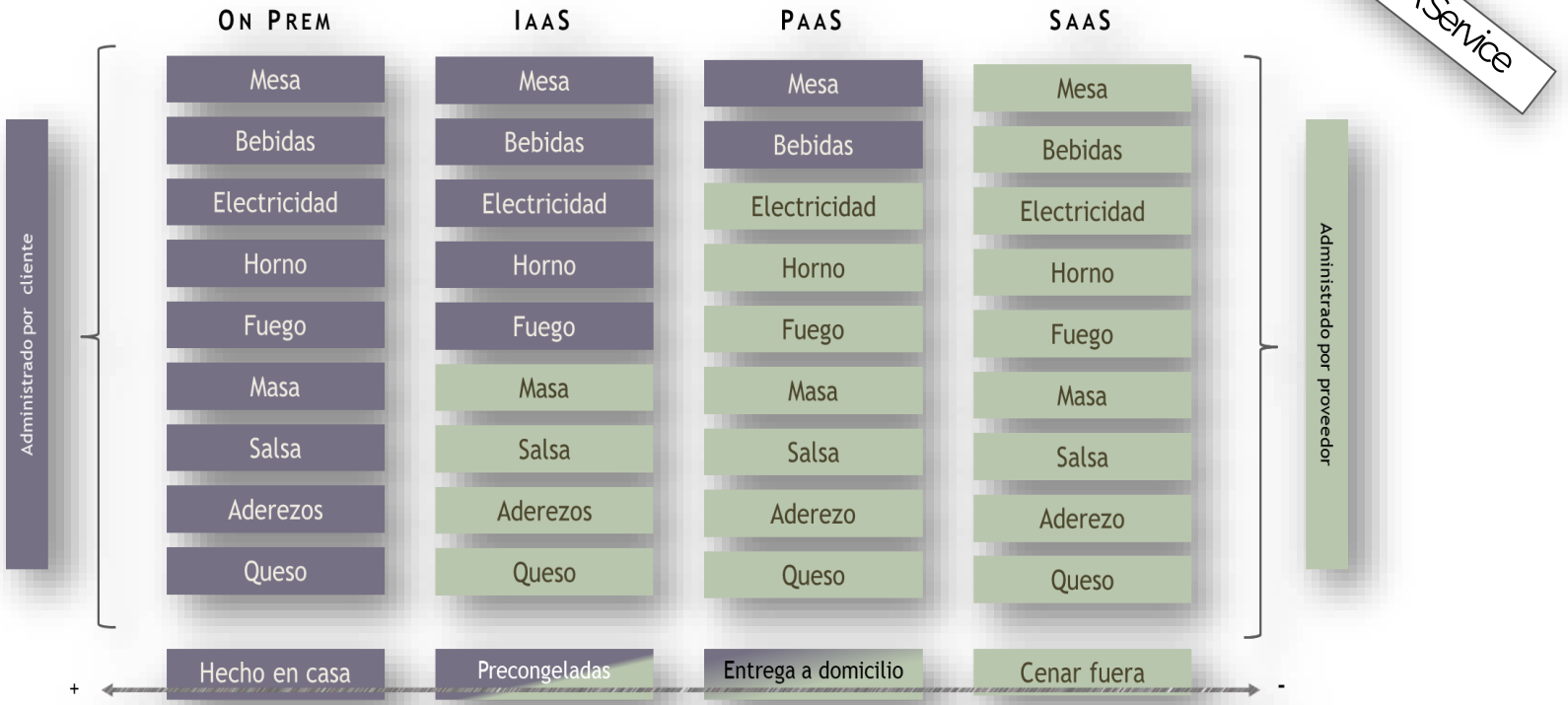
Adaptar los recursos a las necesidades de cada momento, sin necesidad de sobreestimar por crecimientos futuros o por picos puntuales.

Cloud Computing | Tipos según despliegue

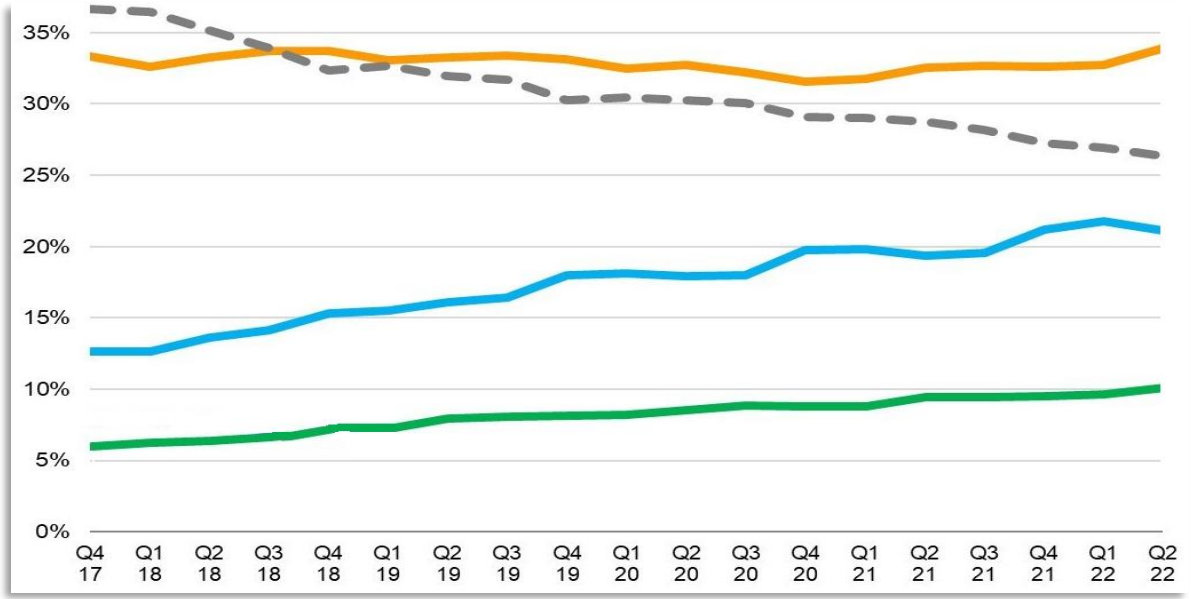


Cloud Computing | Tipos según despliegue

Pizza As A Service



Cloud Computing | Vendors



Fuente Synergy Research Group

Lab 03

CLOUD

Lab 03

- Registrarnos en Azure para usar su free tier: https://azure.microsoft.com/en-us/free/ https://bit.ly/3N9mTgc
    - Necesitaremos una cuenta Live y un número de tarjeta para registrarnos (no hacen cargo alguno)

- Seguiremos los pasos indicados en https://docs.microsoft.com/en-us/azure/azure-sql/database/single-database-create-quickstart?tabs=azure-portal, especificando: https://bit* ly/3MVoOok

Grupo de recursos	`rg_ironhack`
Server name	`sql<iniciales del alumno>123`
Location	`(Europe) West Europe`
Database	`AdventureWorks`
User name	`azureuser`
Password	`P@$$w0rd123`

- Pulsamos en “Go to resource” cuando haya terminado de desplegarse el servicio.

Lab 03

- Nos descargamos Azure Data Studio de https://docs.microsoft.com/en-us/sql/azure-data-studio/download-azure-data-studio?view=sql-server-ver15

https://bit.ly/3u5O7xB

- Una vez instalado, lo lanzamos y creamos una nueva conexión hacia el servidor recién desplegado:

Server	`sql<iniciales del alumno>123.database.windows.net`
Authentication type	`SQL Login`
User name	`azureuser`
Password	`P@$$w0rd123`
Database	`AdventureWorks`

- Pulsamos en “New query” y escribimos la siguiente instrucción:

```
SELECT TOP 20 pc.Name as CategoryName, p.name as ProductName
FROM SalesLT.ProductCategory pc JOIN SalesLT.Product p ON pc.productcategoryid = p.productcategoryid
ORDER BY CategoryName;
```

Lab 03

5:00 Stop

- **Challenge 1 – Qué ha comprado cada cliente?**
  - Hay que combinar varias tablas para saber qué productos ha comprado cada cliente
  - Hay que considerar como mínimo las columnas `First Name, Last Name (SalesLT.Customer)` y `Name (SalesLT.Product)`, pero la salida debe ser como la imagen siguiente:

	Customer Fullname	Product Name
1	Andrea Thomsen	Rear Brakes
2	Anthony Chor	HL Touring Frame - Blue, 50
3	Anthony Chor	HL Touring Frame - Blue, 54
4	Anthony Chor	HL Touring Frame - Blue, 60
5	Anthony Chor	HL Touring Frame - Yellow, 60
6	Anthony Chor	HL Touring Handlebars
7	Anthony Chor	HL Touring Seat/Saddle
8	Anthony Chor	LL Touring Frame - Yellow, 44
9	Anthony Chor	LL Touring Frame - Yellow, 50

La imagen no muestra la salida completa

Lab 03

5:00 Stop

- **Challenge 2 – Mostrar la descripción árabe del producto cuyo código es el 710**
  - La cultura árabe tiene la abreviatura ‘ar’
  - La salida debe ser como la imagen siguiente:

	Product Model	Description
1	Mountain Bike Socks	…يجفافها وتعمل كوسائد ملائمة

La imagen muestra la salida completa

Lab 03

5:00 Stop

- **Challenge 3 (B O N U S) – Total de ventas por producto, ordenado descendentemente**
  - La salida debe ser como la imagen siguiente:

	name	Total Orders
1	Classic Vest, S	10
2	Long-Sleeve Logo Jersey, L	10
3	AWC Logo Cap	9
4	Short-Sleeve Classic Jersey,…	9
5	Short-Sleeve Classic Jersey,…	8
6	Hitch Rack - 4-Bike	8
7	Bike Wash - Dissolver	7
8	Front Brakes	7

La imagen no muestra la salida completa

Lab 03 | Soluciones

- **Challenge 1 – Qué ha comprado cada cliente?**

```
SELECT c.FirstName + ' ' + c.LastName AS [Customer Fullname], p.Name AS [Product Name]
FROM SalesLT.Customer AS c
    INNER JOIN SalesLT.SalesOrderHeader AS soh ON c.CustomerID = soh.CustomerID
    INNER JOIN SalesLT.SalesOrderDetail AS shd ON soh.SalesOrderID = shd.SalesOrderID
    INNER JOIN SalesLT.Product AS p ON shd.ProductID = p.ProductID
ORDER BY [Customer Fullname], [Product Name]
```

- **Challenge 2 – Mostrar la descripción árabe del producto cuyo código es el 710**

```
SELECT  pm.Name AS 'Product Model', pd.[Description]
FROM SalesLT.ProductModel AS pm
    INNER JOIN SalesLT.ProductModelProductDescription AS pmpd ON pm.ProductModelID = pmpd.ProductModelID
    INNER JOIN SalesLT.ProductDescription AS pd on PD.ProductDescriptionID = pmpd.ProductDescriptionID
    INNER JOIN SalesLT.Product AS p ON p.ProductModelID = pm.ProductModelID
WHERE pmpd.Culture = 'ar' AND p.ProductID = 710;
```

- **Challenge 3 (B O N U S) – Total de ventas por producto, ordenado descendentemente**

```
SELECT   p.name, COUNT(*) AS 'Total Orders'
FROM SalesLT.Product AS p
    INNER JOIN SalesLT.SalesOrderDetail AS sod ON p.ProductID = sod.ProductID
GROUP BY  p.Name
ORDER BY 'Total Orders' DESC
```

SQL | tips

#1: *La codificación en SQL es de **dentro a afuera** (a diferencia de en un programa que es de arriba abajo o viceversa): se prueba la consulta más interna, luego se superpone otra consulta y se prueba, etc.*

#2: *Eliminar registros innecesarios **lo antes posible**, preferiblemente en consultas internas.*

#3: *Aprender a pensar en términos de **conjuntos de registros** y operaciones de conjunto; evitando en la medida de lo posible el procesamiento de "un registro a la vez".*

#4: *Entender cómo funciona el **optimizador de consultas** para poder escribir un código SQL mejor y más eficiente.*

#5: *Utiliza el **servidor** para que haga la parte pesada de la consulta, devolviendo solo los datos necesarios*

Fuente: https://www.linkedin.com/feed/update/urn:li:activity:7021231884349173760/

SQL | Summary

#1: Nos hemos iniciado en el cloud computing.

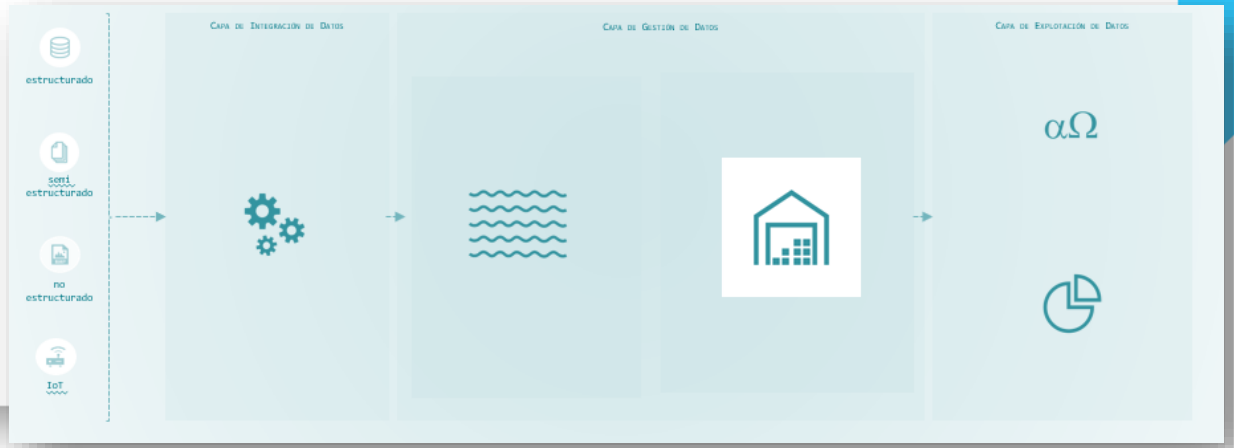
#2: Hemos desplegado una base de datos en Azure, el cloud provider de Microsoft.

#3: Hemos interactuado con una base de datos más completa y practicado SQL con ella.

Mundo analítico

Analítica en base de datos

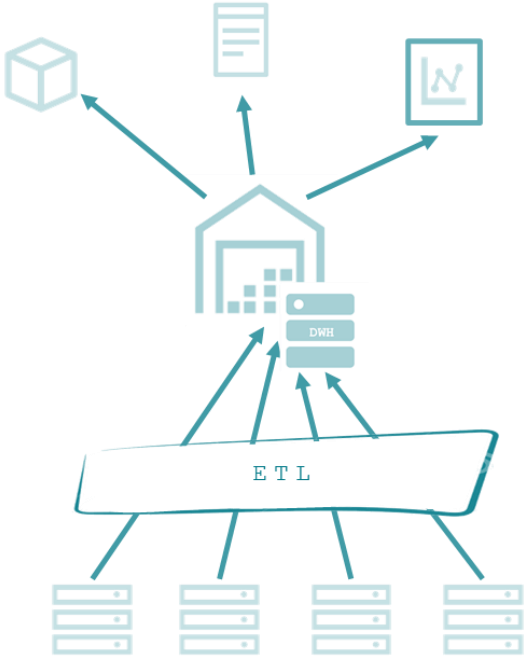
- En el último ejercicio del laboratorio aparecía una pequeña función (`COUNT`) que no hemos visto aun, pero que abre un nuevo mundo en el uso de las bases de datos: el mundo informacional.
- Gracias (en parte) a estas funciones, es posible empezar a extraer información de los datos.
- Pero también gracias a un concepto, igualmente con cierta carga negativa: el data warehouse



Data Warehouse

- Ese otro repositorio creado para dar soporte a los informes analíticos, destino final de información proveniente de distintas fuentes, es lo que se denomina Data Warehouse

- En otras palabras: es el sitio en donde los responsables se basarán para tomar las **decisiones** de la empresa, puesto que es donde se encuentran persistidos los datos historificados, orientados a negocio y confiables.



Kimball

Los datos se organizan en hechos e información de referencia

(Bottom-Up) El EDW es un compendio de datamarts departamentales

Agilidad en la construcción

Fácilmente entendible por negocio

El rendimiento es muy bueno

Las herramientas de BI entienden el modelado dimensional

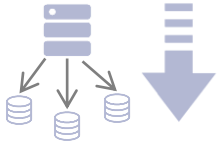
Inmon

Se siguen las reglas de normalización de bbdd (E.F.Codd) en la construcción del DWH

(Top-Down) Primero se crea el EDW y sobre él los datamart departamentales

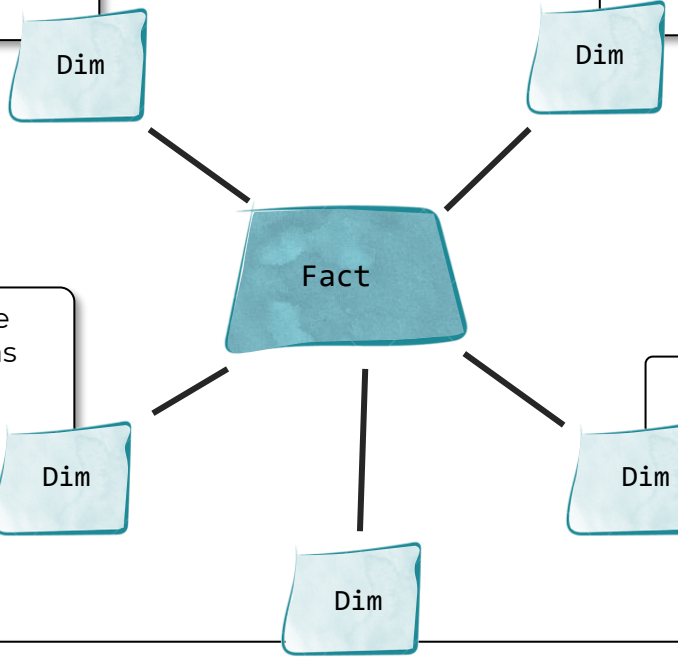
El EDW es realmente el único punto de verdad del dato

Es relativamente sencillo añadir información (muchas tablas normalizadas)



Es la técnica más común para la construcción de un DWH

Fue desarrollada por Ralph Kimball en su libro “The Data Warehouse Toolkit”

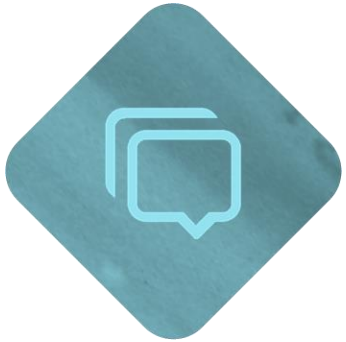


Tiene la ventaja añadida que este modelo es entendido por muchas herramientas, lo que hace la navegación más sencilla para usuarios no técnicos

La alternativa es usar las formas normales, pero no está optimizada para el propósito de un DWH

El modelado dimensional está diseñado para:

- ser comprensible por usuarios de negocio
- ser eficiente en la respuesta a las consultas de los informes



El concepto más importante en el modelado dimensional es saber dividir los datos en dos categorías:

- Conjuntos de datos delimitados (datos de referencia), también llamados **dimensiones**
- Conjuntos de datos sin límites, también llamados **hechos**

Hechos

Representan **eventos del negocio** (ventas, envíos, etc.) sobre los que se definen las medidas.

Se implementan físicamente en las **tablas de hechos**, que crecen rápidamente y no se modifican.

Estas tablas solo contienen esas **medidas y claves** que apuntan a las dimensiones

Dimensiones

Proporcionan el contexto alrededor del evento: el **quién, qué, dónde y cuándo**

Se implementan en **tablas de dimensiones**, no contienen tanta información pero sí suelen modificarse sus datos.

Contienen propiedades que definen la entidad, muchas veces gobernadas por un responsable

SELECT | Agregaciones

- La implementación de los conceptos que acabamos de ver se apoyan en las **consultas de agregación** para mostrar todo su potencial.
- Hasta ahora nos hemos centrado en manipular y consultar datos, así como aprender las técnicas básicas de modelado para sistemas operacionales.
- Veamos ahora cómo podemos extraer información de esos datos para poder tomar **decisiones informadas**.

SELECT | Agregaciones

- La cláusula `GROUP BY` permite dividir el resultado en **conjuntos** de filas.
- Normalmente, esos grupos se les usa para realizar algún tipo de **cálculo sobre ellos**.
- Los grupos se determinan en base a las **columnas o expresiones** que se especifican en la cláusula.
- Las expresiones que **no forman parte** de la función de agregado **deben aparecer** en la cláusula `GROUP BY`.
- Recordad que la cláusula `WHERE` elimina las filas **antes** de que `GROUP BY` las ponga en el grupo correspondiente.

GROUP BY

Divide el conjunto de resultados en conjuntos de filas

GROUP BY { <column_expression> | GROUPING SETS (<grouping_set> [,…]) }[,…]

```
SELECT c1, COUNT(*) AS 'total' FROM dbo.myTable GROUP BY c1;
```

SELECT | Agregaciones

- Pero la cláusula `GROUP BY` necesita verse acompañada por una **función de agregación** que permita realizar **cálculos** sobre cada uno de los grupos que nos devuelve.
- Funciones de agregación hay muchas, incluso cada motor implementa las suyas propias, pero como norma general podemos encontrarnos con las siguientes

SUM ([ALL | DISTINCT] <expression>)	Devuelve la suma de todos los valores del grupo
*COUNT ([ALL | DISTINCT] <expression> | *)*	Devuelve el número de elementos encontrados en el grupo
MAX ([ALL | DISTINCT] <expression>)	Devuelve el valor máximo de los valores del grupo
MIN ([ALL | DISTINCT] <expression>)	Devuelve el valor mínimo de los valores del grupo
AVG ([ALL | DISTINCT] <expression>)	Devuelve la media de todos los valores del grupo
VAR ([ALL | DISTINCT] <expression>)	Devuelve la varianza de todos los valores del grupo
STDEV ([ALL | DISTINCT] <expression>)	Devuelve la desviación estándar de todos los valores del grupo

SELECT | Agregaciones

- La cláusula `GROUP BY` es **flexible** para incluir varias columnas o expresiones por las que agrupar.
- Pero al final, las funciones de agregado **aplican a los grupos** que conforman la unión de esas columnas o expresiones
- Sin embargo, también tenemos disponible la opción `GROUPING SETS` que nos permite especificar **diferentes grupos** en la misma consulta**.**
- Dicho de otro modo: es como si estuviéramos ejecutando **varias instrucciones `GROUP BY` independientes** en las que solo cambiara las columnas o expresiones por las que agrupamos los resultados.

```
SELECT col1, col2, COUNT(*) AS total
FROM t1
GROUP BY GROUPING SETS ((col1), (col1, col2))
```

SELECT | Agregaciones

- La cláusula `HAVING` nos permite **filtrar grupos** que no cumplan las condiciones establecidas.
- `HAVING` es a `GROUP BY` lo que `WHERE` es al `FROM`.
- Como no se filtran registros individuales sino grupos, siempre se usa con una **función de agregación**.

```
SELECT col1, col2, COUNT(*) AS total
FROM t1
GROUP BY col1, col2
HAVING COUNT(*) > 10
```

Lab 04

AGREGACIONES

Lab 04

- Conectamos a nuestra base de datos con Azure Data Studio

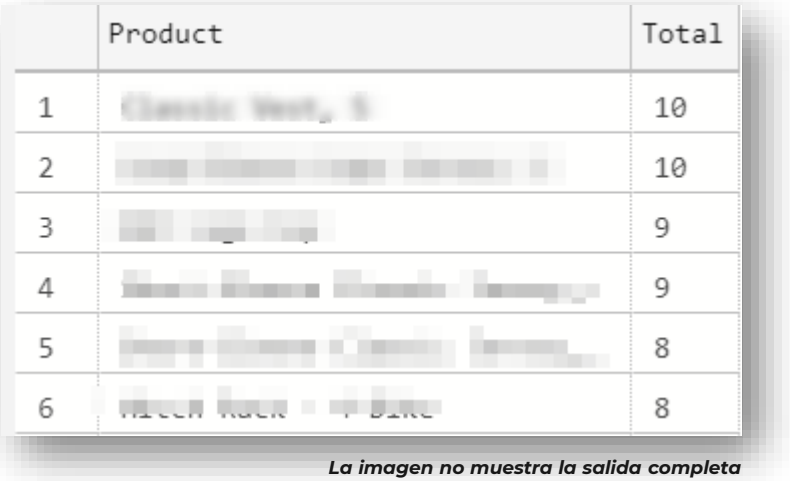
Server	`sql<iniciales del alumno>123.database.windows.net`
Authentication type	`SQL Login`
User name	`azureuser`
Password	`P@$$w0rd123`
Database	`AdventureWorks`

- Vamos a hacer unos retos para practicar con las agrupaciones

Lab 04

5:00 Stop

- **Challenge 1 – Qué producto ha sido el más vendido?**
  - Por “más vendido” nos referimos a número de tickets, no unidades vendidas
  - Para encontrarlo, habrá que ordenar por el número de ventas de forma descendente.



	Product	Total
1	[illegible]	10
2	[illegible]	10
3	[illegible]	9
4	[illegible]	9
5	[illegible]	8
6	[illegible]	8

La imagen no muestra la salida completa

Lab 04

5:00 Stop

- **Challenge 2 – Cuántas unidades han sido vendidas por categoría y producto?**
  - Ordena el resultado por Categoría y Producto.

	Category	Product	Total Qty
1	Bike Racks	Hitch Rack - 4-Bike	32
2	Bottles and Cages	Water Bottle - 30 oz.	54
3	Bottom Brackets	HL Bottom Bracket	15
4	Bottom Brackets	LL Bottom Bracket	7
5	Brakes	Front Brakes	12
6	Brakes	Rear Brakes	1
7	Cans	AWC Logo Can	52

La imagen no muestra la salida completa

Lab 04

5:00 Stop

- **Challenge 3 – Cuántas unidades han sido vendidas por categoría, y por categoría-producto?**
  - Ordena el resultado por Categoría y Producto.
  - Hay que hacerlo en una sola instrucción.

	Category	Product	Total Qty
1	Bike Racks	*NULL*	32
2	Bike Racks	Hitch Rack - 4-Bike	32
3	Brakes	*NULL*	13
4	Caps	*NULL*	52
5	Caps	AWC Logo Cap	52
6	Derailleurs	*NULL*	21
7	Gloves	*NULL*	57

La imagen no muestra la salida completa

Lab 04

5:00 Stop

- **Challenge 4 (B O N U S) – De la consulta anterior, descarta los grupos que tengan menos de 8 tickets de venta**
  - 1 ticket de venta es 1 registro de la tabla SalesLT.SalesOrderHeader
  - Ordena el resultado por Categoría y Producto.

	Category	Product	Total Qty
1	Bike Racks	*NULL*	32
2	Bike Racks	Hitch Rack - 4-Bike	32
3	Brakes	*NULL*	13
4	Caps	*NULL*	52
5	Caps	AWC Logo Cap	52
6	Handlebars	*NULL*	27
7	Helmets	*NULL*	124

La imagen no muestra la salida completa

Lab 04 | Soluciones

- **Challenge 1 – Qué producto ha sido el más vendido?**

```
SELECT p.Name As 'Product', COUNT(*) AS 'Total’ – aunque más correcto sería hacer
COUNT(DISTINCT soh.SalesOrderID)
FROM SalesLT.SalesOrderHeader AS soh
    INNER JOIN SalesLT.SalesOrderDetail AS shd ON soh.SalesOrderID = shd.SalesOrderID
    INNER JOIN SalesLT.Product AS p ON shd.ProductID = p.ProductID
GROUP BY p.Name
ORDER BY 2 DESC
```

- **Challenge 2 – Cuántas unidades han sido vendidas por categoría y producto? (ordenado por categoría y producto)**

```
SELECT  pc.Name AS 'Category', p.name AS Product, SUM(sod.OrderQty) AS 'Total Qty'
  FROM SalesLT.Product AS p
      INNER JOIN SalesLT.SalesOrderDetail AS sod ON p.ProductID = sod.ProductID
      INNER JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
  GROUP BY  pc.Name, p.Name
  ORDER BY 1, 2
```

Lab 04 | Soluciones

- **Challenge 3 – Cuántas unidades han sido vendidas por categoría, y por categoría-producto?**

```
SELECT pc.Name AS 'Category',  p.Name As 'Product', SUM(shd.OrderQty) AS 'Total Qty'
FROM SalesLT.SalesOrderHeader AS soh
    INNER JOIN SalesLT.SalesOrderDetail AS shd ON soh.SalesOrderID = shd.SalesOrderID
    INNER JOIN SalesLT.Product AS p ON shd.ProductID = p.ProductID
    INNER JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
GROUP BY GROUPING SETS ((pc.Name), (pc.Name, p.Name))
ORDER BY 1, 2
```

- **Challenge 4 (B O N U S) – De la consulta anterior, descarta los grupos que tengan menos de 8 tickets de venta**

```
SELECT pc.Name AS 'Category',  p.Name As 'Product', SUM(shd.OrderQty) AS 'Total Qty'
FROM SalesLT.SalesOrderHeader AS soh
    INNER JOIN SalesLT.SalesOrderDetail AS shd ON soh.SalesOrderID = shd.SalesOrderID
    INNER JOIN SalesLT.Product AS p ON shd.ProductID = p.ProductID
    INNER JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
GROUP BY GROUPING SETS ((pc.Name), (pc.Name, p.Name))
HAVING COUNT(DISTINCT soh.SalesOrderID) >= 8
ORDER BY 1, 2
```

SELECT | aggregate window functions

- Hemos visto la potencia de `GROUP BY` combinado con funciones de agregado para obtener insights de los datos.
- El problema es que **reduce** el número de registros devueltos, no tenemos forma (fácil) de incluir algún dato que no esté en la cláusula `GROUP BY`.
- Pero existe también la posibilidad de usar **funciones de ventana**, las cuales también operan sobre grupos de registros, pero **no reducen** el conjunto de resultados final.

OVER

Realiza un cálculo sobre un conjunto de datos sin limitar los resultados

OVER ([<partition_by_clause>] [<order_by_clause>] [<range_clause>])

```
SELECT c1, c2, COUNT(*) OVER (PARTITION BY c1) 'total c1' FROM dbo.myTable;
```

SELECT | aggregate window functions

Tenemos por tanto las dos opciones: tener en el resultado los totales por grupo, pero también **el detalle y un cálculo** según la agrupación que nos interese.

`SUM ([ALL] <expression>) OVER ([ <partition_by_clause>] <order_by_clause>)`	Calcula la suma de las particiones encontradas
`COUNT ([ALL] { <expression> | * }) OVER ([ <partition_by_clause>])`	Calcula el número de elementos encontrados en la partición
`MAX ([ALL] <expression>) OVER (<partition_by_clause> [ <order_by_clause>])`	Calcula el valor máximo de las particiones encontradas
`MIN ([ALL] <expression>) OVER ([<partition_by_clause>] <order_by_clause>)`	Calcula el valor mínimo de las particiones encontradas
`AVG ([ALL | DISTINCT] <expression>) OVER ([<partition_by_clause>] <order_by_clause>)`	Calcula la media de las particiones encontradas
`VAR ([ALL | DISTINCT] <expression>) OVER ([<partition_by_clause>] <order_by_clause>)`	Calcula la varianza de las particiones encontradas
`STDEV ([ALL | DISTINCT] <expression>) OVER ([<partition_by_clause>] <order_by_clause>)`	Calcula la desviación estándar de las particiones encontradas

SELECT | ranking window functions

- La cláusula OVER nos abre un **mundo de posibilidades** nuevos dentro de SQL, porque junto con ella aparecen otras funciones no tan conocidas.
- Un tipo de estas nuevas funciones son las de **ranking.**

ROW_NUMBER() OVER ([PARTITION BY <partition_by_clause>] ORDER BY <order_by_clause>)	Devuelve un número secuencial para cada una de las filas dentro de la partición especificada
RANK() OVER ([PARTITION BY <partition_by_clause>] ORDER BY <order_by_clause>)	Es igual a ROW_NUMBER, pero RANK devuelve el mismo número en caso de ser el mismo valor, saltando el número para el siguiente registro.
DENSE_RANK() OVER ([PARTITION BY <partition_by_clause>] ORDER BY <order_by_clause>)	Es igual a RANK, pero sin saltos de número en caso de empate.
NTILE(<integer_expression>) OVER ([PARTITION BY <partition_by_clause>] ORDER BY <order_by_clause>)	Distribuye las filas en una partición ordenada en el número especificado de grupos

SELECT | analytical window functions

- Otras funciones de ventana muy útiles son las **analíticas**, las cuales calculan un **valor agregado** basado en un grupo de filas.
- La diferencia con las que conocemos es que pueden devolver **varias filas** dentro de cada grupo.
- Se usan en escenarios de moving average, running totals, porcentajes... dentro de un grupo

LAG(<scalar_expression> [, <offset_expression>] [, <default_expression>]) OVER ([<partition_by_clause>] <order_by_clause>)	Accede al valor de una fila previa del conjunto de resultados dentro de la partición especificada
LEAD(<scalar_expression> [, <offset_expression>] [, <default_expression>]) OVER ([<partition_by_clause>] <order_by_clause>)	Accede al valor de una fila subsiguiente del conjunto de resultados dentro de la partición especificada
FIRST_VALUE(<scalar_expression>) OVER ([<partition_by_clause>] <order_by_clause> [rows_range_clause])	Devuelve el primer valor de un conjunto ordenado de valores
LAST_VALUE(<scalar_expression>) OVER ([<partition_by_clause>] <order_by_clause> [rows_range_clause])	Devuelve el último valor de un conjunto ordenado de valores

**** Existen más funciones, pero son para escenarios más específicos***

Lab 05

WINDOW FUNCTIONS

Lab 05

- Conectamos a nuestra base de datos con Azure Data Studio

Server	`sql<iniciales del alumno>123.database.windows.net`
Authentication type	`SQL Login`
User name	`azureuser`
Password	`P@$$w0rd123`
Database	`AdventureWorks`

- Vamos a comprobar los resultados que nos devuelven algunas funciones de ventana

Lab 05

- **Challenge 1 – Revisa la salida de la siguiente instrucción**
    - Fíjate en los resultados para entender mejor la potencia de estas funciones

```
SELECT p.ProductID, pc.Name AS 'Category', p.Name AS 'Product', p.[Size]

    , ROW_NUMBER() OVER (PARTITION BY p.ProductCategoryID ORDER BY p.Size) AS 'Row Number Per Category & Size'
    , RANK() OVER (PARTITION BY p.ProductCategoryID ORDER BY p.Size) AS 'Rank Per Category & Size'
    , DENSE_RANK() OVER (PARTITION BY p.ProductCategoryID ORDER BY p.Size) AS 'Dense Rank Per Category & Size'
    , NTILE(2) OVER (PARTITION BY p.ProductCategoryID ORDER BY p.Name) AS 'NTile Per Category & Name'

    , SUM(p.StandardCost) OVER() AS 'Standard Cost Grand Total'
    , SUM(p.StandardCost) OVER(PARTITION BY p.ProductCategoryID) AS 'Standard Cost Per Category'

    , LAG(p.Name, 1, '-- NOT FOUND --') OVER(PARTITION BY p.ProductCategoryID ORDER BY p.Name) AS 'Previous Product Per Category'
    , LEAD(p.Name, 1, '-- NOT FOUND --') OVER(PARTITION BY p.ProductCategoryID ORDER BY p.Name) AS 'Next Product Per Category'

    , FIRST_VALUE(p.Name) OVER(PARTITION BY p.ProductCategoryID ORDER BY p.Name) AS 'First Product Per Category'
    , LAST_VALUE(p.Name) OVER(PARTITION BY p.ProductCategoryID ORDER BY p.Name) AS 'Last Product Per Category'
FROM SalesLT.Product AS p
    INNER JOIN SalesLT.ProductCategory AS pc ON p.ProductCategoryID = pc.ProductCategoryID
ORDER BY pc.Name, p.Name
```

SQL | Summary

#1: Nos hemos iniciado en el mundo informacional.

#2: Hemos conocido cómo calcular agregados.

#3: Hemos visto la versatilidad de las funciones de ventana.

Elementos procedurales

Elementos Procedurales | Definición

- Existe la posibilidad de agrupar comandos SQL en un único objeto, incluyendo además otros elementos como operadores, variables, flujo de control, etc.
- Esto nos aporta una serie de ventajas como
  - **Productividad**, al poder reutilizar el código en distintas partes de la aplicación, sin necesidad de reescribirlo constantemente
  - **Rendimiento,** se producen menos llamadas entre cliente y servidor, además de encontrarse pre-compilados
  - **Seguridad,** proporcionando una interfaz de acceso más controlada sin necesidad de tener que dar permisos sobre los objetos subyacentes
- Todos los motores de base de datos los ofrecen, aunque es aquí donde más diferencias se pueden encontrar entre cada uno de ellos.
- Y aunque normalmente se usa SQL como lenguaje, en realidad hay extensiones que permiten usar otros como .NET, Java, Javascript, Python, etc.

Elementos Procedurales | Definición

- Normalmente son ofrecidos en dos sabores: **procedimientos almacenados** (sprocs) y **funciones definidas de usuario** (UDF)

Procedimiento Almacenado	Función Definida de Usuario
Puede devolver o no un valor (o un dataset)	Siempre debe devolver un valor
Puede usar DML, DQL y algunas DDL	Solo puede usar DQL
Se invocan de forma explícita	Se embeben dentro de instrucciones SQL
Puede incluir llamadas a otras UDF y Sprocs	No puede llamar a un procedimiento almacenado
Pueden usarse bloques de control, tablas temporales, transacciones, etc.	No permite hacer uso de estas funcionalidades
Puede tener parámetros de entrada y salida	Solo puede tener parámetros de entrada

Elementos Procedurales | CREATE

STORE PROCEDURE

Crea o modifica un procedimiento almacenado en SQL Server

```
CREATE [ OR ALTER ] { PROC | PROCEDURE }
    [schema_name.] procedure_name [ ; number ]
    [ { @parameter_name [ type_schema_name. ] data_type }
        [ VARYING ] [ NULL ] [ = default ] [ OUT | OUTPUT | [READONLY]
    ] [ ,...n ]
[ WITH <procedure_option> [ ,...n ] ]
[ FOR REPLICATION ]
AS { [ BEGIN ] sql_statement [;] [ ...n ] [ END ] }
[;]
```

```
CREATE PROCEDURE dbo.miFirstSProc AS
   PRINT 'Hello World from a Sproc';
```

USER DEFINED FUNCTION

Crea o modifica una función de usuario en SQL Server

```
CREATE [ OR ALTER ] FUNCTION [ schema_name. ] function_name
( [ { @parameter_name [ AS ][ type_schema_name. ] parameter_data_type
[ NULL ]
 [ = default ] [ READONLY ] }
    [ ,...n ]
  ]
)
RETURNS return_data_type
    [ WITH <function_option> [ ,...n ] ]
    [ AS ]
    BEGIN
        function_body
        RETURN scalar_expression
    END
[ ; ]
```

```
CREATE FUNCTION dbo.miFirstUDF (@number INT) RETURNS VARCHAR(4) AS
BEGIN
   DECLARE @r VARCHAR(4);
   SET @r = CASE WHEN @number % 2 = 0 THEN 'EVEN' ELSE 'ODD' END;
   RETURN @r;
END;
```

Lab 06

STORE PROCEDURE

Lab 06

- Conectamos a nuestra base de datos con Azure Data Studio

Server	`sql<iniciales del alumno>123.database.windows.net`
Authentication type	`SQL Login`
User name	`azureuser`
Password	`P@$$w0rd123`
Database	`AdventureWorks`

- Vamos a revisar el código de un procedimiento almacenado y su comportamiento

Lab 06

- Abre el archivo *Lab06 - sproc create.sql*, revisa su código y ejecútalo para crear el procedimiento almacenado.
- El archivo *Lab06 - sproc call.sql* permite comprobar lo que hace con un cliente de ejemplo.

SQL | Summary

#1: Nos hemos iniciado en el mundo de los elementos procedurales en SQL.

#2: Hemos visto las diferencias entre los procedimientos almacenados y las funciones definidas de usuario.

#3: Hemos visto un pequeño ejemplo para empezar a familiarizarnos.

Escenarios habituales

Escenarios | Habituales

- Partiendo de la base de datos en Azure que hemos creado en laboratorios anteriores (AdventureWorksLT), vamos a ver soluciones a problemas comunes.

- Estas soluciones representan solo **una forma de hacer las cosas**. Es posible encontrar otras devuelvan los mismos resultados, la óptima dependerá en muchas ocasiones del motor usado y el diseño físico del modelo.

- En cualquier caso, permiten mostrar un planteamiento de cómo abordar este tipo de problemas y cómo poner en práctica lo aprendido hasta ahora.

Valores únicos | identificar duplicados

Por ejemplo, partiendo de

```
SELECT OrderQty, ProductId, UnitPrice FROM [SalesLT].[SalesOrderDetail]
```

¿Cómo identificar los registros duplicados en mi dataset?

En este caso, simplemente añadiendo a la consulta un filtro para cada uno de los grupos identificados (HAVING)

```
[…]
GROUP BY OrderQty, ProductId, UnitPrice
HAVING COUNT(*)>1
```

OrderQty	ProductId	UnitPrice
1	836	356,898
1	822	356,898
1	907	63,90
4	905	218,454
2	983	461,694
6	988	112,998
2	748	818,70
1	990	323,994
1	926	149,874
1	743	809,76
4	782	1376,994

La imagen no muestra la salida completa

Valores únicos | seleccionar solo registros únicos

¿Cómo identificar esos registros únicos y a la vez tener todo el contenido del registro?

(Partiendo de la consulta inicial anterior...)

```
; WITH identificandoRegistros AS
(
    SELECT *
    , ROW_NUMBER() OVER (PARTITION BY OrderQty, ProductId, UnitPrice ORDER BY SalesOrderID ) AS IdPorGrupo
    FROM [SalesLT].[SalesOrderDetail]
)
, registrosUnicos AS
(
    SELECT
[SalesOrderID],[SalesOrderDetailID],[OrderQty],[ProductID],[UnitPrice],[UnitPriceDiscount],[LineTotal],[rowg
uid],[ModifiedDate]
    FROM identificandoRegistros
    WHERE IdPorGrupo = 1
)
SELECT * FROM registrosUnicos
```

Esta posible opción, aunque más *verbosa* por las CTE, es más clara de entender.

Aparecen más registros que la consulta anterior porque se incluyen también los que no se repetían.

Valores únicos | n registros por grupo

¿Cómo devolver un número determinado de elementos por grupo?

```
; WITH identificandoRegistros AS
(
    SELECT *
    , ROW_NUMBER() OVER (PARTITION BY OrderQty, ProductId, UnitPrice ORDER BY SalesOrderID ) AS IdPorGrupo
    FROM [SalesLT].[SalesOrderDetail]
)
, registrosUnicos AS
(
    SELECT
[SalesOrderID],[SalesOrderDetailID],[OrderQty],[ProductID],[UnitPrice],[UnitPriceDiscount],[LineTotal],[rowg
uid],[ModifiedDate]
    FROM identificandoRegistros
    WHERE IdPorGrupo <= 3
)
SELECT * FROM registrosUnicos
```

Nos sirve la instrucción anterior, solo hace falta cambiar el filtro por el campo *IdPorGrupo*.

Acumulados | reiniciando valores

¿Cómo calcular un running aggregate y devolver además el registro completo?

Nuevamente, las funciones de ventana nos son de gran ayuda

```
SELECT od.[SalesOrderID], od.[SalesOrderDetailID], od.[OrderQty], od.[ProductID], od.[UnitPrice], od.[UnitPriceDiscount], od.[LineTotal]
 , ROW_NUMBER() OVER (PARTITION BY od.SalesOrderID ORDER BY od.SalesOrderID, od.SalesOrderDetailID) AS IdLinea
 , SUM(od.LineTotal) OVER (PARTITION BY od.SalesOrderID ORDER BY od.SalesOrderID, od.SalesOrderDetailID) AS TotalVentasAcumuladoPorPedido
FROM SalesLT.SalesOrderDetail as od
ORDER BY od.[SalesOrderID], od.[SalesOrderDetailID]
```

Cálculos | entre valores

¿Cómo calcular la diferencia entre el valor de una columna y un registro anterior?

Simulamos que el identificador del pedido es el día de venta (las fechas del pedido en la base de datos de ejemplo es siempre el mismo, de ahí que ese escenario no nos vale para probar)

Y para encontrar una solución, otra función de ventana nos facilita el trabajo enormemente

```
SELECT ProductID, SalesOrderID
, SalesOrderID - LAG(SalesOrderID, 1, NULL) OVER(PARTITION BY ProductID ORDER BY SalesOrderID) ["Días" de diferencia entre pedidos]
FROM SalesLT.SalesOrderDetail
ORDER BY ProductID, SalesOrderID
```

Categorización | de ventas

¿Cómo agrupar un producto en base a la frecuencia de sus ventas?

```
; WITH clasificacion AS
(
    SELECT ProductID
        , CASE
            WHEN SalesOrderID - LAG(SalesOrderID, 1, NULL) OVER(PARTITION BY ProductID ORDER BY SalesOrderID) BETWEEN 1 AND 5 THEN 'Frecuente'
            WHEN SalesOrderID - LAG(SalesOrderID, 1, NULL) OVER(PARTITION BY ProductID ORDER BY SalesOrderID) BETWEEN 5 AND 10 THEN 'Poco Frecuente'
            ELSE 'Ocasional'
        END AS [Ventas del Producto]
    FROM SalesLT.SalesOrderDetail
)
, conteo AS
(
    SELECT ProductID, [Ventas del Producto], COUNT(*) AS [Nº Repeticiones]
    FROM clasificacion
    GROUP BY ProductID, [Ventas del Producto]
)
, maximo AS
(
    SELECT ProductID, MAX([Nº Repeticiones]) AS [Nº Repeticiones]
    FROM conteo
    GROUP BY ProductID
)
SELECT p.Name AS Producto, conteo.[Nº Repeticiones]
FROM conteo
    INNER JOIN maximo
        ON conteo.ProductID = maximo.ProductID AND conteo.[Nº Repeticiones] = maximo.[Nº Repeticiones]
    JOIN SalesLT.Product AS p
        ON conteo.ProductID = p.ProductID
ORDER BY Producto, [Nº Repeticiones] DESC, [Ventas del Producto]
```

Reutilizamos la función del ejemplo anterior de una forma un poco diferente, el resto de las CTE nos sirven para hacer cálculos intermedios que nos permitan filtrar los resultados de la forma que queremos

Clientes | sin compras

¿Cómo identificar los clientes que no realizaron compras ?

Este es un escenario conocido, puesto que en la explicación de los diferentes tipos de JOIN se mostraba cómo conseguir el resultado aquí buscado.

En esta ocasión mostramos diferentes alternativas para obtener lo que queremos

```
-- opc 1
SELECT *
FROM SalesLT.Customer AS c
WHERE c.CustomerID NOT IN (SELECT CustomerID FROM SalesLT.SalesOrderHeader )

-- opc 2
SELECT *
FROM SalesLT.Customer AS c
WHERE NOT EXISTS (SELECT 1 FROM SalesLT.SalesOrderHeader WHERE CustomerID = c.CustomerID)

-- opc 3
SELECT c.*
FROM SalesLT.Customer AS c
    LEFT JOIN SalesLT.SalesOrderHeader AS h on h.CustomerID = c.CustomerID
WHERE h.CustomerID IS NULL
```

Basket | analysis

Identificar los productos que se suelen comprar juntos usualmente

Construimos los pares de valores de los productos vendidos en los pedidos y contamos su frecuencia. Luego lo que hacemos es filtrar por aquellos que se repiten más que la mediana (la media en este caso no es una referencia válida)

```
; WITH f AS
(
    SELECT CONCAT(a.ProductID, ' - ', b.ProductID ) AS pares, COUNT(*) AS Frecuencia
    FROM SalesLT.SalesOrderDetail AS a
        INNER JOIN SalesLT.SalesOrderDetail AS b
            ON a.SalesOrderID = b.SalesOrderID
                AND a.ProductID < b.ProductID
    GROUP BY CONCAT(a.ProductID, ' - ', b.ProductID )
)
SELECT CONCAT('"',  prodA.Name, '" se ha comprado ', CAST(f.Frecuencia AS VARCHAR(5)), ' veces conjuntamente con "', prodB.Name,  '"') AS [Basket Analysis]
FROM f
    INNER JOIN SalesLT.Product AS prodA
        ON LEFT(f.pares, 3) = prodA.ProductID
    INNER JOIN SalesLT.Product AS prodB
        ON RIGHT(f.pares, 3) = prodB.ProductID
WHERE f.Frecuencia >= (SELECT (MAX(Frecuencia) + MIN(Frecuencia)) / 2 FROM f)
ORDER BY Frecuencia DESC
```

Comentarios finales

Recursos | Bibliografía

- **T-SQL Querying (Developer Reference)**, Itzik Ben-Gan, 2015
- **SQL Cookbook**, Anthony Molinaro, 2005
- **SQL: the complete reference**, Paul Weinberg, 2009
- **Learning SQL**, Alan Beaulieu, 2014
- **Data Analysis Using SQL and Excel**, Gordon S. Linoff, 2015
- **The Data Warehouse Toolkit**, Ralph Kimball, 2013

Fuente: https://www.amazon.com/Computer-Programmer-Funny-Joke-T-shirt/dp/B078V378PR

Recursos| Enlaces

Reglas formales de base de datos
https://en.wikipedia.org/wiki/Database_normalization
https://towardsdatascience.com/a-complete-guide-to-database-normalization-in-sql-6b16544deb0
https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description

Modelado de base de datos
https://community.idera.com/database-tools/blog/b/community_blog/posts/types-of-data-model-conceptual-logical-physical

Instrucciones DDL
Create Table (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/create-table-transact-sql?view=sql-server-ver15
Alter Table (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/alter-table-transact-sql?view=sql-server-ver15
Drop Table (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/drop-table-transact-sql?view=sql-server-ver15

Instrucciones DML
Insert (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/insert-transact-sql?view=sql-server-ver15
Update (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/update-transact-sql?view=sql-server-ver15
Delete (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/delete-transact-sql?view=sql-server-ver15
Merge (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/statements/merge-transact-sql?view=sql-server-ver15

Instrucciones DML
Select (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-transact-sql?view=sql-server-ver15
Joins (SQL Server):
https://docs.microsoft.com/en-us/sql/relational-databases/performance/joins?view=sql-server-ver15

Set Operators
Union (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-union-transact-sql?view=sql-server-ver15
Except & Intersect (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/language-elements/set-operators-except-and-intersect-transact-sql?view=sql-server-ver15

Order
Order by (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-order-by-clause-transact-sql?view=sql-server-ver15

Where
where (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/where-transact-sql?view=sql-server-ver15

Recursos| Enlaces

Subqueries
Subqueries (SQL Server):
https://docs.microsoft.com/en-us/sql/relational-databases/performance/subqueries?view=sql-server-ver15

Common Table Expressions
CTE (SQL Server): https://docs.microsoft.com/en-us/sql/t-sql/queries/with-common-table-expression-transact-sql?view=sql-server-ver15

Cloud Computing
Cloud Computing Intro (Azure):
https://azure.microsoft.com/en-us/overview/what-is-cloud-computing/

Data Warehouse
What is a DWH (AWS):
https://aws.amazon.com/data-warehouse/

Kimball vs Inmon
https://www.keboola.com/blog/kimball-vs-inmon

Modelado dimensional
Dimensional Modeling Tehcniques (Kimball):
https://www.kimballgroup.com/data-warehouse-business-intelligence-resources/kimball-techniques/dimensional-modeling-techniques/

Hechos y dimensiones
Dimension and facts in terms of data warehousing:
https://www.mozartdata.com/post/dimensions-and-facts-in-terms-of-data-warehousing

Group by
Group by (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-group-by-transact-sql?view=sql-server-ver15

Agregaciones
Aggregate Functions (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15

Having
Having (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/queries/select-having-transact-sql?view=sql-server-ver15

Over
Over (SQL Server): https://docs.microsoft.com/en-us/sql/t-sql/queries/select-over-clause-transact-sql?view=sql-server-ver15

Aggregate Window Functions
Aggregate Window Functions (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/functions/aggregate-functions-transact-sql?view=sql-server-ver15

Ranking Window Functions
Ranking Window Functions (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/functions/ranking-functions-transact-sql?view=sql-server-ver15

Analytical Window Functions
Analytical Window Functions (SQL Server):
https://docs.microsoft.com/en-us/sql/t-sql/functions/analytic-functions-transact-sql?view=sql-server-ver15

Procedimientos Almacenados
Stored Procedures (SQL Server):
https://learn.microsoft.com/en-us/sql/relational-databases/stored-procedures/stored-procedures-database-engine?view=sql-server-ver16

Recursos| Enlaces

SQL Cheat Sheet
SQL Basics: https://learnsql.com/blog/sql-basics-cheat-sheet/
SQL Joins: https://learnsql.com/blog/sql-join-cheat-sheet/
SQL Window Functions: https://learnsql.com/blog/sql-window-functions-cheat-sheet/
SQL Functions: https://learnsql.com/blog/standard-sql-functions-cheat-sheet/standard-sql-functions-cheat-sheet-a4.pdf

THANKS !