

M1967

OLAP, Diseño Multidimensional y Extensión OLAP-SQL

Diego García Saiz

Grupo de Ingeniería de Software y Tiempo Real
Departamento de Ingeniería Informática y Electrónica

Contenidos

- Introducción y contexto
 - Componentes de un sistema BI con DW y OLAP.
 - OLTP: On-line Transactional Processing
- OLAP: On-line Analytical Processing
 - Cubos OLAP.
 - Tecnologías OLAP.
 - ROLAP, data warehouse y data marts.
 - Resumen y conclusiones.
 - Extensión SQL-OLAP y ejemplos.

Introducción y contexto

Componentes de un sistema BI con DW y OLAP.

OLTP: On-line Transactional Processing

Componentes de un sistema BI con DW y OLAP

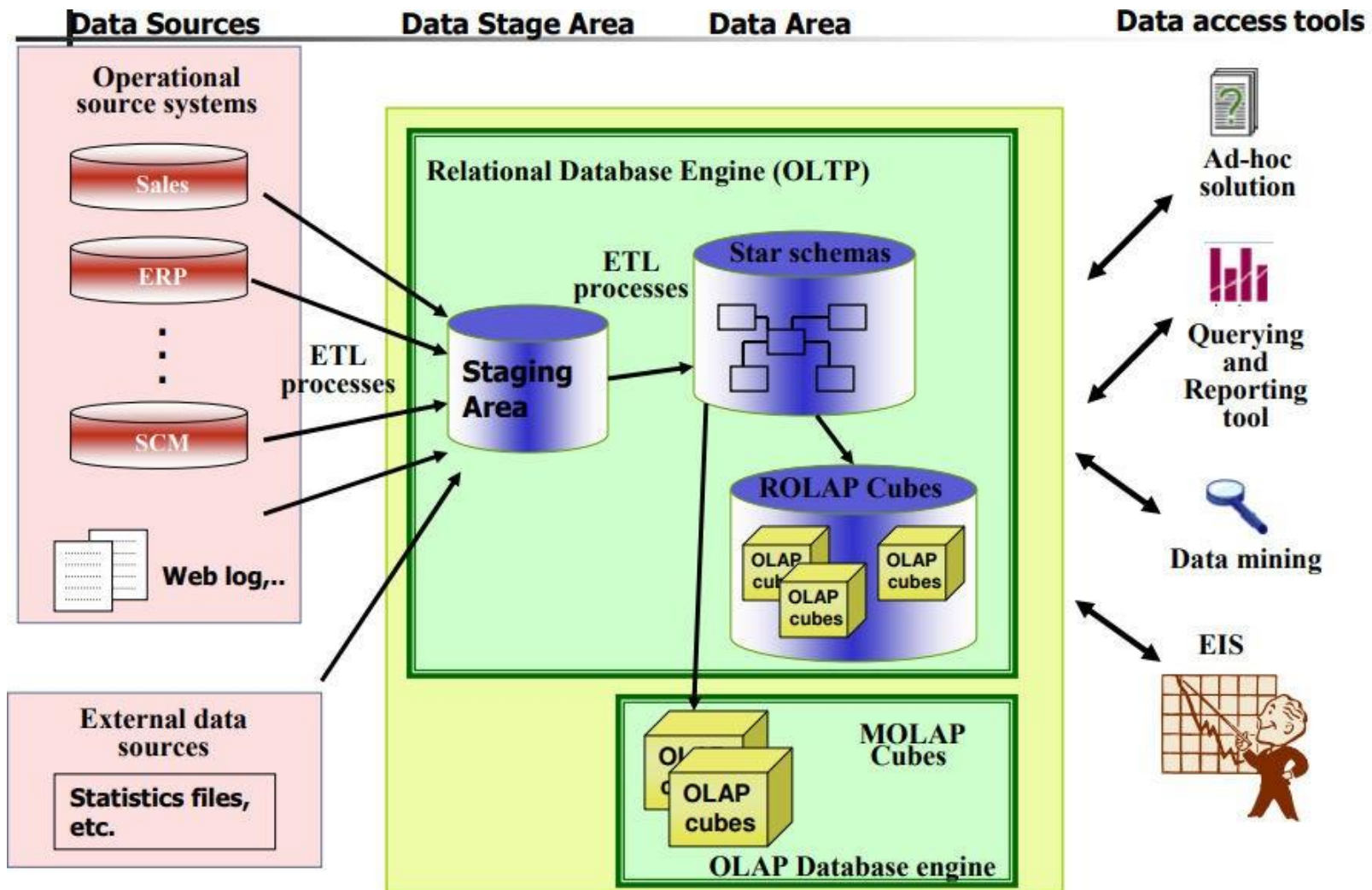


Imagen extraída de <http://docshare01.docshare.tips/files/24773/247735327.pdf>

Componentes de un sistema BI con DW y OLAP

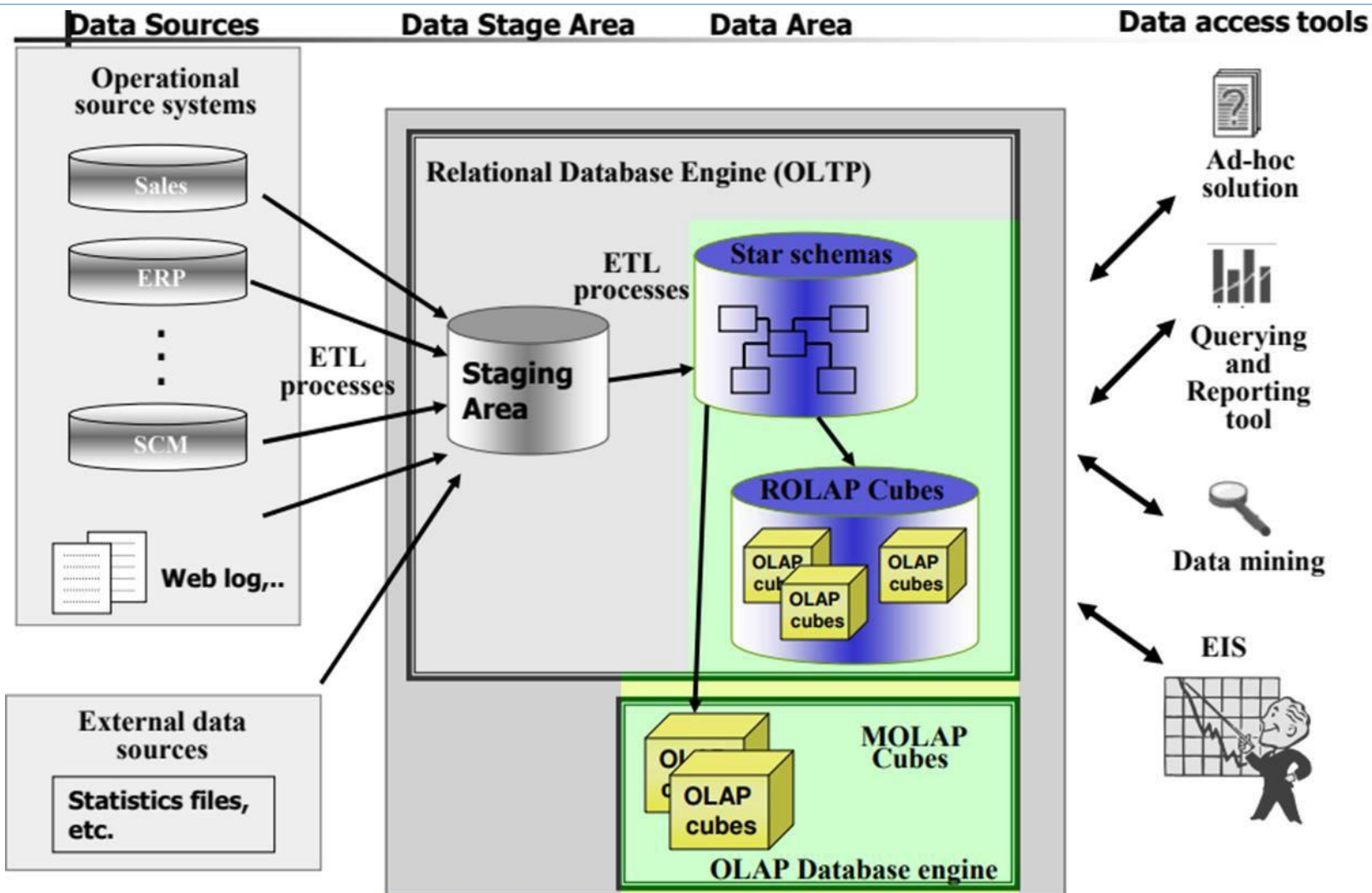


Imagen extraída de <http://docshare01.docshare.tips/files/24773/247735327.pdf>

Componentes de un sistema BI con DW y OLAP

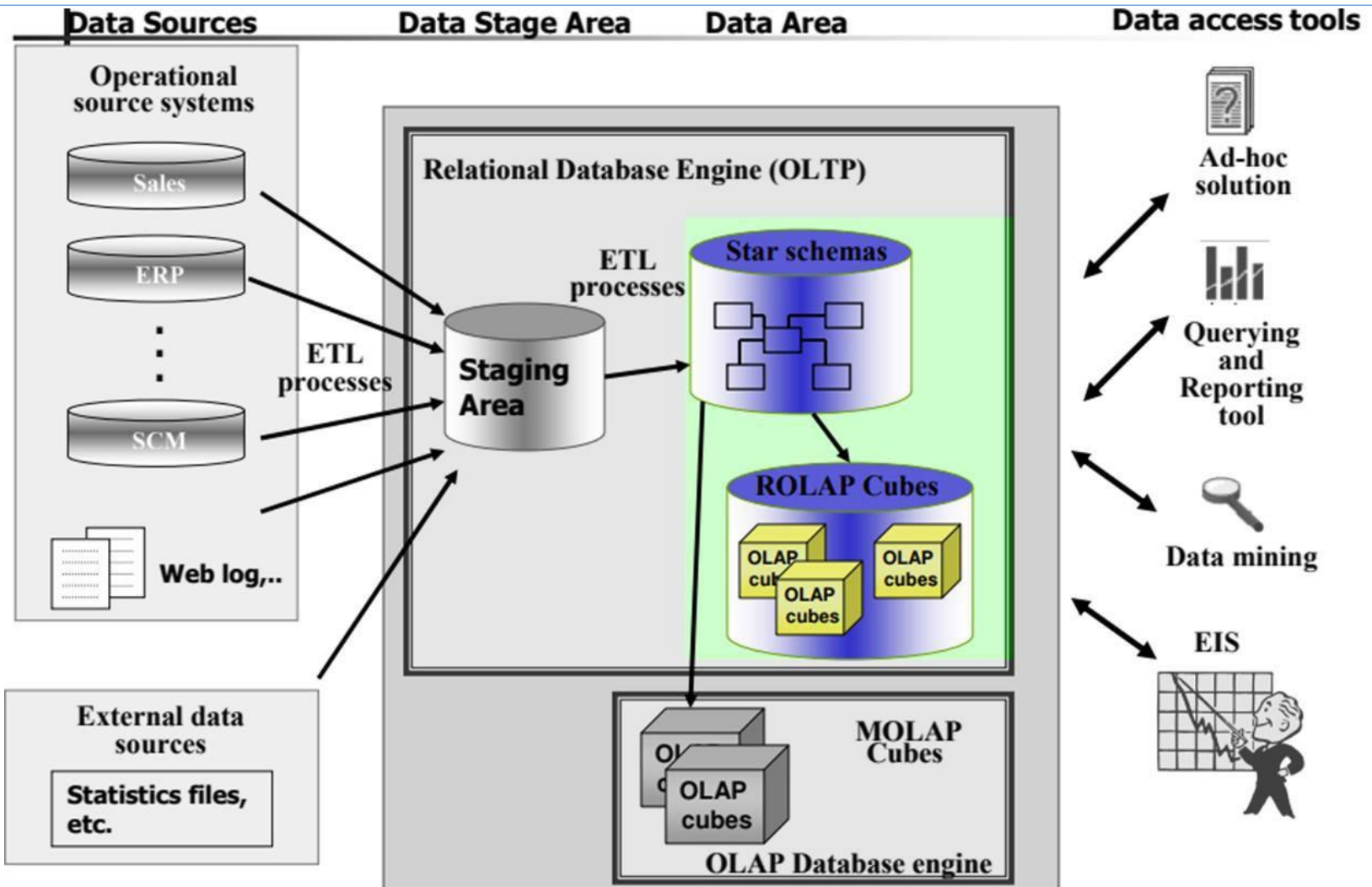


Imagen extraída de <http://docshare01.docshare.tips/files/24773/247735327.pdf>

Introducción y contexto

Componentes de un sistema BI con DW y OLAP

OLTP: On-line Transactional Processing

OLTP: On-line Transactional Processing

- Procesamiento de datos orientado a la transacción.
 - Soporte a los siguientes tipos de operaciones transaccionales de forma atómica:
 - INSERT: insertado de nuevos datos.
 - UPDATE: actualiza datos.
 - DELETE: elimina datos.
 - Además, SELECT para la consulta de datos.
 - Garantizar las transacciones implica garantizar ACID:
 - Atomicidad (Atomicity): “o todo, o nada”. El conjunto de operaciones de una transacción ha de ejecutarse por completo (commit). En caso de que una instrucción falle, se anulan TODAS (rollback).
 - Consistencia (Consistency): toda transacción que lleve a la base de datos a un nuevo estado (nuevos datos o actualizaciones) ha de garantizar que este nuevo estado sea valido, cumpliendo con todas las restricciones (claves, checks, disparadores, tipos de datos, etc.).
 - Aislamiento (Isolation): si se ejecutan diferentes transacciones de forma simultánea, el sistema de control de concurrencia ha de garantizar que dicha ejecución producirá el mismo efecto que en el caso de que se ejecutasen de forma secuencial.
 - Durabilidad (Durability): los cambios efectuados por una transacción han de ser persistentes en el tiempo.



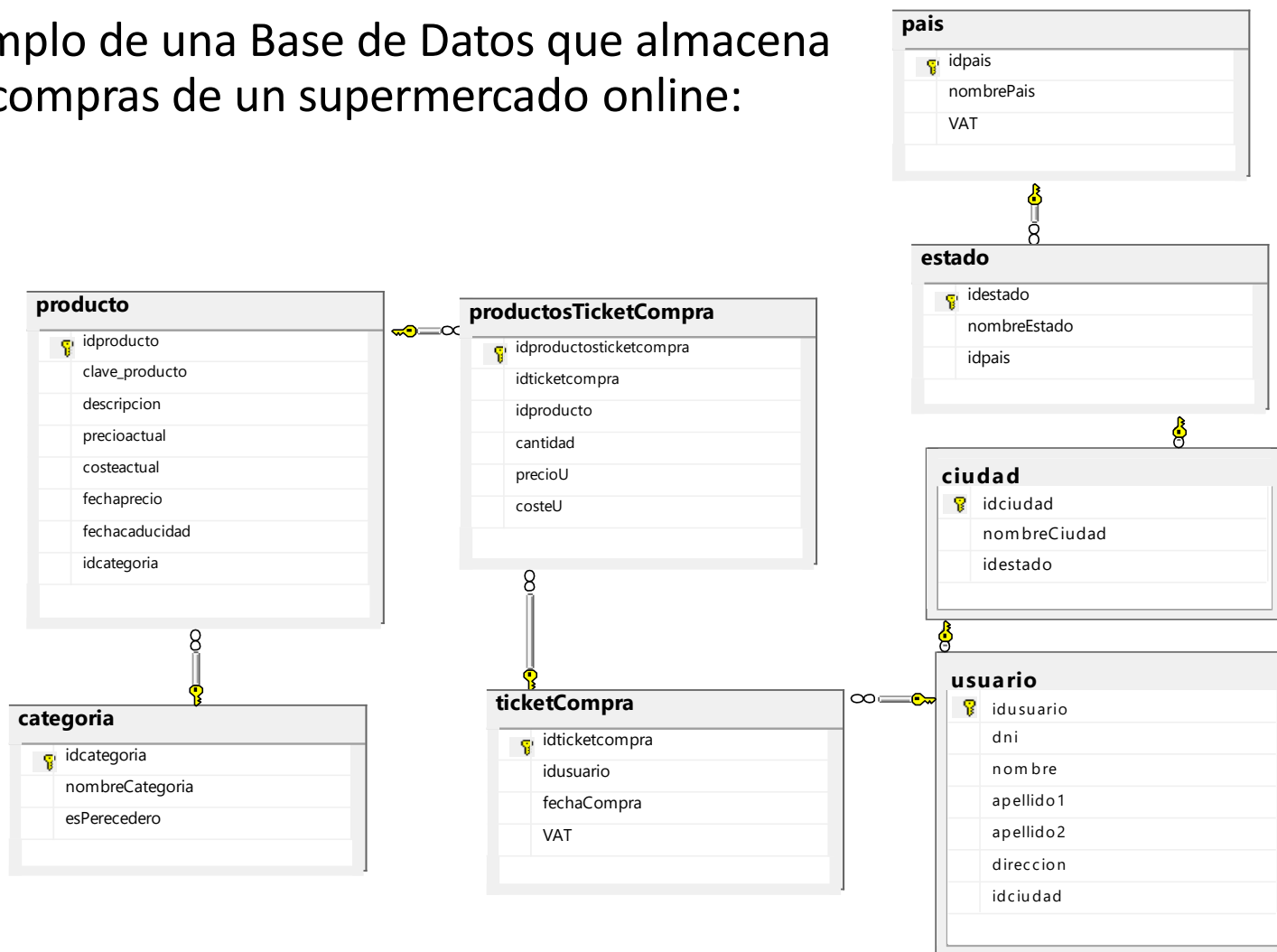
Bases de datos relacionales / SQL

OLTP: On-line Transactional Processing

- Almacena todos los datos del negocio al detalle.
 - Por ejemplo, en una base de datos de una tienda virtual, se almacenarían todos los datos al respecto de los productos actualmente en venta, los productos sin stock, los datos de todos los clientes, las compras de los clientes con todos sus detalles (productos comprados, precio, impuestos, fecha y hora, descuentos, etc.), las facturas generadas, y otros.
- Normalmente, al usar un gestor de bases de datos relacionales, se tendrán múltiples tablas para almacenar los diferentes datos.
 - Siguiendo el ejemplo anterior, se tendrá una tabla para los productos, otra para los clientes, otra para compras, otra para las líneas de compra, etc.
- Los sistemas OLTP suelen recibir gran número de operaciones transaccionales (INSERT, UPDATE, DELETE)
 - Están por tanto enfocados en optimizar este tipo de operaciones para que sean lo más rápidas posibles, al tiempo que garantizan la integridad de los datos ACID).

OLTP: Ejemplo de base de datos relacional

- Ejemplo de una Base de Datos que almacena las compras de un supermercado online:



OLTP: Ejemplo de base de datos relacional

- Imaginemos que sobre esta base de datos, deseamos realizar una consulta acerca del beneficio obtenido por año para los artículos de la categoría “alimentación” comprados por usuarios españoles:

```
SELECT STRFTIME('%Y',tc.fechaCompra) AS anio,  
SUM((ptc.precioU-ptc.costeU)*ptc.cantidad) AS beneficioAnual  
FROM ticketCompra tc INNER JOIN productosTicketCompra ptc  
    ON tc.idticketcompra=ptc.idticketcompra  
INNER JOIN usuario u ON u.idusuario = tc.idusuario  
INNER JOIN ciudad ci ON ci.idciudad = u.idciudad  
INNER JOIN estado es ON es.idestado=ci.idestado  
INNER JOIN pais pa ON es.idpais=pa.idpais  
INNER JOIN producto pr ON pr.idproducto=ptc.idproducto  
INNER JOIN categoria ca ON ca.idcategoria=pr.idcategoria  
WHERE ca.nombreCategoria = 'alimentación' AND pa.nombrePais='España'  
GROUP BY STRFTIME('%Y',tc.fechaCompra);
```

- ¡Siete JOINS!. Esta consulta puede ser tremendamente ineficiente.
- ¿Y si tuviésemos los datos almacenados en una estructura paralela, y agrupados convenientemente?

OLAP: On-line Analytical Processing

Cubos OLAP.

Tecnologías OLAP.

ROLAP, data warehouse y data marts.

Comparativa OLAP vs OLTP.

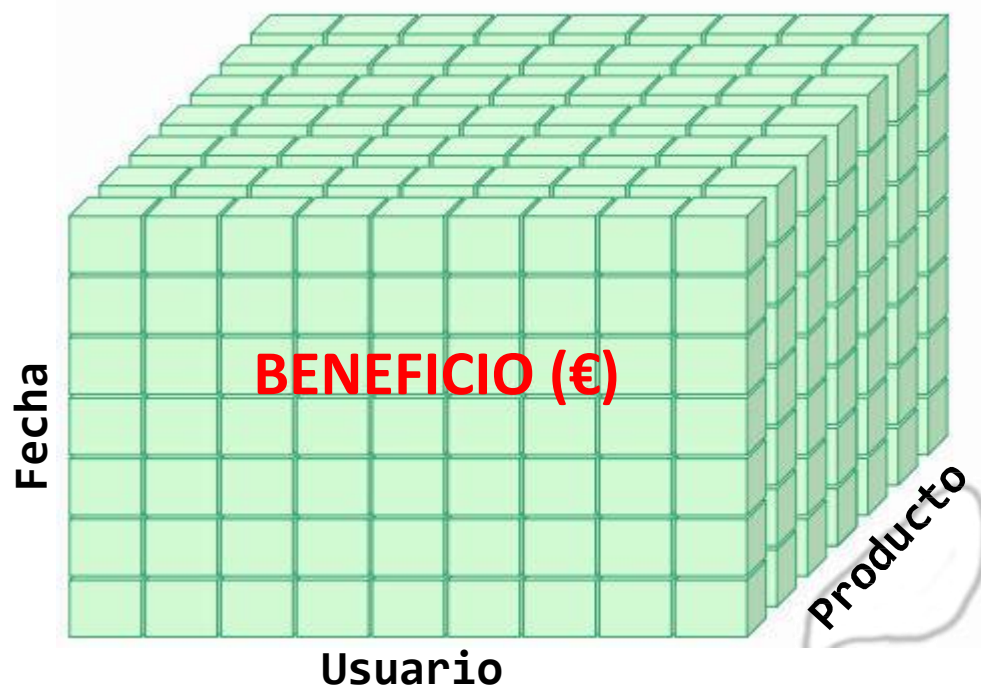
Extensión SQL-OLAP y ejemplos.

OLAP: On-line Analytical Processing

- En contraposición a OLTP, encontramos OLAP, acrónimo en castellano de Procesamiento Analítico en Línea.
- OLAP surge de la necesidad de dotar a las organizaciones de procesos alternativos al OLTP que permitiesen agilizar las consultas.
 - Como hemos visto, las bases de datos relacionales (OLTP) son más adecuadas para soportar gran cantidad de transacciones, pero están muy limitadas en cuanto a la optimización de consultas que comprendan grandes cantidades de datos y lecturas a varias tablas.
- Para dar un mejor soporte a las consultas, los datos se almacenan en estructuras multidimensionales, conocidas como cubos OLAP.

OLAP: ejemplo de cubo

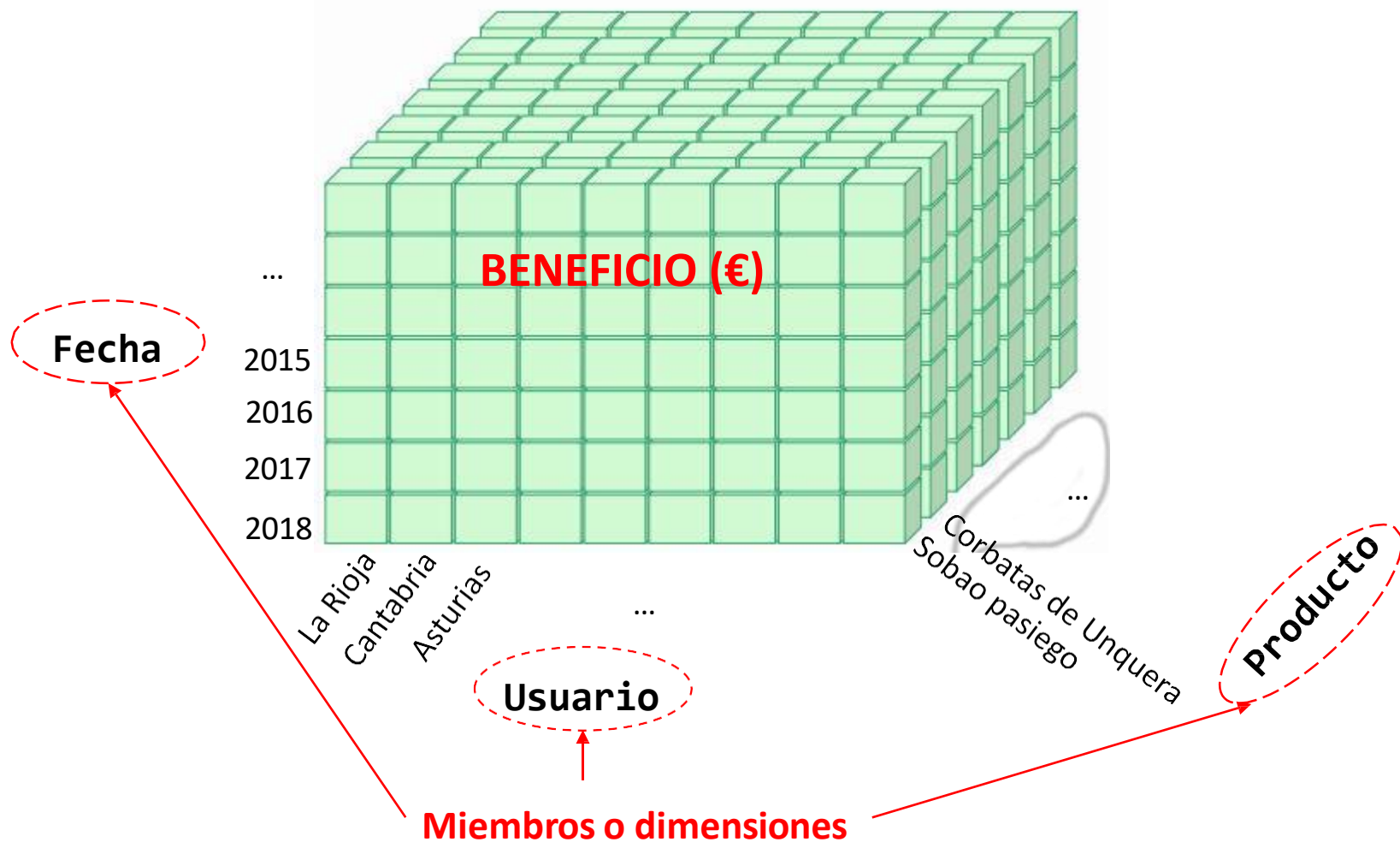
- La siguiente figura muestra un ejemplo de cubo OLAP con tres dimensiones: usuarios, fecha y producto



- Cada celda de la figura dentro del cubo representa un dato relativo a una compra de un producto, por parte de un usuario, y en una fecha determinada.
- Estos cubos contienen los datos necesarios para que el interesado, en este caso por ejemplo la empresa de supermercados, pueda hacer consultas concretas para un futuro análisis:
 - País del usuario.
 - Día, mes, año, hora y minutos de la compra.
 - Categoría del producto.
 - Beneficio obtenido por la venta del producto.
 - Otros...

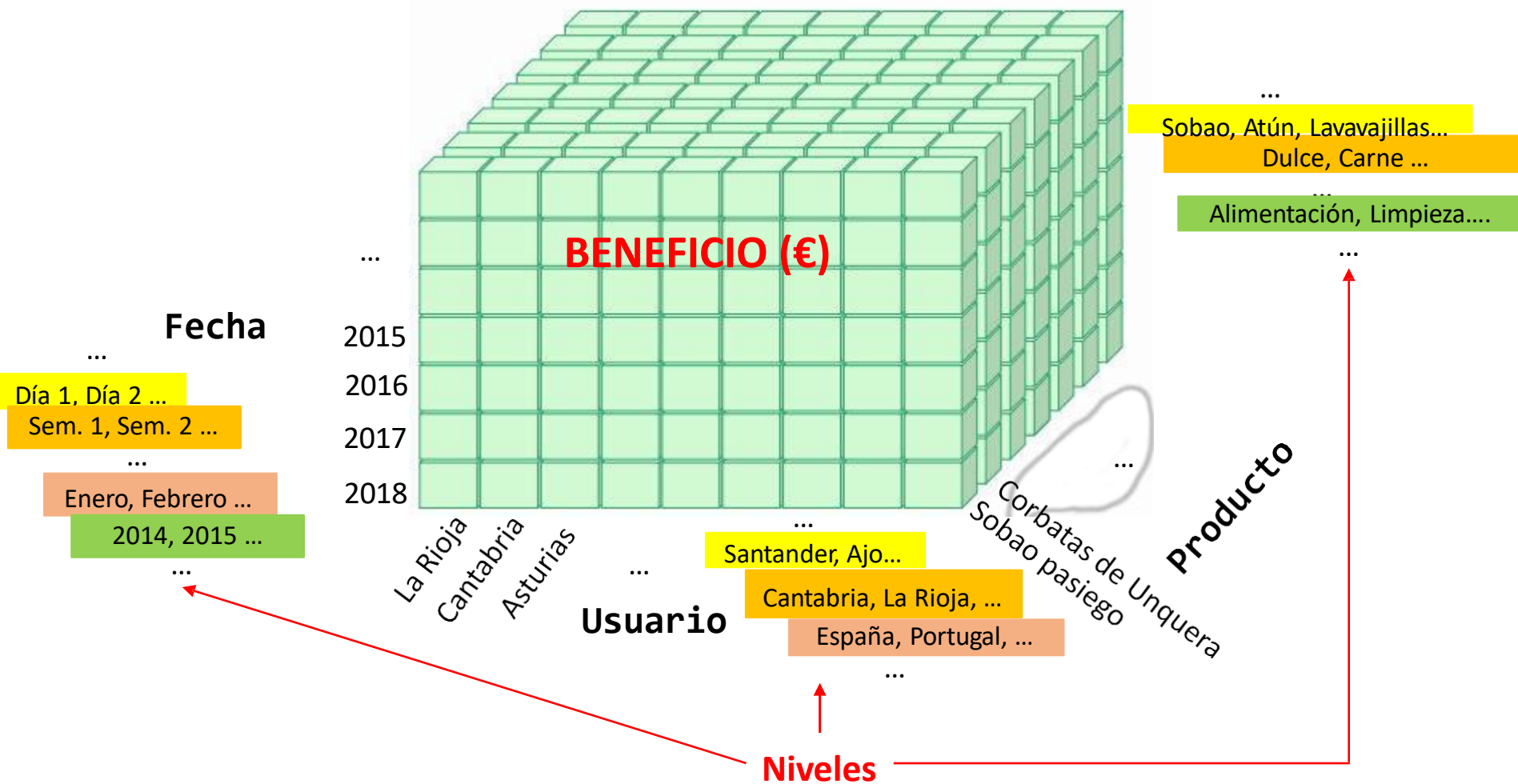
OLAP: elementos de un cubo

- Los elementos de un cubo son los siguientes:



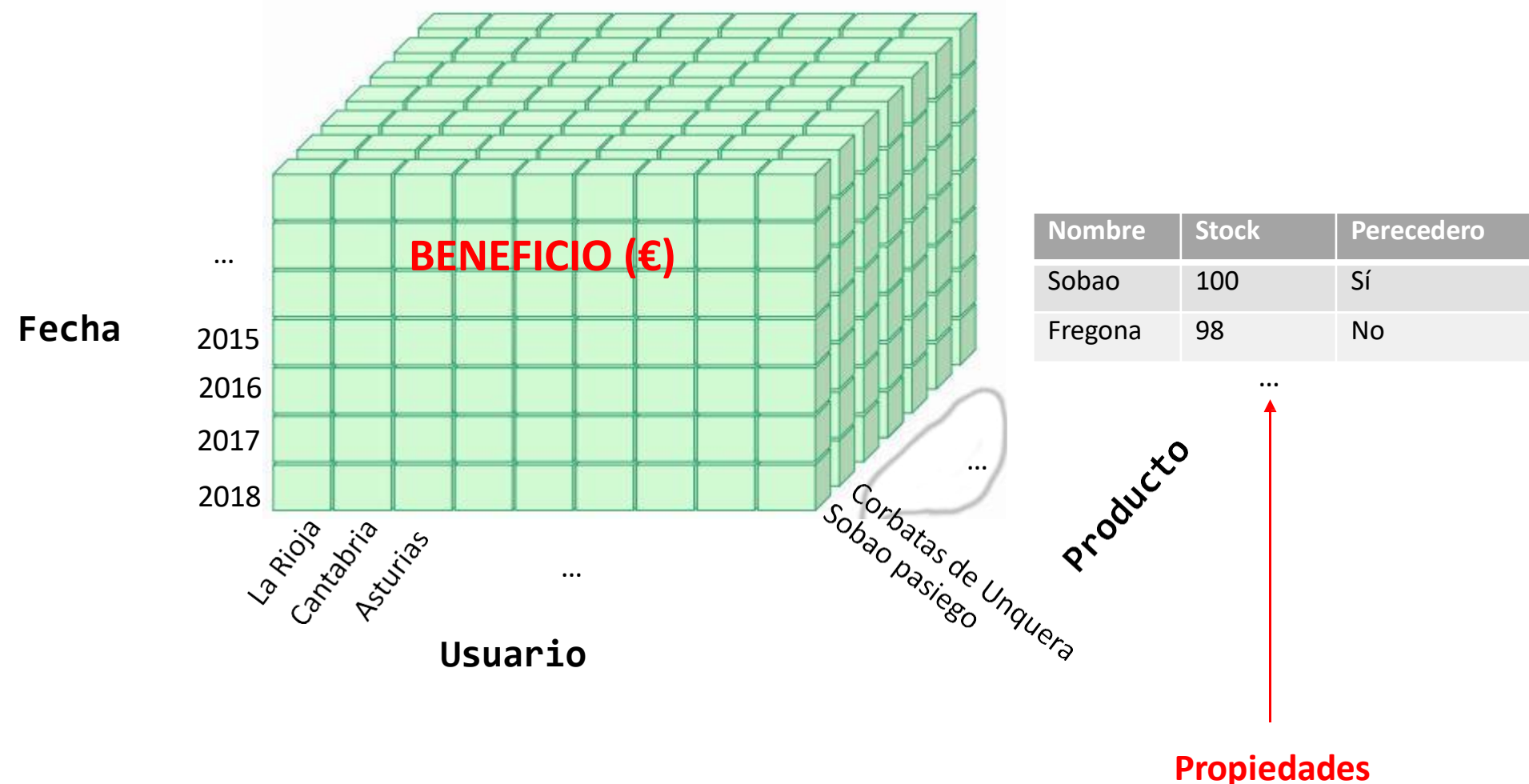
OLAP: elementos de un cubo

- Los elementos de un cubo son los siguientes:



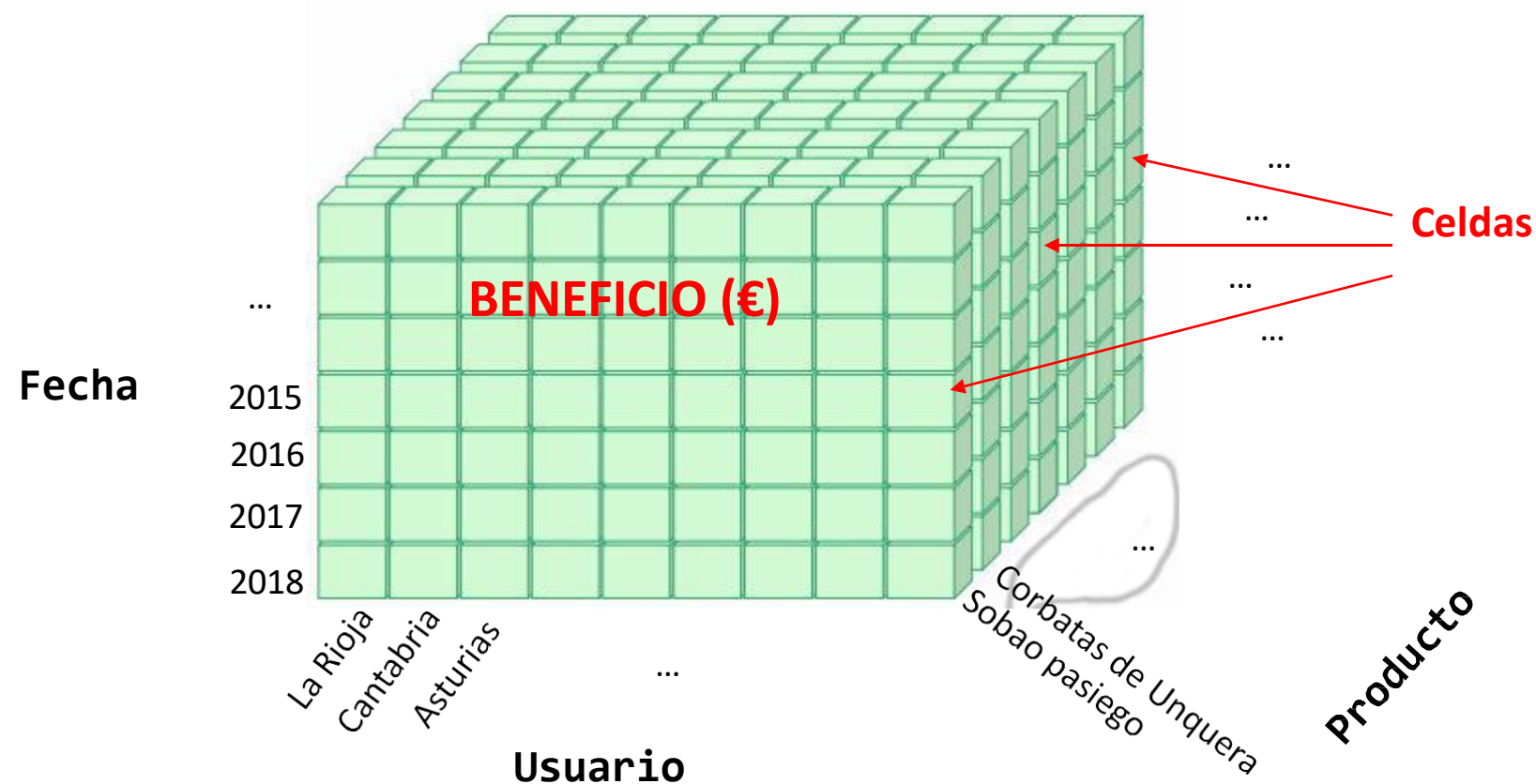
OLAP: elementos de un cubo

- Los elementos de un cubo son los siguientes:



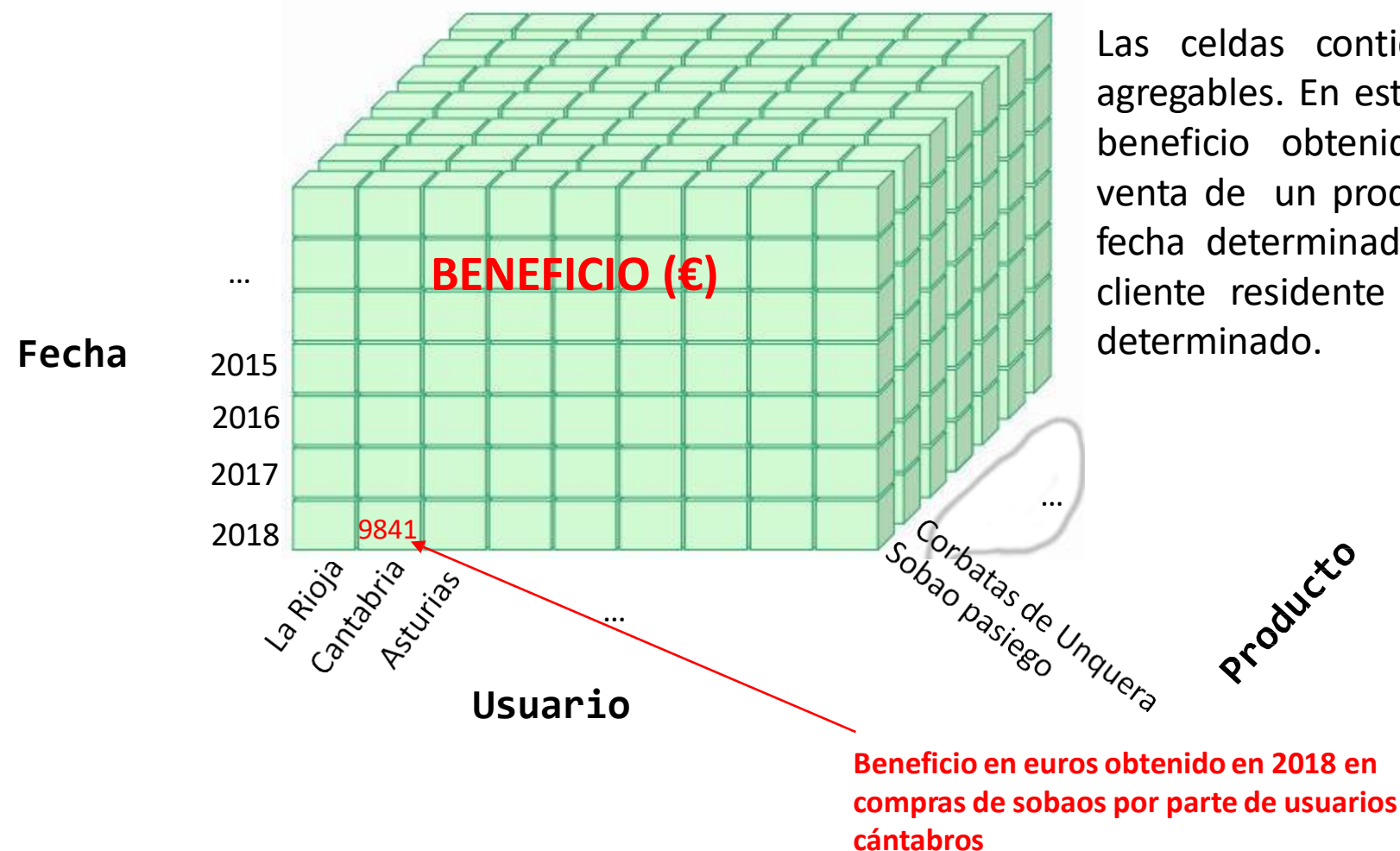
OLAP: elementos de un cubo

- Los elementos de un cubo son los siguientes:



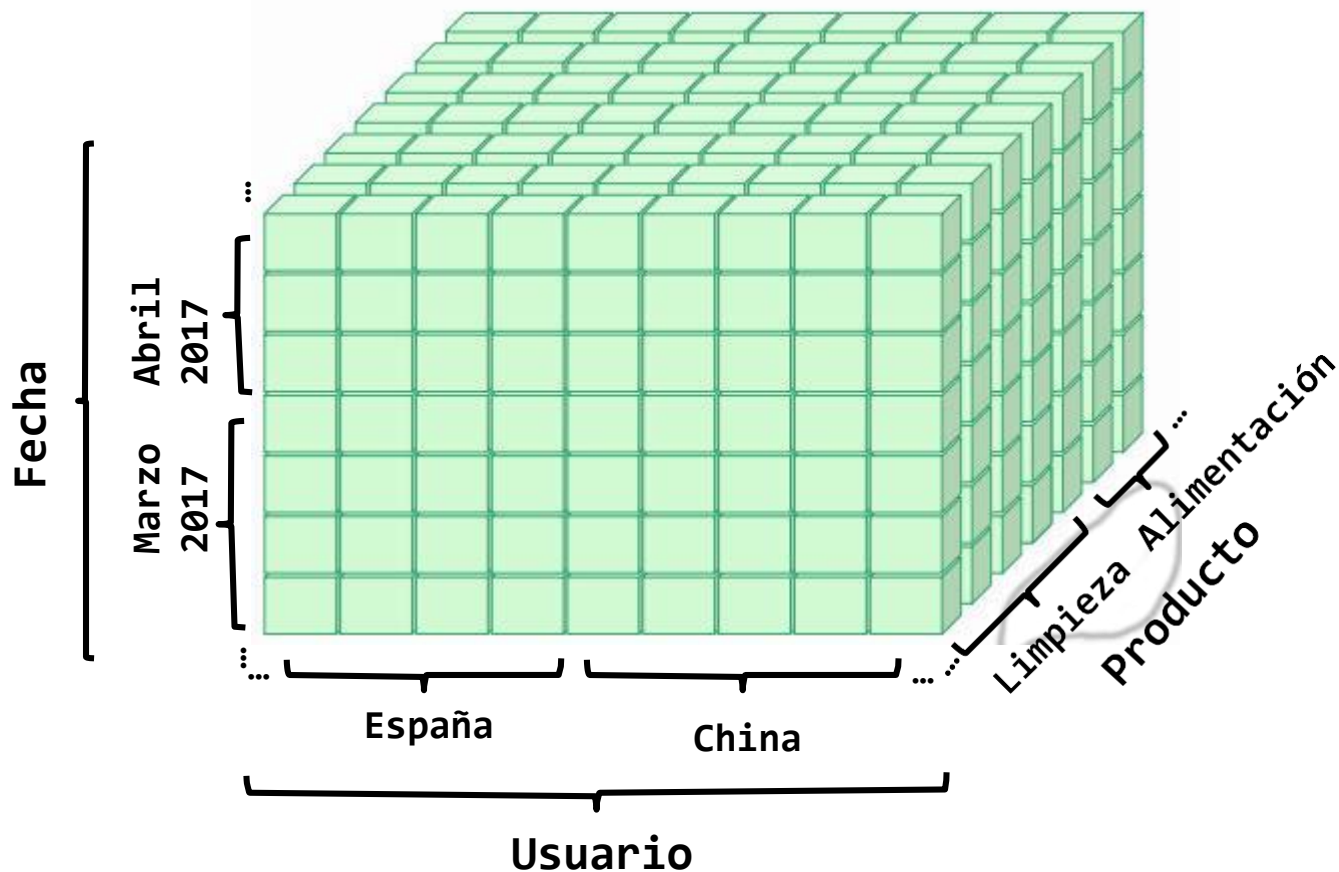
OLAP: elementos de un cubo

- Los elementos de un cubo son los siguientes:



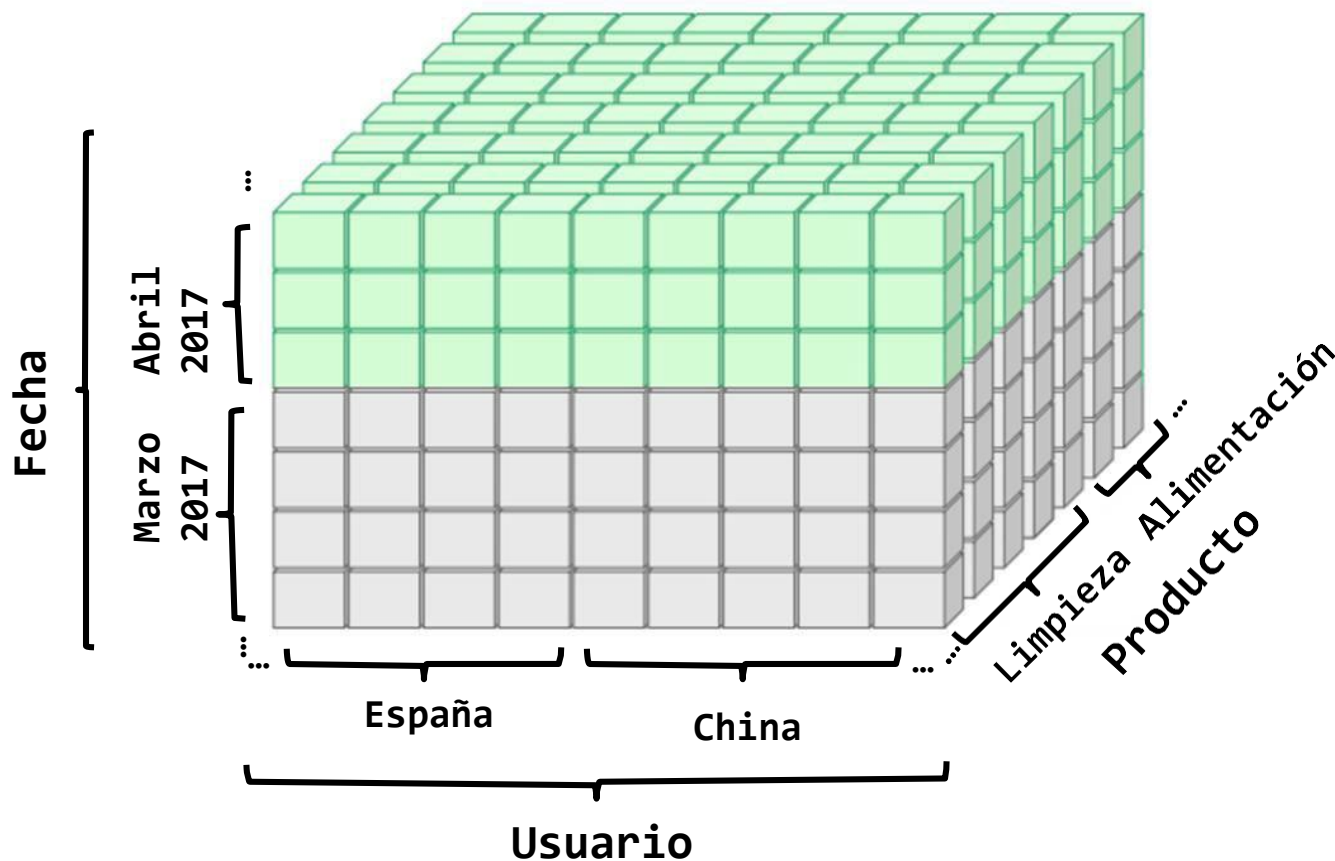
OLAP: ejemplo de cubo

- El cubo OLAP facilitaría por tanto las consultas agregadas. Por ejemplo, imaginemos que en el cubo tenemos los siguientes datos:



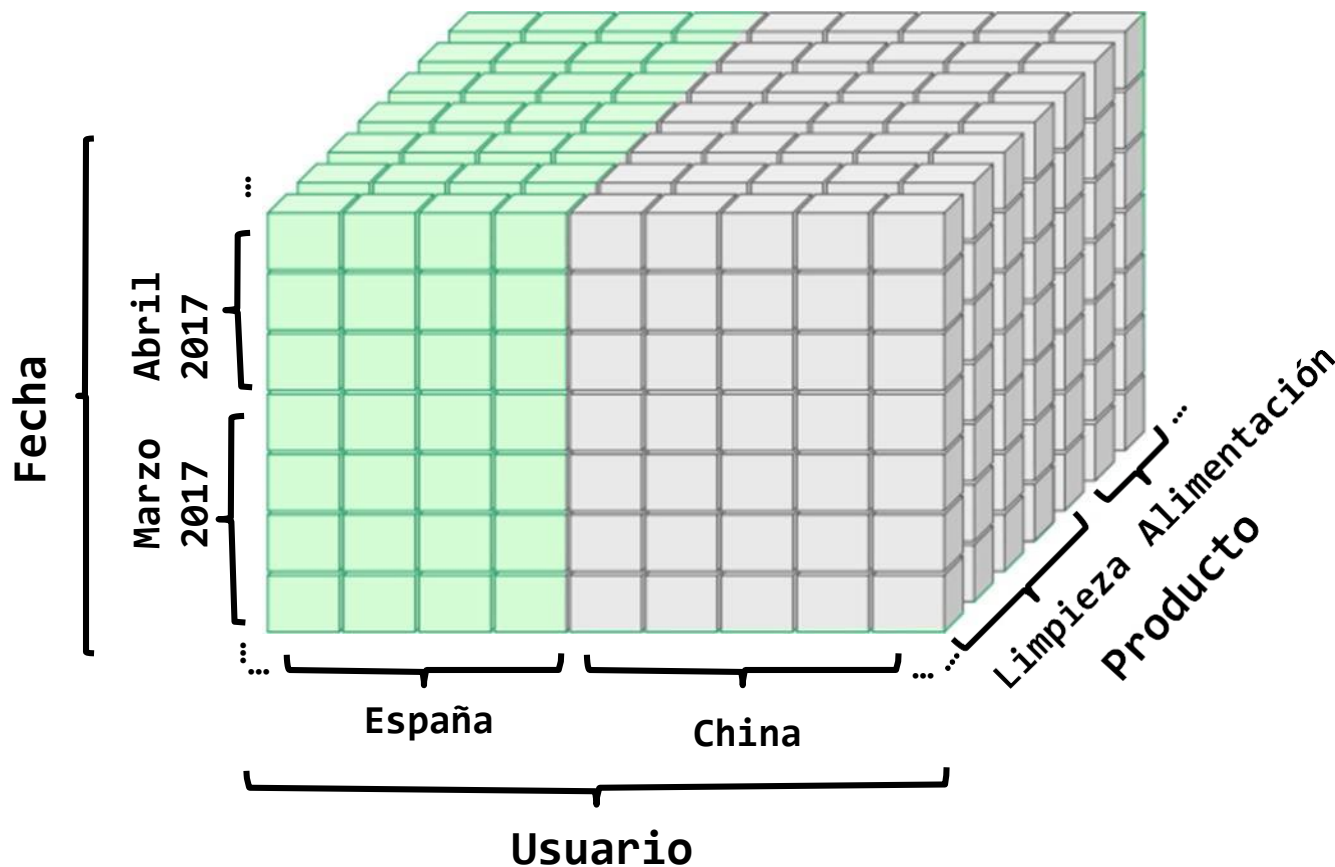
OLAP: ejemplo de cubo

- Podríamos consultar el beneficio obtenido en abril de 2017 para todos los tipos de productos, sumando el beneficio de cada cubo que pertenezca a esa fecha:



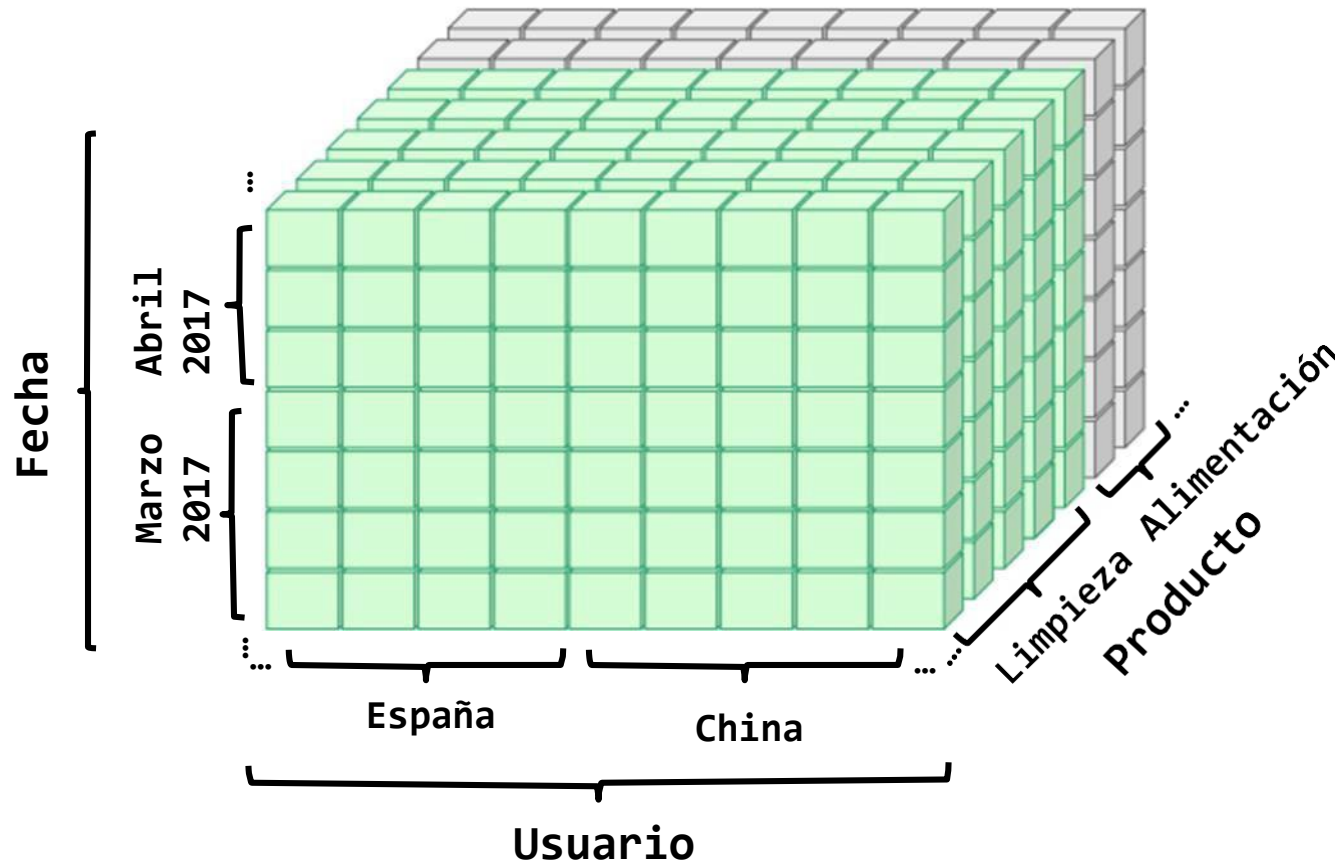
OLAP: ejemplo de cubo

También consultar el beneficio de obtenidos de las compras realizadas por usuarios españoles:



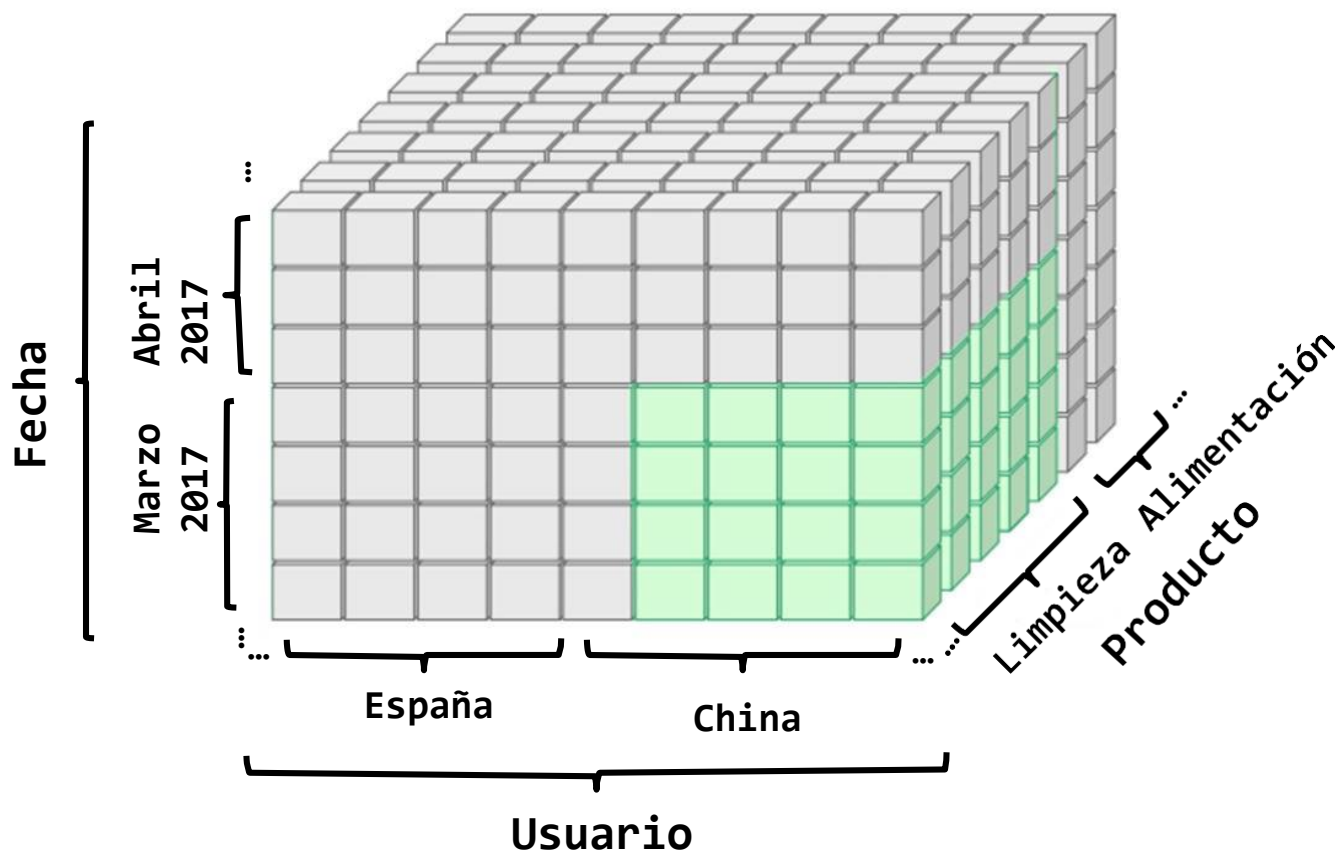
OLAP: ejemplo de cubo

O el beneficio obtenido por únicamente los productos de limpieza:



OLAP: ejemplo de cubo

También se podrían hacer agrupaciones por varias dimensiones. Por ejemplo, beneficio de los productos de limpieza vendidos a usuarios chinos en marzo de 2017:

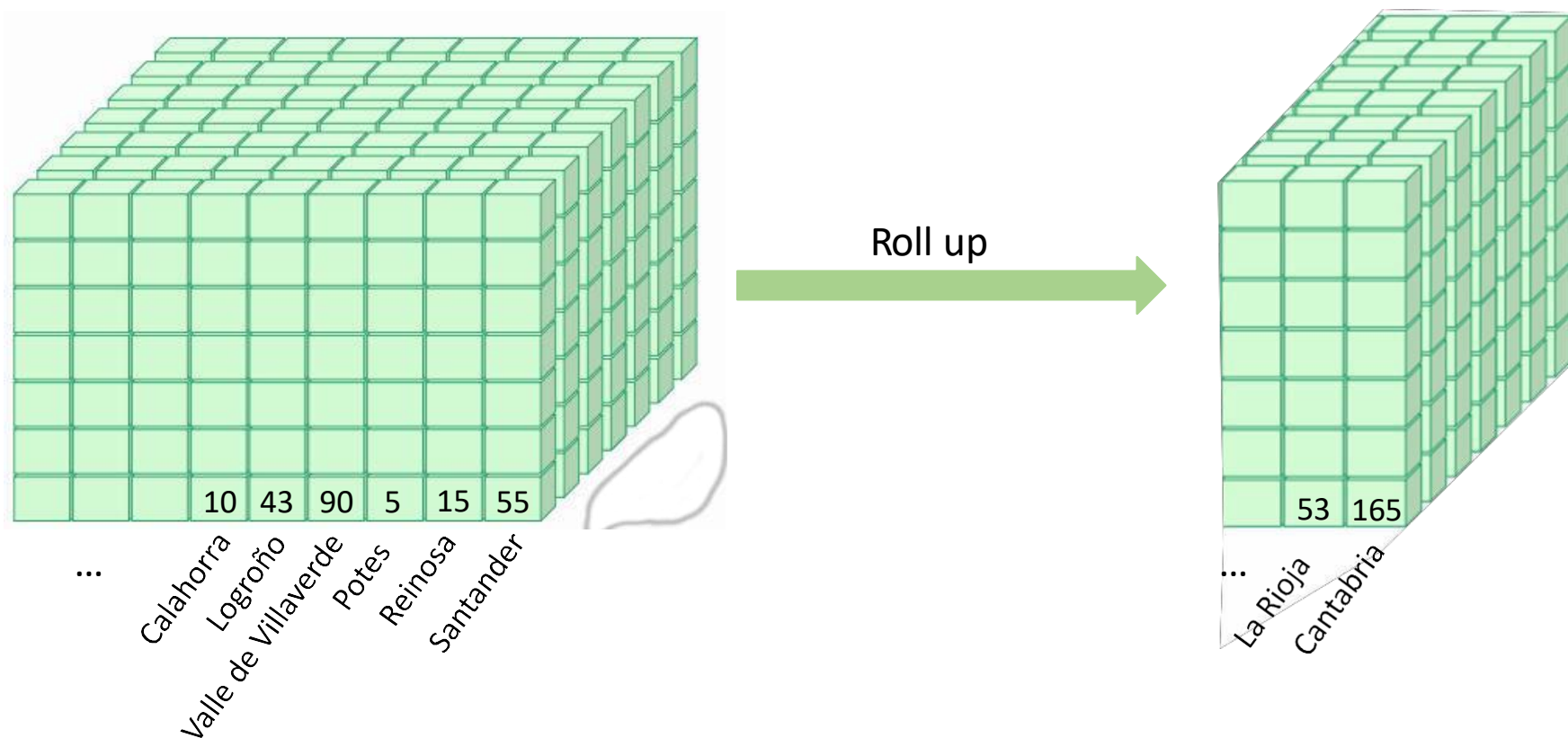


OLAP: operaciones

- En los cubos OLAP se definen las siguientes operaciones:
 - Roll up (drill up): para resumir datos. Reduce las dimensiones. Por ejemplo, pasar de consultar por meses a consultar por años en la dimensión fecha.
 - Roll down (drill down): operación contraria a roll up. Introduce nuevas dimensiones. Por ejemplo, pasar de consultar por países a consultar por ciudades en la dimensión usuario.
 - Slice and dice: seleccionar datos y proyectarlos (similar a las consultas anteriormente expuestas).
 - Pivot (rotar): reorientar el cubo.
 - Drill: se utiliza la información de una celda del cubo para moverse a otro cubo y poder consultar información relacionada.

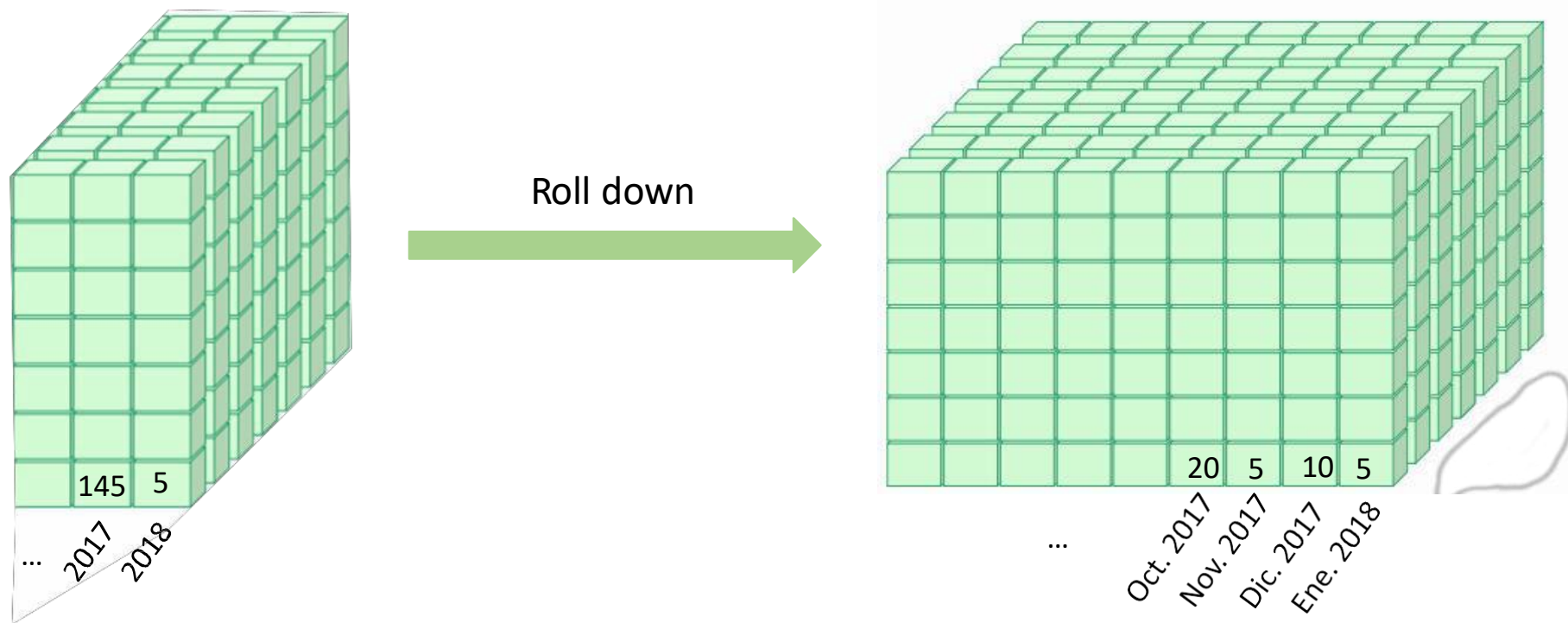
OLAP: operación Roll up

- Operación Roll up sobre la dimensión de domicilio de un usuario. En el cubo original se muestran los datos agrupados por ciudad. Tras la operación, se muestran agrupados por comunidad autónoma. La cifra de las celdas indica el beneficio obtenido por las compras y agrupado (sumado según la agrupación):



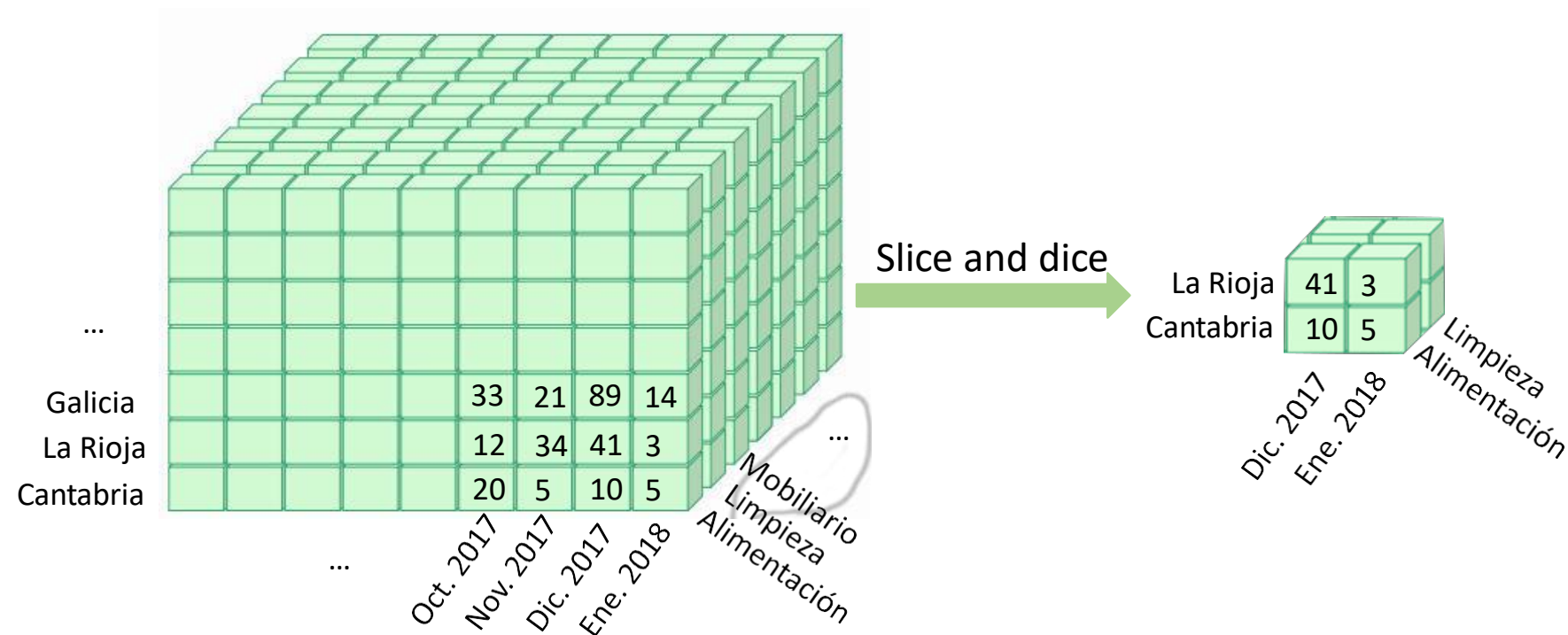
OLAP: operación Roll down

- Operación Roll down sobre la dimensión de fecha, pasando de agrupar por años a agrupar por meses. La cifra de las celdas indica el beneficio obtenido por las compras y agrupado (sumado según la agrupación):



OLAP: operación Slice and dice

- Operación Slice and dice que devuelve y muestra el beneficio obtenido por compras de productos de alimentación o limpieza, por parte de usuarios de Cantabria y La Rioja, entre Diciembre de 2017 y Enero de 2018.



OLAP: bueno para consultas, no tanto para transacciones

- En definitiva, los cubos OLAP, como ya se mencionó anteriormente, son eficientes de cara a realizar consultas de datos.
 - En el cubo, están almacenados los datos según las diferentes dimensiones requeridas para realizar agrupaciones, en un modelo desnormalizado y opuesto al relacional.
 - Ofrece operaciones Roll up, Roll down, Slice and dice, Pivot y Drill que permiten obtener los datos agrupados en diferentes niveles de granularidad según las dimensiones.
 - ¡No es bueno gestionando transacciones!. Los cubos OLAP están optimizados para consultas, pero cualquier actualización, borrado o insertado de nuevos datos daría lugar a una reestructuración del cubo, algo tremendamente ineficiente.
 - Los cubos OLAP suelen almacenar datos históricos hasta un determinado momento, provenientes por ejemplo de una base de datos relacional. Por tanto, estos cubos no se actualizan cada vez que los datos originales reciben nuevas transacciones, sino que se renuevan “cada cierto tiempo”.
 - Se podría decir que un cubo OLAP es un “snapshot” de datos agrupados en un momento concreto.

OLAP: elección de las dimensiones y hechos a almacenar

- El cubo OLAP es en realidad un hipercubo: puede (y suele) haber más de tres dimensiones.
- ¿Cómo eligen las dimensiones?. Es algo que depende de las necesidades del negocio y lo que se quiera consultar.
 - Si se desea realizar consultas por fechas, estas tendrá que conformar una dimensión.
 - Si además, se desean consultas por países de origen de los usuarios, se tendrá una dimensión de usuarios que almacene este dato, tal y como se ha visto en el ejemplo anterior.
- Y en cuanto a la granularidad de las dimensiones, ¿existe límite?
 - El límite a la granularidad que puedan alcanzar las dimensiones del cubo vendrá impuesto por la granularidad de los datos de origen con los que se construya el cubo.
 - Por ejemplo, si en los datos de origen la fecha de compra tiene una precisión en segundos, esa será la granularidad máxima a la que podrá aspirar la dimensión fecha en el cubo.
 - Por supuesto, las dimensiones pueden tener menor granularidad que los datos de origen. Por ejemplo, si de los clientes se almacenase la ciudad y el país en los datos de origen, pero sólo requiriesen consultas agrupadas por país, en el cubo sólo se almacenaría este último dato, ignorando el otro.

OLAP: On-line Analytical Processing

Cubos OLAP.

Tecnologías OLAP.

ROLAP, data warehouse y data marts.

Comparativa OLAP vs OLTP.

Extensión SQL-OLAP y ejemplos.

OLAP: tecnologías de almacenamiento

- Vale, ya tenemos claro lo que es un cubo OLAP, pero... ¿cómo se almacenan los datos?.
- Existen tres tipos diferentes de tecnologías OLAP:
 - MOLAP: acrónimo de Multidimensional On-Line Analytical Processing. Los datos se almacenan en una estructura multidimensional (por ejemplo, usando arrays o matrices).
 - ROLAP: acrónimo de Relational On-Line Analytical Processing. La estructura de datos multidimensional se define en un gestor de bases de datos relacional. Un ejemplo son los DataWarehouses, de los que hablaremos a continuación.
 - HOLAP: acrónimo de Holistic On-Line Analytical Processing. Combina MOLAP y ROLAP.

MOLAP: ventajas

- Las consultas suelen ser más rápidas que en un sistema ROLAP debido a la optimización en el rendimiento del almacenamiento (indexación, memoria caché...).
- Puede ocupar menos tamaño en disco que los ROLAP gracias a las técnicas de compresión.
- El modelo de almacenamiento basado en arrays y matrices proporciona un sistema de indexación natural.
- La complejidad de la BD permanece oculta a los usuarios.
- Almacena directamente los agregados para facilitar un acceso más rápido a los datos.

MOLAP: desventajas

- Todos los datos agregados en los diferentes niveles de las dimensiones han de ser volcados al cubo, un proceso que puede demandar mucho tiempo y recursos.
 - Esto se puede evitar, cuando es posible, con un procesamiento incremental de los datos bajo demanda.
- No se puede acceder a datos que no están en el cubo.
- Suele haber problemas de rendimiento cuando el número de dimensiones es muy alto (por encima de 10 según algunos autores).

ROLAP: ventajas

- Al implementarse en el propio gestor de BDs relacionales, no se necesita de otros recursos de software.
- Tiene acceso a las capacidades de seguridad e integridad del gestor relacional.
- Soporta un número de dimensiones más alto que los MOLAP.
- Es fácilmente escalable, se pueden añadir nuevas dimensiones con poco coste.
- Se puede acceder fácilmente a datos que no están en el cubo si estos se encuentran en una BD del mismo gestor relacional.

ROLAP: desventajas

- Algunas funcionalidades específicas de los sistemas MOLAP no se encuentran en ROLAP, debido a las limitaciones de los sistemas relacionales.
 - Aunque gracias a la extensión SQL/OLAP, existen funciones que permiten realizar multitud de consultas de agregados.
- Las consultas pueden ser menos eficientes que en MOLAP, sobre todo si son complejas, debido a que pueden requerir de la lectura de varias estructuras de tablas diferentes.
- El espacio de almacenamiento en disco requerido suele ser superior al de los sistemas MOLAP.

Software para OLAP

- Existen gran cantidad de herramientas de software dedicadas a OLAP o que lo incluyen:
 - InstantOLAP: <http://www.instantolap.com/>
 - Miner3d: <https://secure.miner3d.com/>
 - Mondrian (Pentaho): <https://mondrian.pentaho.com/>
 - IBM Cognos: <https://www-01.ibm.com/software/es/analytics/cognos/>
 - SAS: https://www.sas.com/en_us/software/olap.html
- Software orientado a la minería de datos, como RapidMiner (<https://rapidminer.com/>), KNIME (<https://www.knime.com/>), etc.
- Librerías: Python (<https://github.com/DataBrewery/cubes>), Java (<http://www.olap4j.org/>),...
- Soporte en sistemas SGDBR tradicionales (data warehouse, extensión SQL-OLAP): SQL Server (<https://www.microsoft.com/es-es/sql-server/sql-server-downloads>), Oracle (<https://www.oracle.com/database/index.html>), etc...

OLAP: On-line Analytical Processing

Cubos OLAP.

Tecnologías OLAP.

ROLAP, data warehouse y data marts.

Comparativa OLAP vs OLTP.

Extensión SQL-OLAP y ejemplos.

Data warehouse: definición

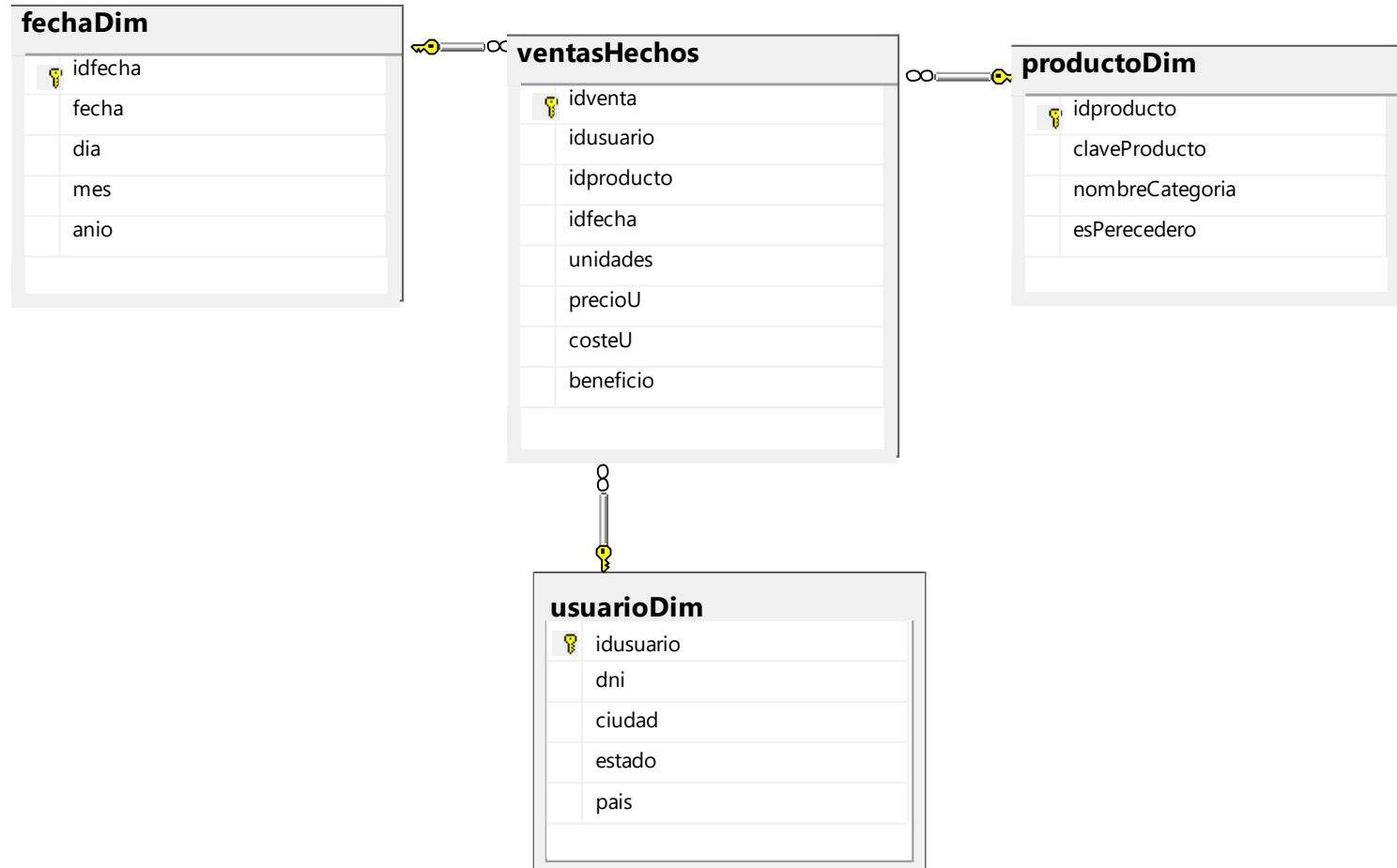
- Definición de data warehouse (DW) (almacén de datos) según Kimball:

Un almacén de datos es una copia de los datos transaccionales estructurados específicamente para **consultas y análisis**.

- La clave de esta definición está en el objetivo de los data warehouses: consultas y análisis.
- Los data warehouses (DWs) pueden ser implementados en SGBDRs utilizando tablas.
 - Y dando por tanto soporte a la creación de cubos OLAP (ROLAP).
 - Si bien se implementan en SGBDRs, los DW siguen un modelo dimensional, en contraposición al modelo relacional.

Data warehouse: ejemplo de diagrama en estrella

- Un DW puede ser implementado como un diagrama en estrella, como muestra el ejemplo siguiente:



Componentes de un sistema BI con DW y OLAP

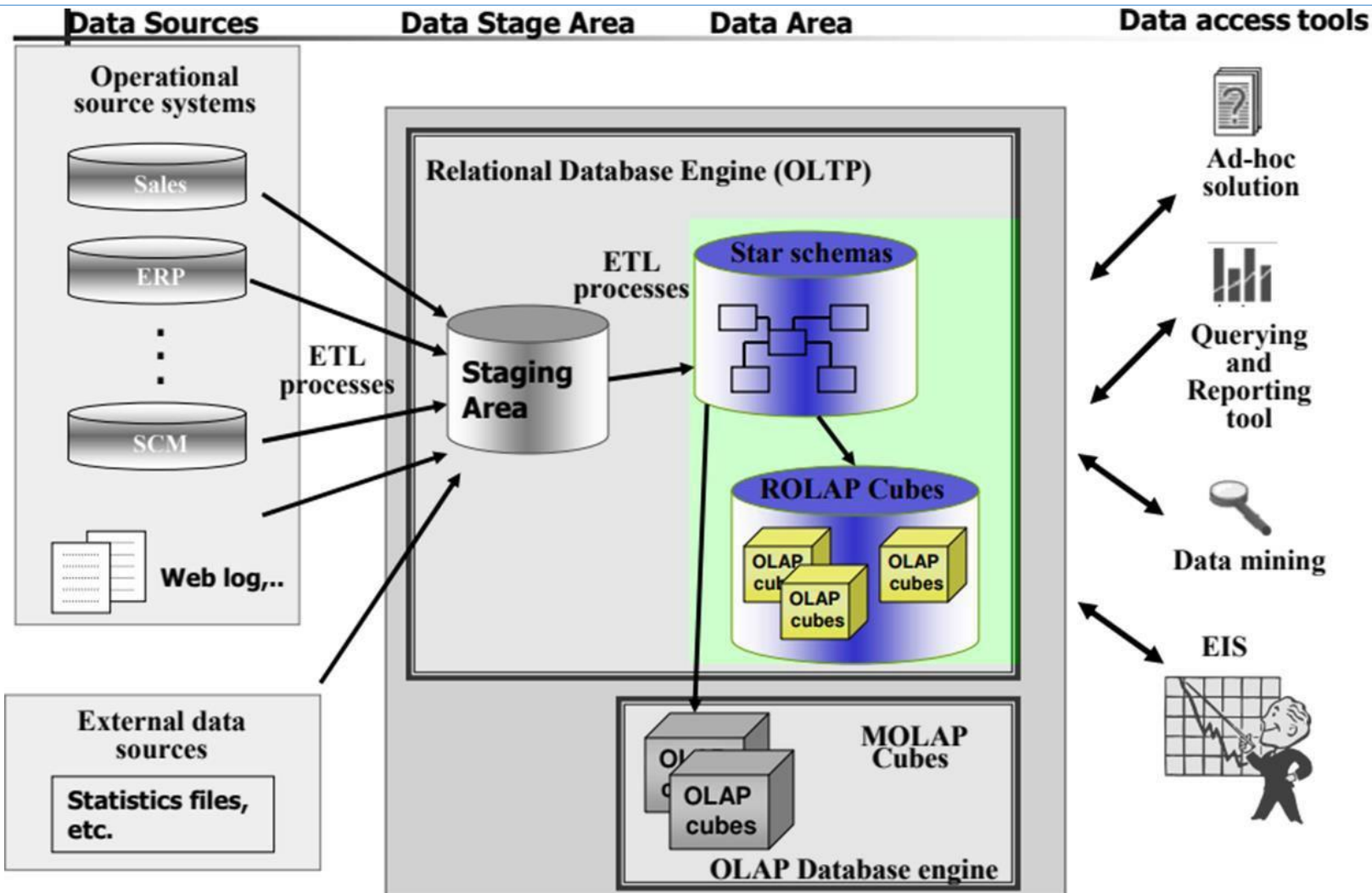


Imagen extraída de <http://docshare01.docshare.tips/files/24773/247735327.pdf>

Diagrama en estrella: elementos

- Tablas de dimensiones: representan los diferentes factores de agrupación sobre los que se quiere analizar el negocio.
- Dentro de las dimensiones hay 3 tipos de columnas diferentes: claves simples, claves de gestión y atributos.

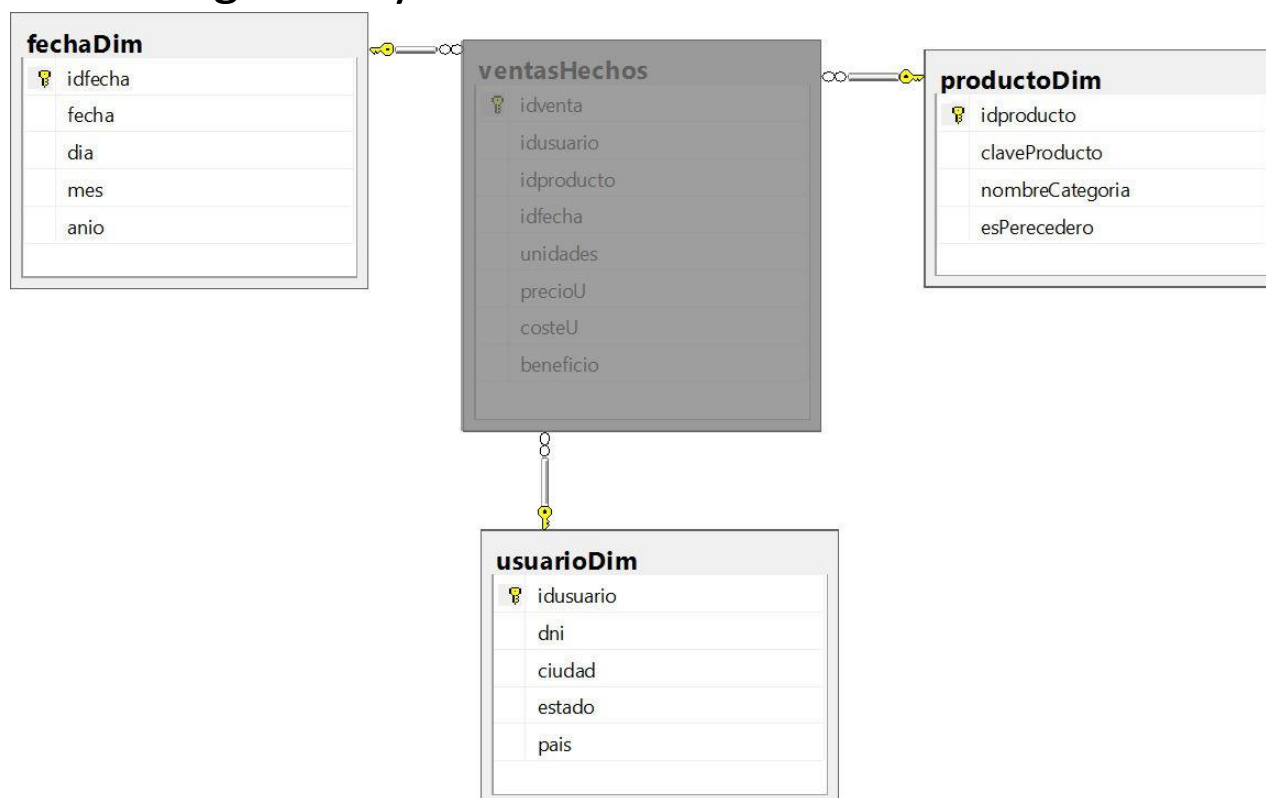


Diagrama en estrella: elementos

- Claves simples: identifican unívocamente las filas de la dimensión. No tienen significado propio. Proporcionan independencia ante cambios en las claves de producción.

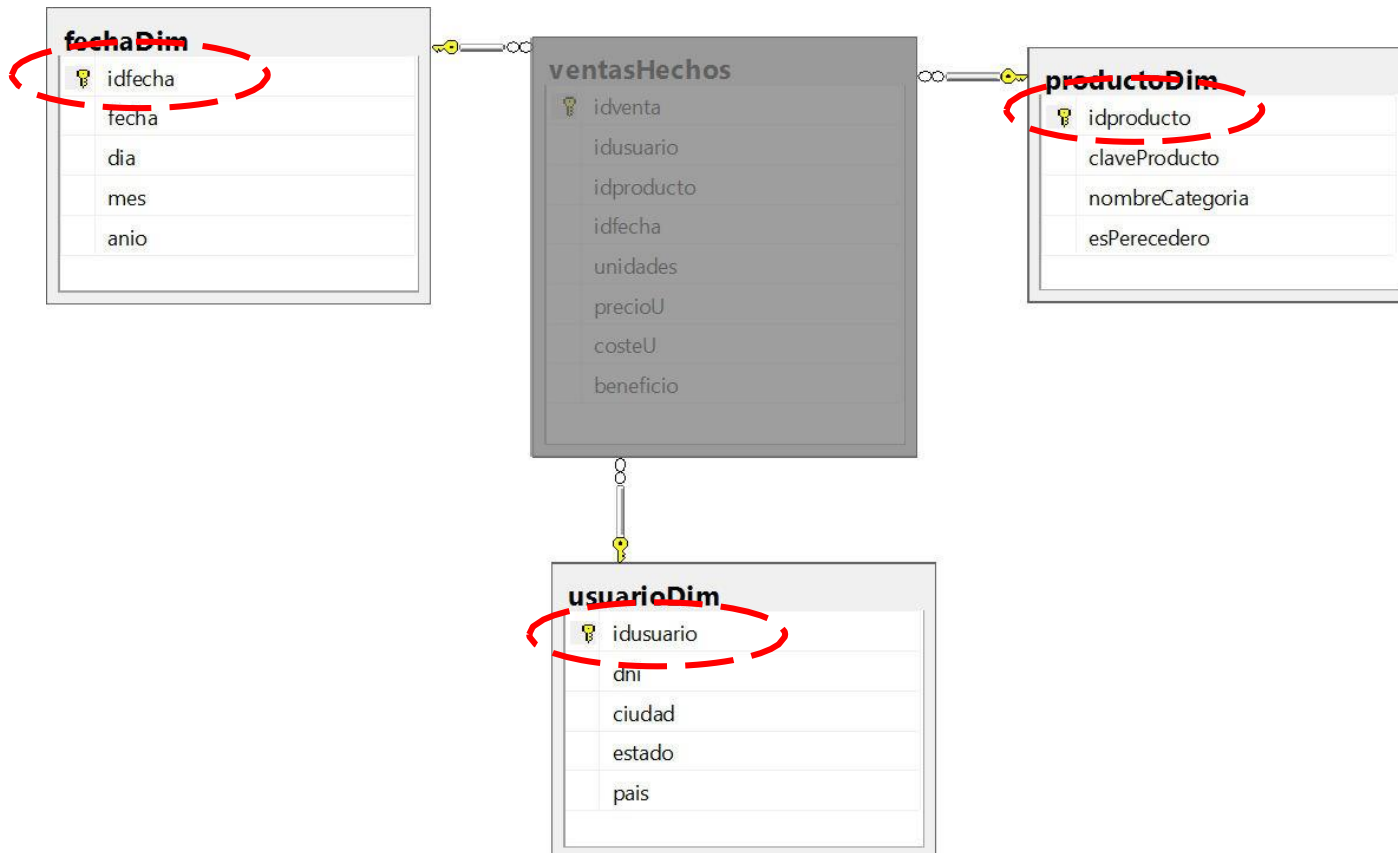


Diagrama en estrella: elementos

- Claves de gestión: identifican a las filas. Están dotadas de significado. En el caso de la dimensión de fecha, la clave de gestión es la fecha completa.

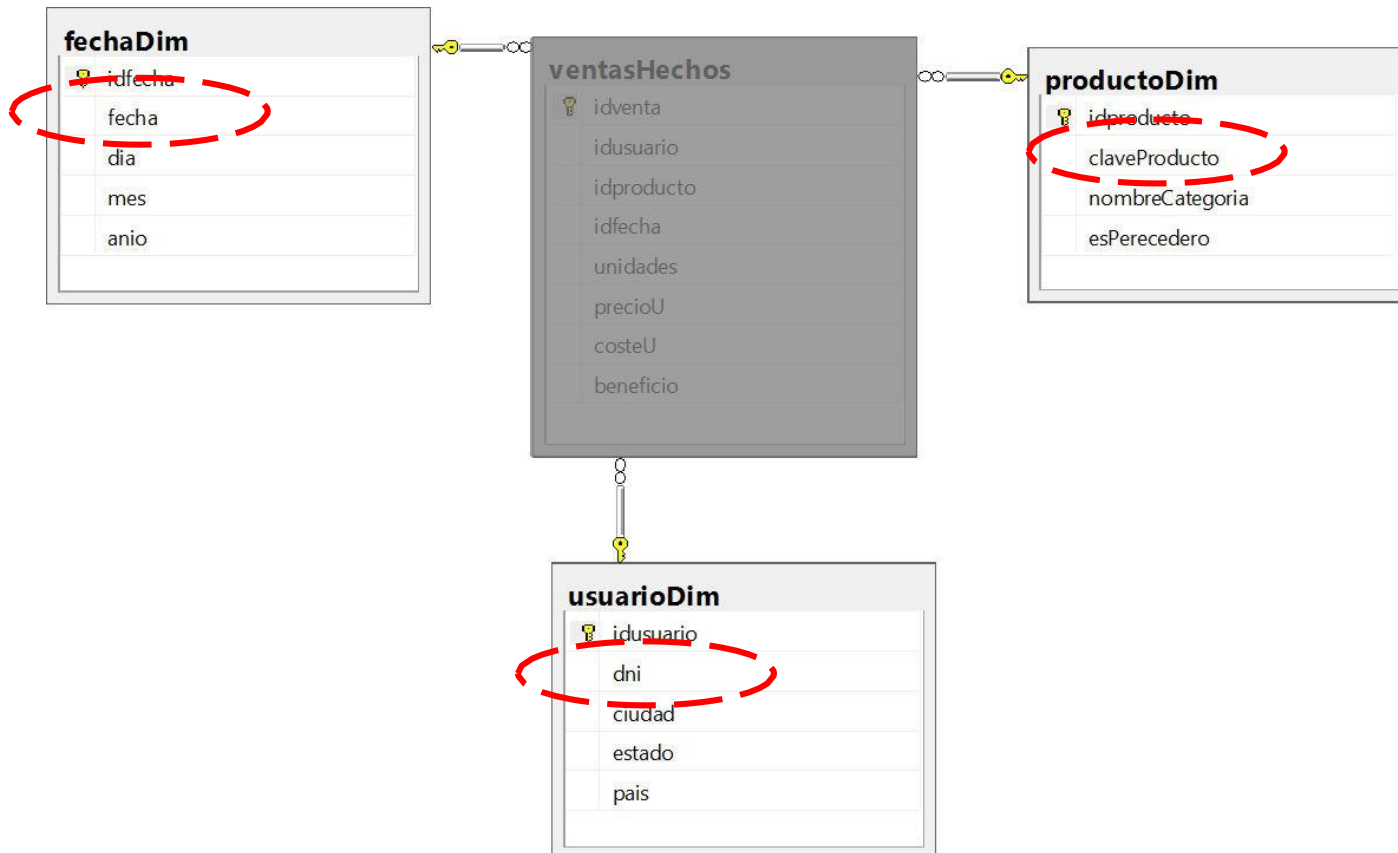


Diagrama en estrella: elementos

- Atributos: criterios de agregación para las consultas y el análisis.

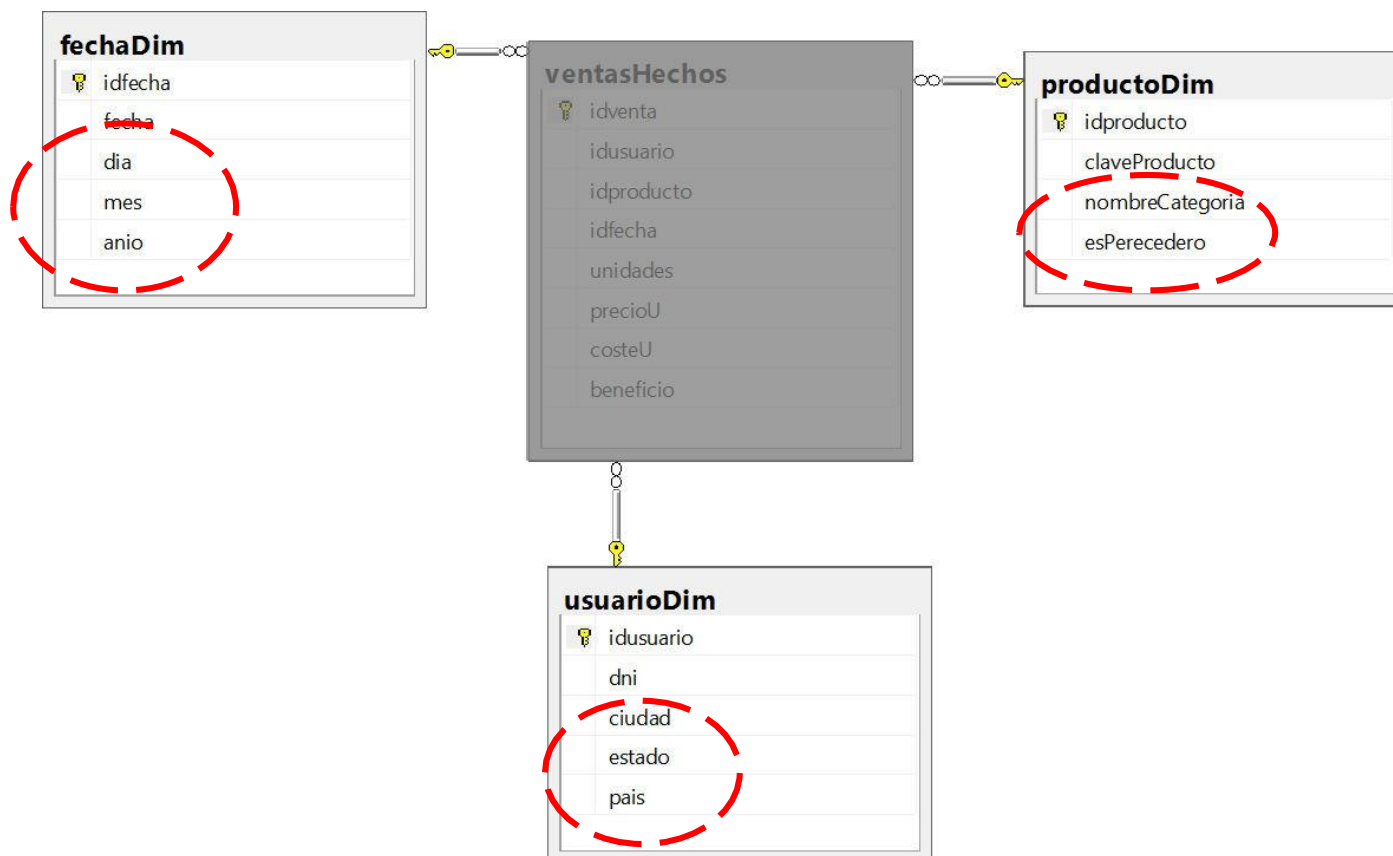


Diagrama en estrellas: elementos

- Tabla de hechos: contiene los valores de medida del negocio.
- Tres tipos de columnas: clave simple, claves de referencia y atributos (hechos).

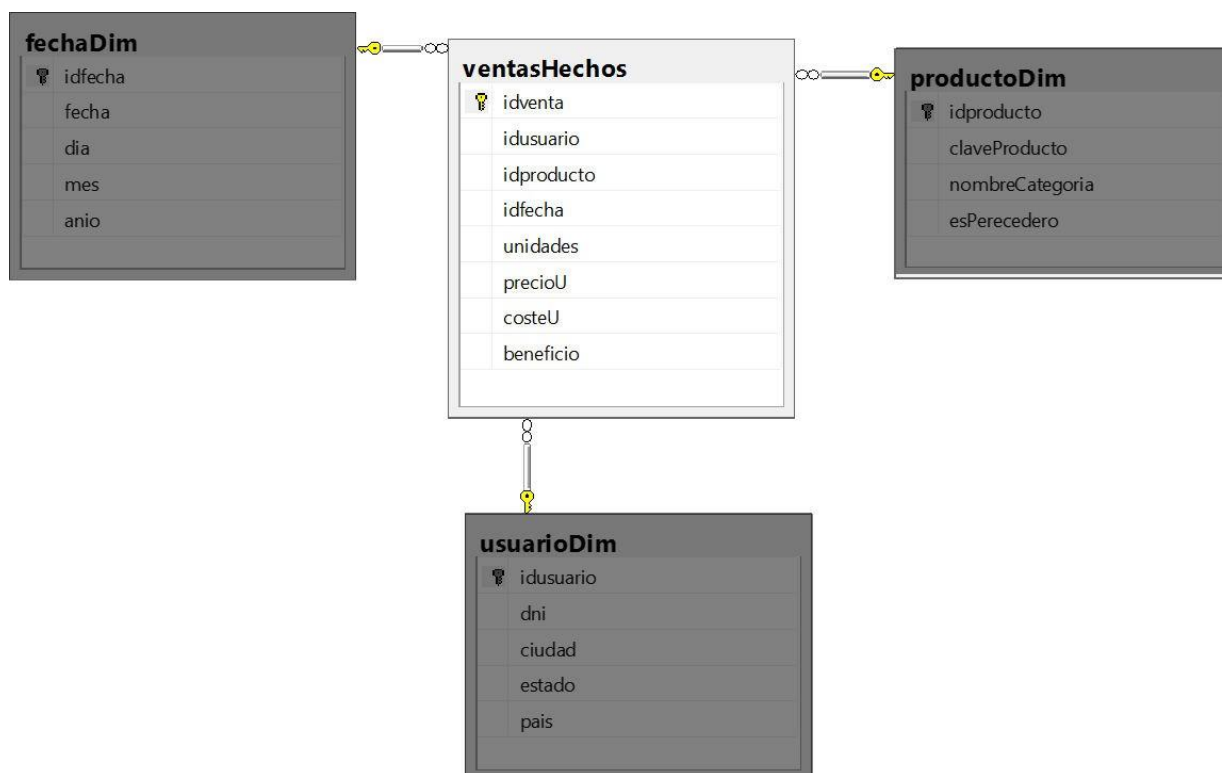


Diagrama en estrellas: elementos

- Clave simple: identifica unívocamente a cada una de las filas de la tabla de hechos, que en este caso contienen los datos de las compras realizadas por fecha, usuario y producto.

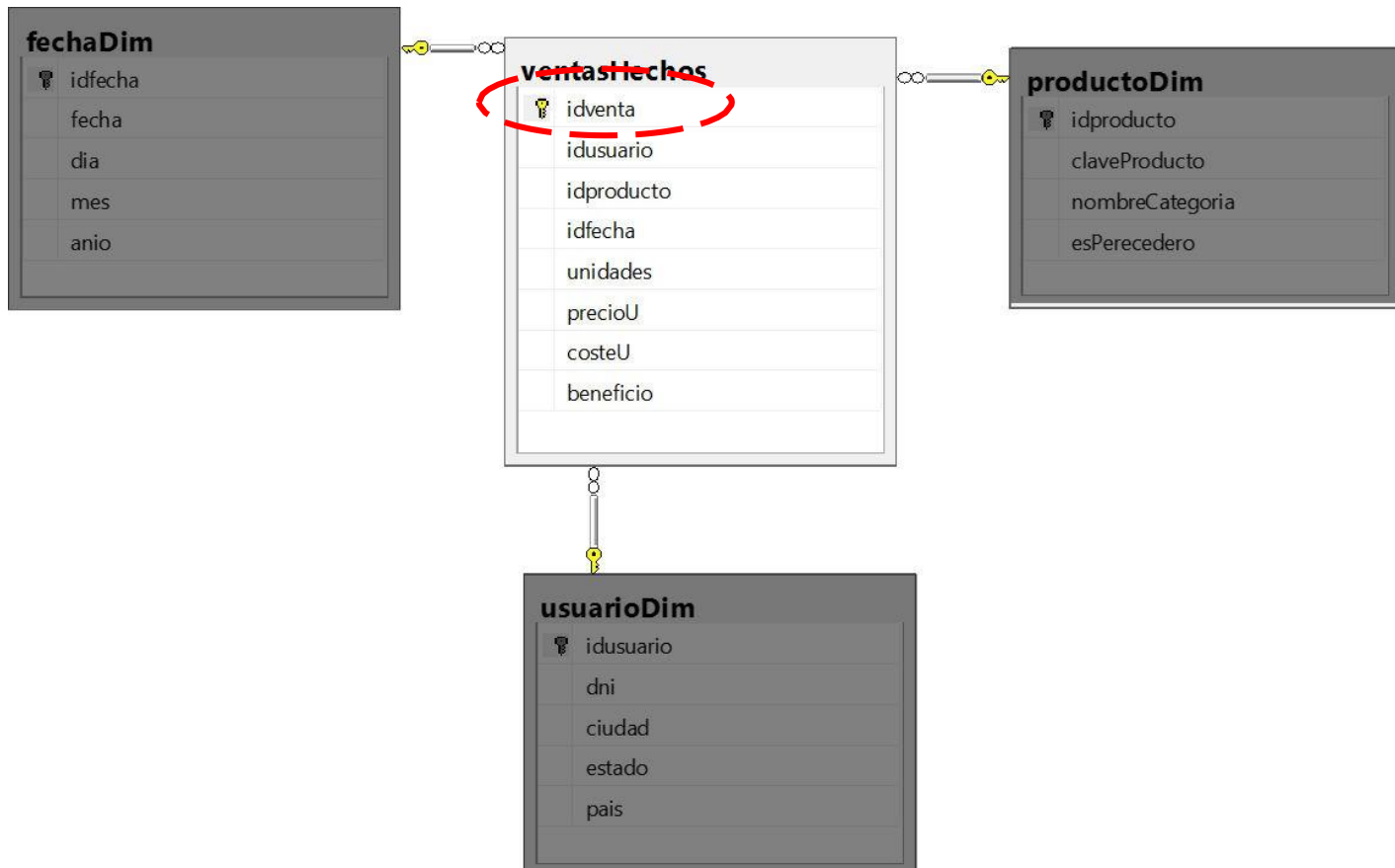


Diagrama en estrellas: elementos

- Clave de referencia: contienen las referencias a cada una de las dimensiones.

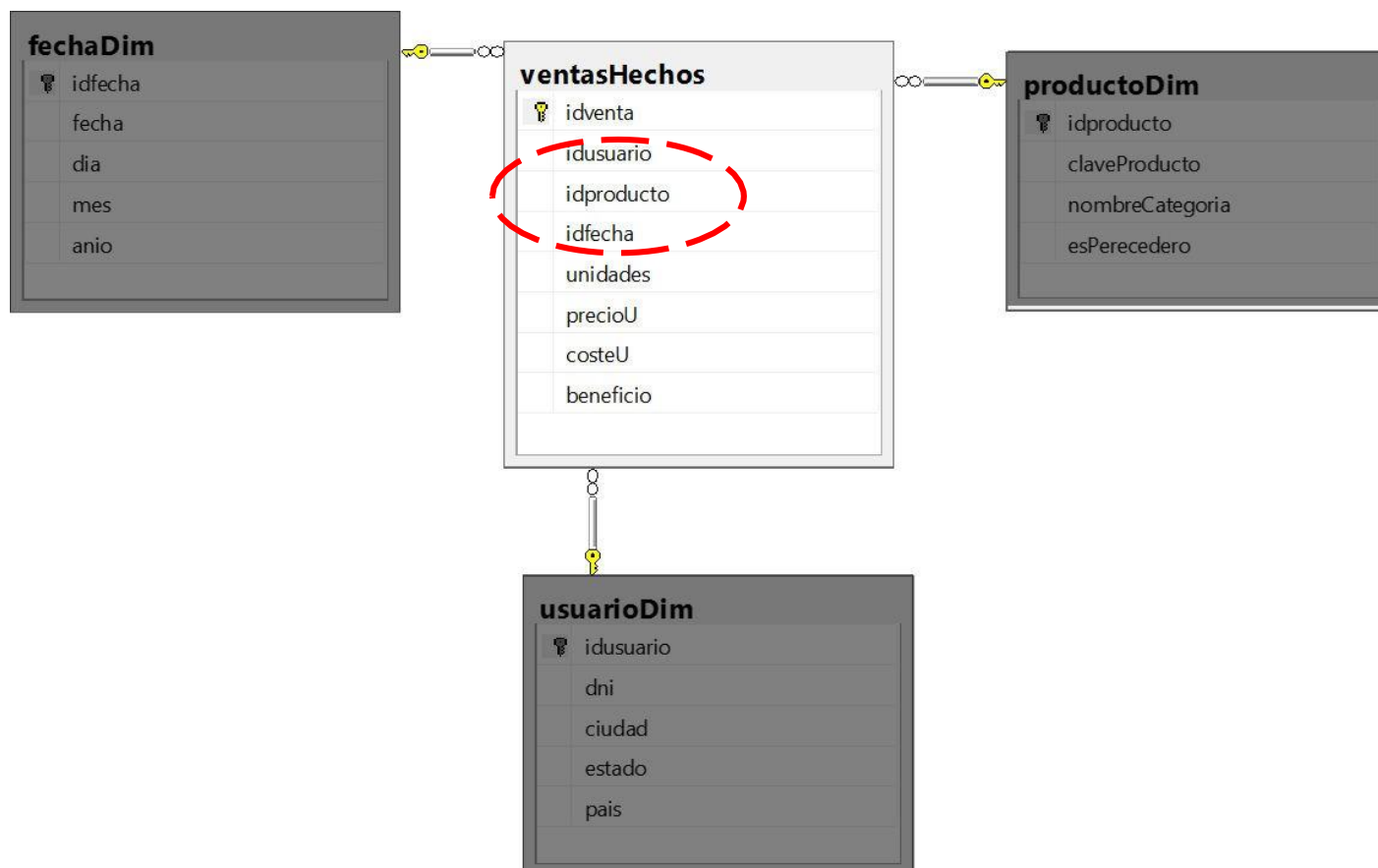


Diagrama en estrellas: elementos

- Atributos (hechos): medidas de negocio que se desea analizar.

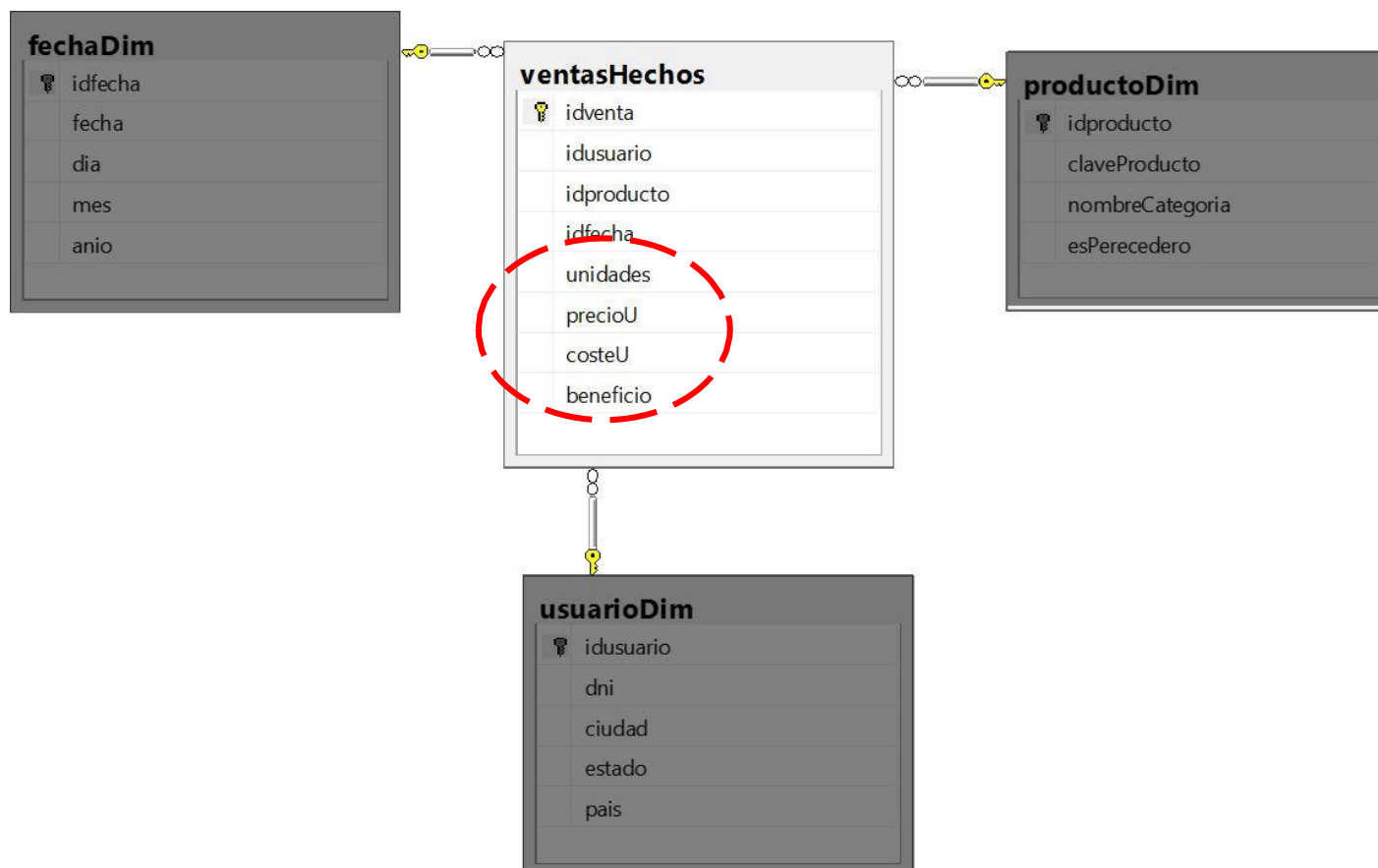


Diagrama en estrella: ejemplo de creación en SQL

- Dado que se está usando un sistema SGBDR para crear el diagrama en estrella, pueden usarse las estructuras de tabla para crearlo.
- Ejemplo de código para crear las tablas de dimensiones (en SQL Server 2016 Express):

```
create table productoDim(  
idproducto int not null identity primary key,  
claveProducto varchar(30) not null,  
nombreCategoria varchar(50) not null,  
esPerecedero bit not null  
);
```

```
create table usuarioDim(  
idusuario int not null identity primary key,  
dni varchar(100) not null,  
ciudad varchar(20) not null,  
estado varchar(20) null, pais  
varchar(20) not null  
);
```

```
create table fechaDim(  
idfecha int not null identity primary key,  
fecha date not null,  
dia int not null,  
mes int not null,  
anio int not null,  
);
```

Campo autonumérico (identity)



Diagrama en estrella: ejemplo de creación en SQL

- Ejemplo de código para crear la tabla de hechos(en SQL Server 2016 Express):

```
create table ventasHechos(  
  idventa int not null identity primary key,  
  idusuario int not null foreign key references usuarioDim(idusuario),  
  idproducto int not null foreign key references productoDim(idproducto),  
  idfecha int not null foreign key references fechaDim (idfecha), unidades  
  money not null,  
  precio money not null, coste  
  money not null, beneficio as  
  (precio-coste)  
);
```

Campo autonumérico (identity)

Referencias a las dimensiones

Beneficio como campo calculado y almacenado

Diagrama en estrella: comparativa de consulta con respecto a la realizada en la BD relacional

- Anteriormente, en el modelo relacional, vimos como sería una consulta que devolviese los beneficios obtenidos por año para las categorías de alimentación y en base a consumidores españoles.
- Esta consulta requería de siete JOINS y una agrupación SUM con una operación interna:

```
SELECT YEAR(tc.fechaCompra) AS anio, SUM((ptc.precioU-ptc.costeU)*ptc.cantidad)
AS beneficioAnual
FROM ticketCompra tc INNER JOIN productosTicketCompra ptc ON
tc.idticketcompra=ptc.idticketcompra
INNER JOIN usuario u ON u.idusuario = tc.idusuario
INNER JOIN ciudad ci ON ci.idciudad = u.idciudad
INNER JOIN estado es ON es.idestado=ci.idestado
INNER JOIN pais pa ON es.idpais=pa.idpais
INNER JOIN producto pr ON pr.idproducto=ptc.idproducto
INNER JOIN categoria ca ON ca.idcategoria=pr.idcategoria
WHERE ca.nombreCategoria = 'alimentación' AND pa.nombrePais='España'
GROUP BY YEAR(tc.fechaCompra);
```

Diagrama en estrella: comparativa de consulta con respecto a la realizada en la BD relacional

- En el DW, la misma consulta se podría realizar de la siguiente forma:

No es necesario usar la función YEAR, el año ya está almacenado

No es necesario calcular el beneficio en base a la formula anteriormente descrita, ya está almacenado

```
SELECT fdim.anio, sum(vh.beneficio) as benefioAnio
FROM ventasHechos vh INNER JOIN fechaDim fdim ON vh.idfecha = fdim.idfecha
INNER JOIN usuarioDim udim ON udim.idusuario = vh.idusuario
INNER JOIN productoDim pdim ON pdim.idproducto = vh.idproducto WHERE
udim.pais = 'España' AND pdim.nombreCategoria = 'alimentación' GROUP
BY(fdim.anio);
```

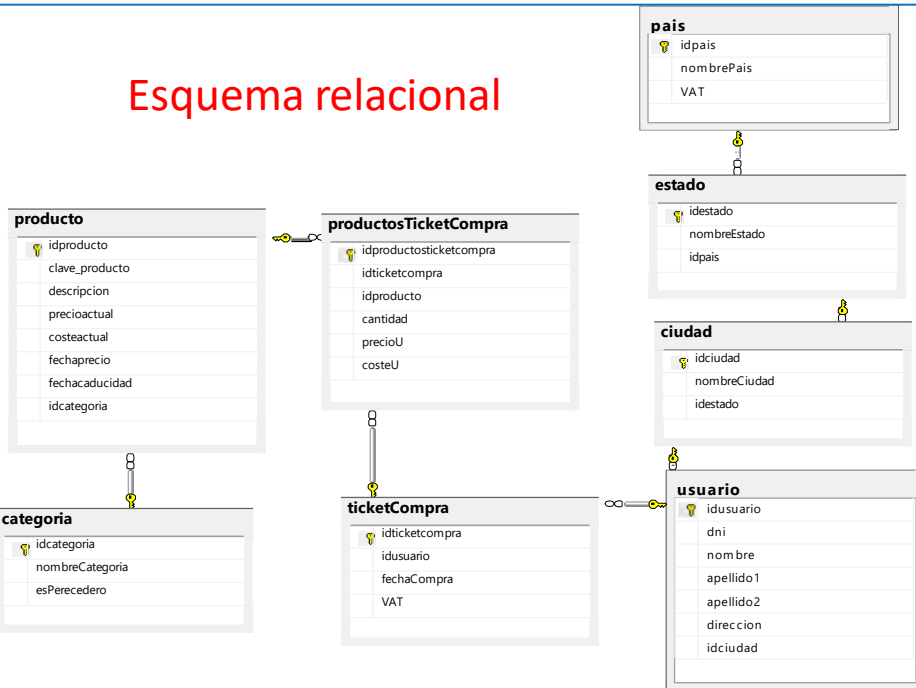
¡Sólo tres JOINS!

Diagrama en estrella: ETL desde modelo relacional

- Ya sabemos como crear un diagrama en estrella que permita la construcción de cubos OLAP y la ejecución de consultas...
 - ... ¡pero aún nos faltan los datos!.
- Es necesario realizar un proceso ETL (Extract, Transform, Load) que extraiga los datos de la fuente de almacenamiento original, los transforme de forma adecuada y finalmente los almacene en el DW.
- En el caso de ejemplo que hemos seguido en la presentación, los datos en bruto se almacenaban en una BD relacional.
 - A su vez, el DW en este ejemplo se ha implementado en el mismo SGBDR.
 - Por ello, el proceso ETL podría realizarse mediante operaciones SQL.

Diagrama en estrella: ETL desde modelo relacional

Esquema relacional



Proceso ETL



Esquema multidimensional (estrella)

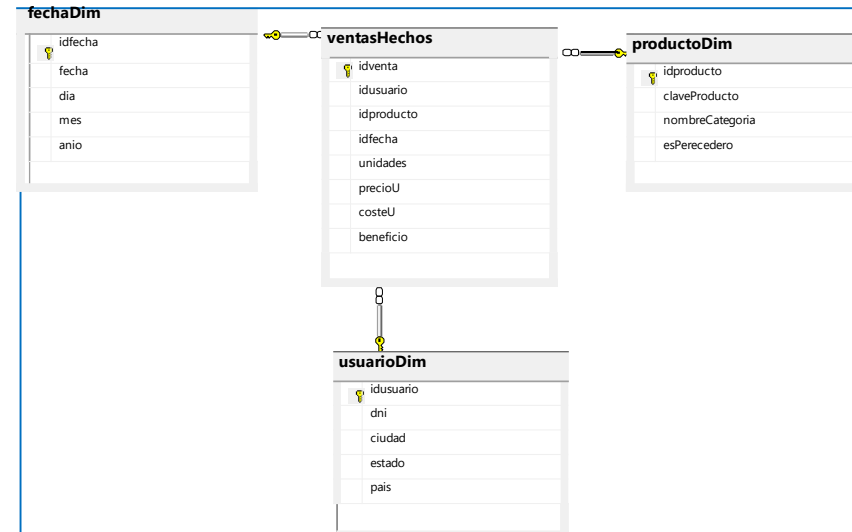


Diagrama en estrella: ETL desde modelo relacional

- Ejemplos de carga de datos en las dimensiones de productos y usuarios en SQL Server 2014:

```
INSERT INTO [dw_name].[dw_owner].productoDIM
    (claveProducto, nombreCategoria, esPerecedero)
SELECT p.clave_producto, c.nombreCategoria, c.esPerecedero
FROM [dbr_name].[dbr_owner].producto p
    INNER JOIN [dbr_name].[dbr_owner]. categoria c
    ON p.idcategoria = c.idcategoria;
```

```
INSERT INTO [dw_name].[dw_owner].usuarioDIM(dni, ciudad, estado, pais)
SELECT u.dni, c.nombreCiudad, e.nombreEstado, p.nombrePais
FROM [dbr_name].[dbr_owner]. usuario u
    INNER JOIN [dbr_name].[dbr_owner].ciudad c
    ON u.idciudad = c.idciudad
    INNER JOIN [dbr_name].[dbr_owner].estado e
    ON c.idestado = e.idestado
    INNER JOIN [dbr_name].[dbr_owner].pais p
    ON e.idpais = p.idpais;
```

dw_name y **dbr_name** son los nombres que se les haya dado a las bases de datos que contienen respectivamente el data warehouse y la base de datos relacional.

dw owner y **dbr owner** son los esquemas de las bases de datos (<https://docs.microsoft.com/en-us/sql/relational-databases/security/authentication-access/create-a-database-schema>)

Diagrama en estrella: ETL desde modelo relacional

- Una vez rellenas las dimensiones, podríamos ya proceder a rellenar la tabla de hechos:

```
INSERT INTO [dw_name].[dw_owner].ventasHechos(idproducto, idusuario, idfecha, unidades,  
                                                precio, coste)  
SELECT pdim.idproducto, udim.idusuario, fdim.idfecha, ptc.cantidad,  
       (ptc.cantidad*ptc.precioU),(ptc.cantidad*ptc.coste)  
FROM [dbr_name].[dbr_owner].productosTicketCompra ptc  
     INNER JOIN [dbr_name].[dbr_owner].ticketCompra tc  
       ON ptc.idticketcompra =  
          tc.idticketcompra  
     INNER JOIN [dbr_name].[dbr_owner].producto  
       p ON p.idproducto = ptc.idproducto  
     INNER JOIN [dbr_name].[dbr_owner].usuario  
       u ON u.idusuario = tc.idusuario  
     INNER JOIN [dw_name].[dw_owner].productoDim pdim  
       ON pdim.claveProducto =  
          p.clave_producto  
     INNER JOIN [dw_name].[dw_owner].usuarioDim  
       udim ON udim.dni = u.dni  
     INNER JOIN [dw_name].[dw_owner].fechaDIM  
       fdim ON fdim.fecha = tc.fechaCompra;
```


Diagrama en estrella: fase de diseño

1. Definir la problemática del negocio y las cuestiones que se quieren resolver o responder sobre el mismo.
2. Definir la granularidad de las dimensiones.
 - Es mejor definir un grano lo más detallado posible.
 - Esto condiciona la flexibilidad de las consultas.
 - Una vez volcados los datos en el DW, no se puede definir un grano más fino en las dimensiones.
 - Por ejemplo, si en dimensión fecha guardamos el detalles en días, meses y años, ya no será posible consultar por franjas horarias.
 - Si quisiésemos un grano mayor en alguna dimensión, sería necesario volver a volcar los datos de la misma así como todos los datos de la tabla de hechos. ¡Extremadamente indeficiente!.
3. Escoger las dimensiones.
 - En ocasiones, habrá que decidir si determinados datos van en dos dimensiones diferentes o se unen en una sola.
 - Por ejemplo, en el caso de la fecha y la hora de la compra, se podrían crear dos dimensiones diferentes, o condensar todos los datos en una sola dimensión.
4. Determinar que hechos son los que han de incluirse.
 - Los hechos han de ser, por norma, aditivos, para poder consultar en diferentes niveles de granularidad y en base a diferentes dimensiones.
 - En caso de almacenar porcentajes y proporciones, es recomendable almacenar todos los datos en varios atributos (p.e. numerador, denominador, porcentaje).
 - Puede haber casos en que lo que se busque sea valorar si un suceso ocurre o no. En este caso, se recomienda dar valores de 0 y 1 (NO y SÍ) para que sea aditivo.
 - Hay medidas que no son aditivas, como el precio unitario de los productos. En vez de éste, se almacena el precio total (precio unitario * unidades).
 - Otras medidas no aditivas pueden agregarse calculando valores medios.

Diagrama en estrella: tablas y atributos

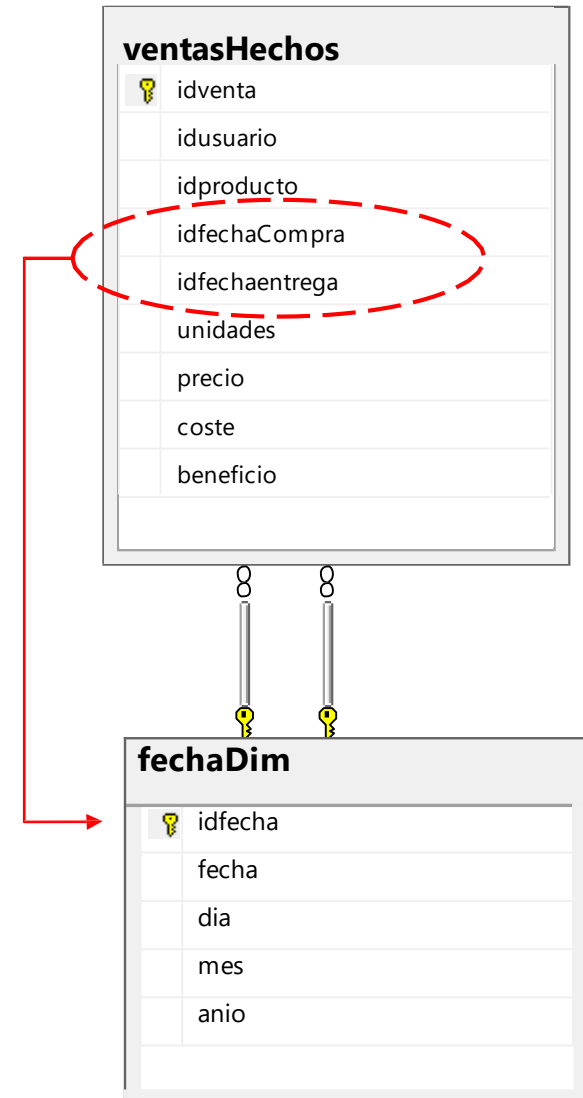
- Las dimensiones pueden y suelen tener gran cantidad de atributos.
 - Esto facilita y hace más eficiente las agregaciones evitando tener que hacer cálculos.
 - Por ejemplo, la dimensión de fecha podría tener atributos no solamente de día, mes y año, sino también otros que indicasen si es o no fin de semana, el trimestre, cuatrimestre y semestre del año, el día del año (1-365/366), el día del mes (1-31), el día de la semana (lunes a domingo), si es o no festivo, etc.
- Un exceso de dimensiones puede indicar que varias de ellas no son independientes.
 - ¿Cuánto es un exceso?. No hay un número determinado a partir del cuál podamos determinar que hay que reducir dimensiones.
 - En la literatura podemos encontrar que muchos autores consideran que esta cifra se encuentra en las 25 dimensiones.
 - En caso de que esto suceda, deben combinarse dimensiones.
- En casi todos los diagramas en estrella hay una dimensión temporal.
 - Cuando se desea una granularidad mayor a los días (horas, minutos, etc.), se recomienda crear dos dimensiones diferentes.

Diagrama en estrella: extensión del esquema

- En un momento determinado, podría surgir la necesidad de extender el esquema.
 - Con nuevas dimensiones: en este caso, sería necesario añadir la nueva clave de dimensión a la tabla de hechos y cargar los nuevos valores.
 - Con nuevas medidas: se añaden a la tabla de hechos. Habrá de rellenarse este valor para los hechos anteriores a este evento.
 - Con nuevos atributos de dimensión: se añaden nuevas columnas a la tabla de dimensión.
 - Con dimensiones existentes a una mayor granularidad: sería necesario eliminar la tabla de hechos y reconstruirla de nuevo.

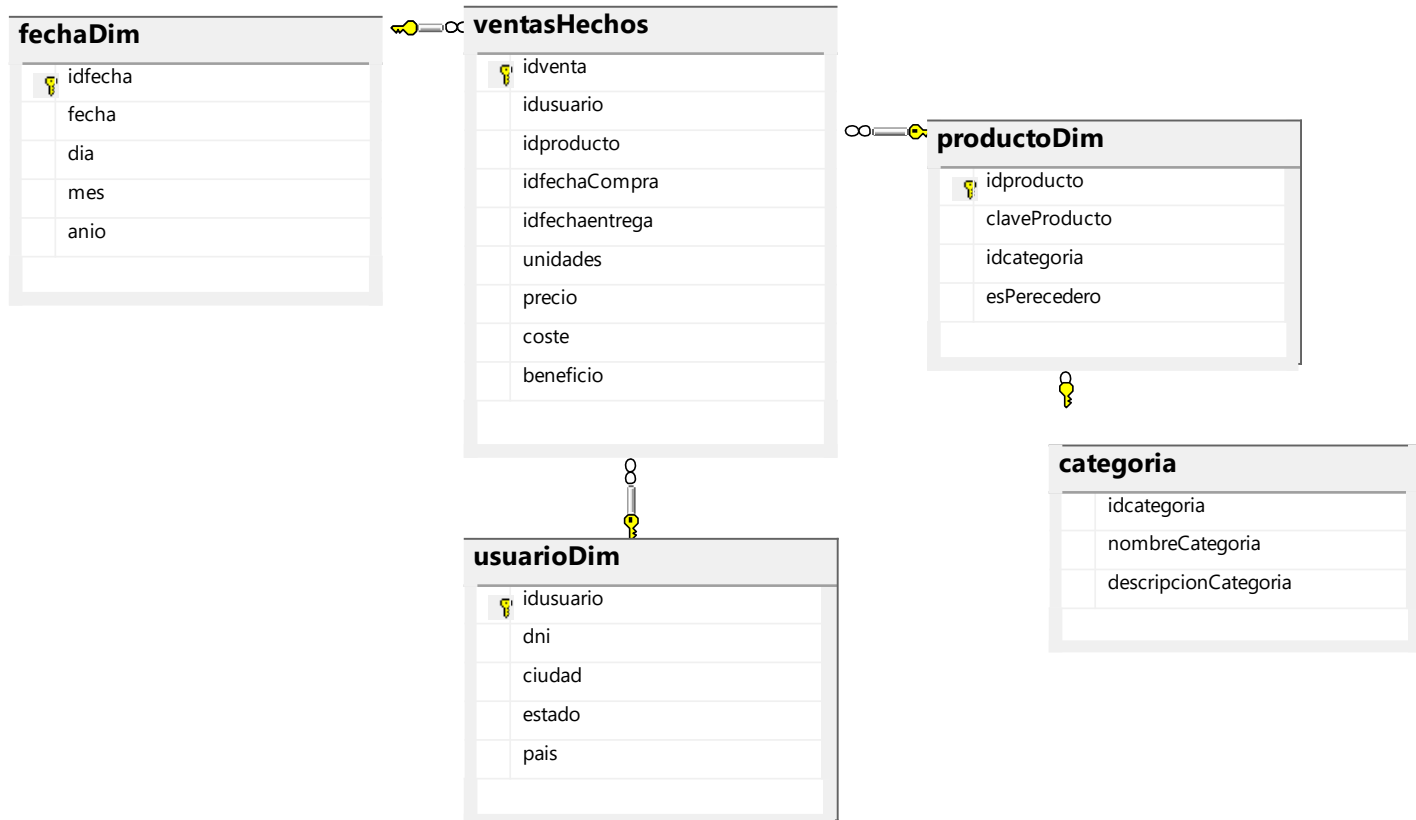
Diagramas en estrella: modelo dimensional extendido

- En ocasiones, en un diagrama en estrella, podría suceder que una tabla de hechos reference varias veces a la misma dimensión con diferentes objetivos.
- En el ejemplo del supermercado, la tabla de hechos podría almacenar la fecha de compra y también la fecha de entrega de los productos.
 - Considerando, por supuesto, que esta última estuviese almacenada en la BDr de origen.
- En ese caso, la tabla de hechos tendría dos claves diferentes apuntando a la tabla de dimensión de fecha:



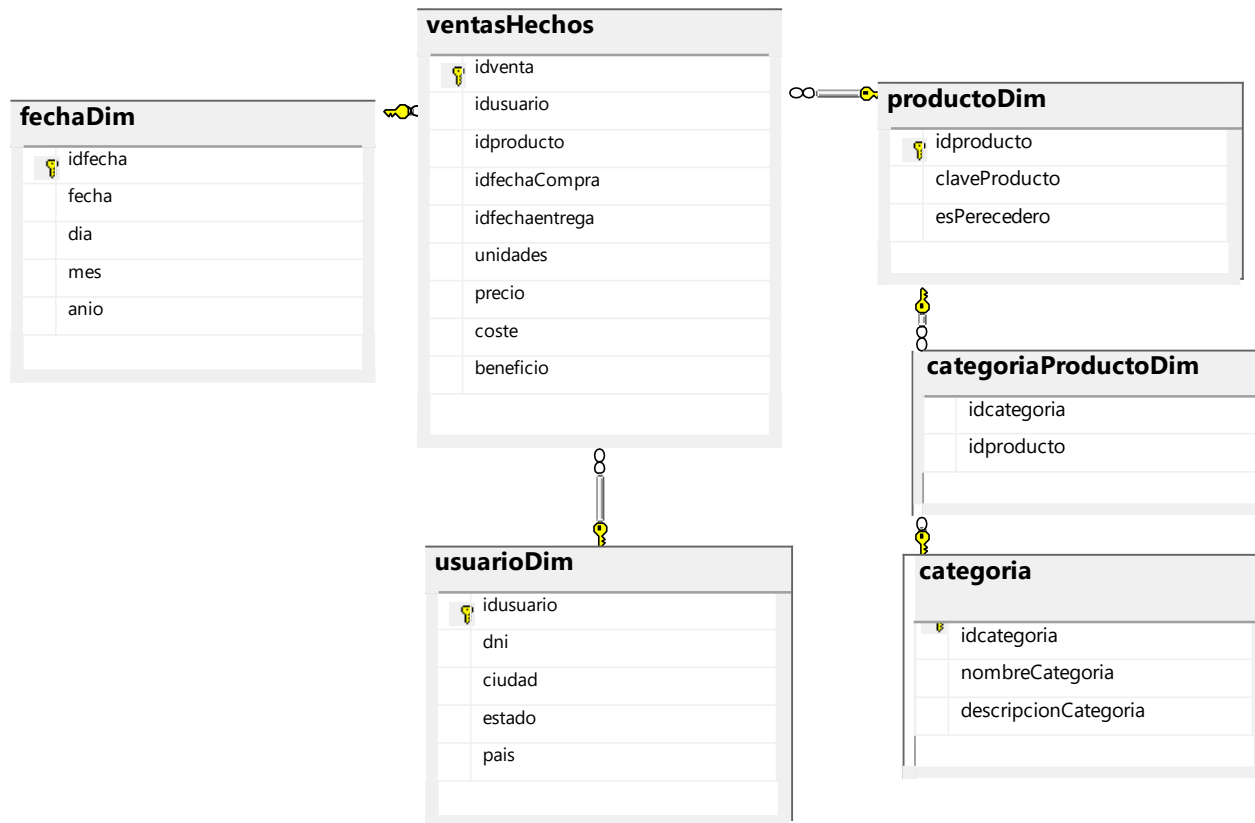
Diagramas en estrella: modelo dimensional extendido

- Las dimensiones pueden ser también jerarquizadas en varias tablas (normalizadas). Por ejemplo, podríamos tener una tabla para almacenar los datos de los productos, y otra los datos de sus categorías. Esto se conoce como esquema en copo de nieve:



Diagramas en estrella: modelo dimensional extendido

- Incluso podría suceder que a un hecho almacenado en la tabla de hechos le correspondan varias filas de una dimensión.
- Por ejemplo, imaginemos que los productos pudiesen tener más de una categoría -> relación N a N

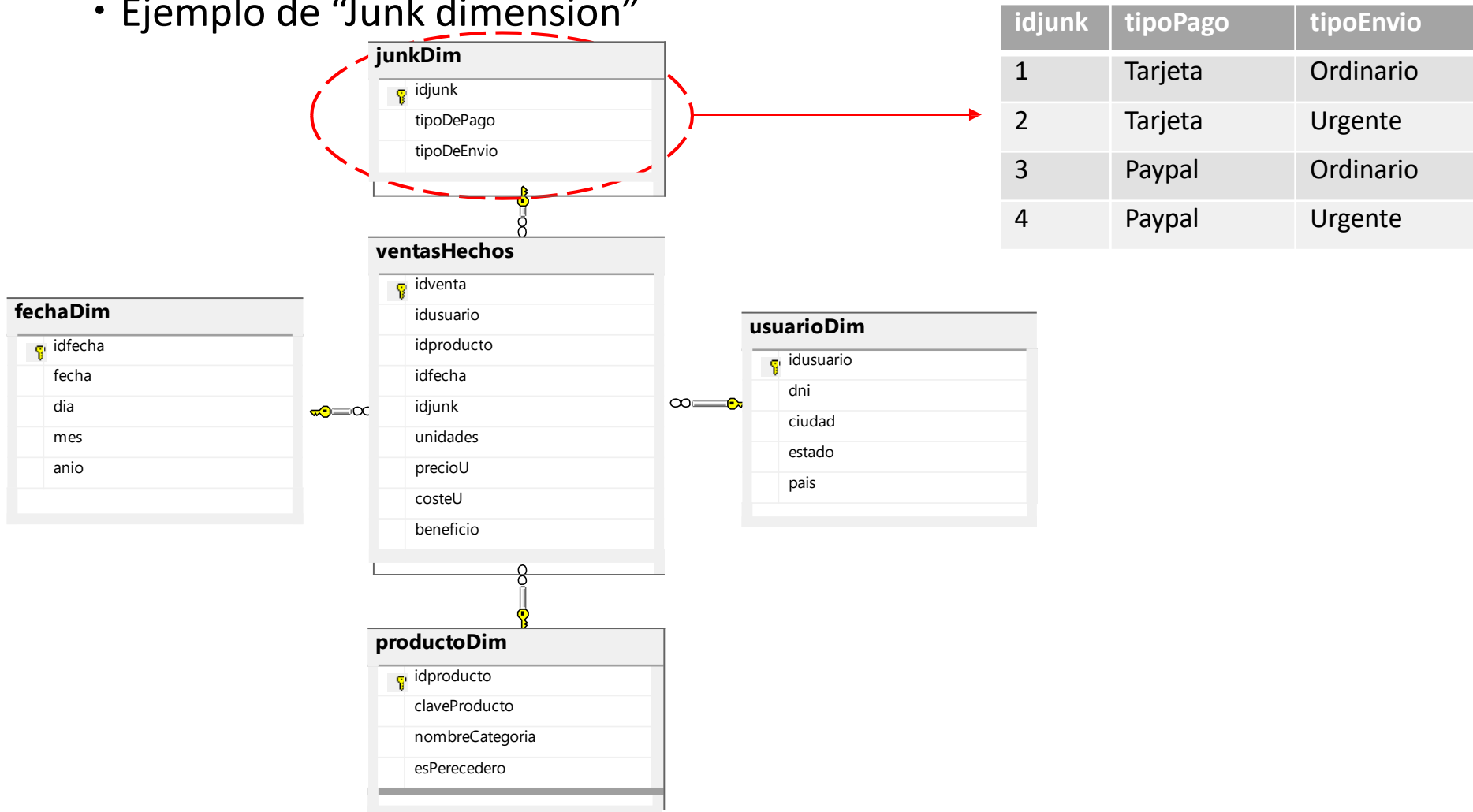


Diagramas en estrella: modelo dimensional extendido

- Puede suceder que aparezcan atributos que no pertenecen a ninguna de las dimensiones creadas y que aparentemente no tienen entidad suficiente para conformar una propia.
- Ejemplos de estos atributos pueden ser el tipo de pago (tarjeta, paypal, etc.), tipo de envío, etc.
- En estos casos, existen varias posibilidades para contemplar e incluir estos atributos en el esquema:
 - Incluirlos directamente en la tabla de hechos.
 - Crear una dimensión para cada atributo, con lo que se corre el riesgo de que el número de dimensiones crezca.
 - Crear lo que se conoce como una Junk dimensión: una dimensión que aglutina todos estos atributos junto con todas sus posibles combinaciones de valores.


Diagramas en estrella: modelo dimensional extendido

- Ejemplo de “Junk dimension”



Diagramas en estrella: modelo dimensional extendido

- En muchos casos, los hechos están relacionados con documentos como tickets de compra, facturas, etc.
- Estos datos, si quisiésemos reflejarlos en el esquema en estrella, daría lugar a dimensiones vacías.
- Por ello, otra propuesta es incluirlos directamente en la tabla de hechos:

ventasHechos *	
	idventa
	idusuario
	idproducto
	idfecha
	numticketcompra
	unidades
	precioU
	costeU
	beneficio

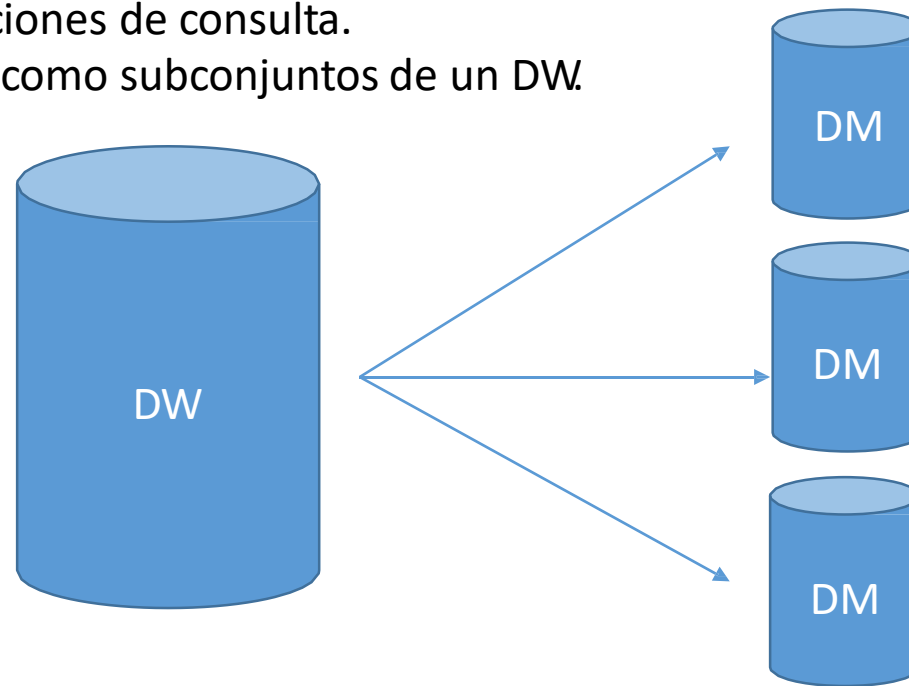
Número de ticket de las ventas

De los data warehouses a los data marts (y viceversa)

- En los negocios, es bastante frecuente encontrarse con diferentes procesos que requieren de un análisis para la toma de decisiones.
- Estos diferentes procesos suelen estar bien diferenciados, y los tipos de usuarios que requieren de acceso a los datos de los mismos y a su análisis también lo son. Por ejemplo, en una cadena de supermercados:
 - un tipo de usuario podría ser el encargado de las adquisiciones de stock, cuyas necesidades son las de analizar los datos del proceso de adquisición de existencias de los proveedores.
 - otro tipo de usuario sería el jefe de ventas, que requiere del acceso a los procesos de ventas a clientes.
 - los dos anteriores también podrían estar a su vez divididos por áreas geográficas u otros criterios.
- Cada tipo de usuario requiere de acceso a diferentes datos de procesos.

Data marts: definición

- Los data marts son un tipo especial de DW que almacenan datos de un área específica del negocio.
 - Permiten a los usuarios interesados en estas áreas acceder única y exclusivamente a los datos de las mismas.
 - Ayudan por tanto a optimizar el acceso a los datos, dividiéndolos en diferentes data marts independientes, en vez de tener un solo DW que reciba todas las peticiones de consulta.
 - Pueden verse como subconjuntos de un DW.



Data marts: cuestiones a tener en cuenta

- Si queremos asegurar la consistencia en los datos, los data marts han de ser rellenados desde el data warehouse, y no desde la fuente de datos original.
- Hay que tener en cuenta que la creación de data marts genera una necesidad de una mayor infraestructura hardware, al tiempo que requieren de recursos computacionales para la carga de datos.
- La granularidad de los data marts puede ser igual o menor que la del data warehouse de origen.
- Otra ventaja de los data marts es que definen “por sí mismos” un control de acceso a los datos, permitiendo a cada tipo de usuario solamente acceder a los datos del propio data mart.

OLAP: On-line Analytical Processing

Cubos OLAP.

Tecnologías OLAP.

ROLAP, data warehouse y data marts.

Resumen y conclusiones.

Extensión SQL-OLAP y ejemplos.

Resumen y conclusiones: Comparativa OLTP vs OLAP

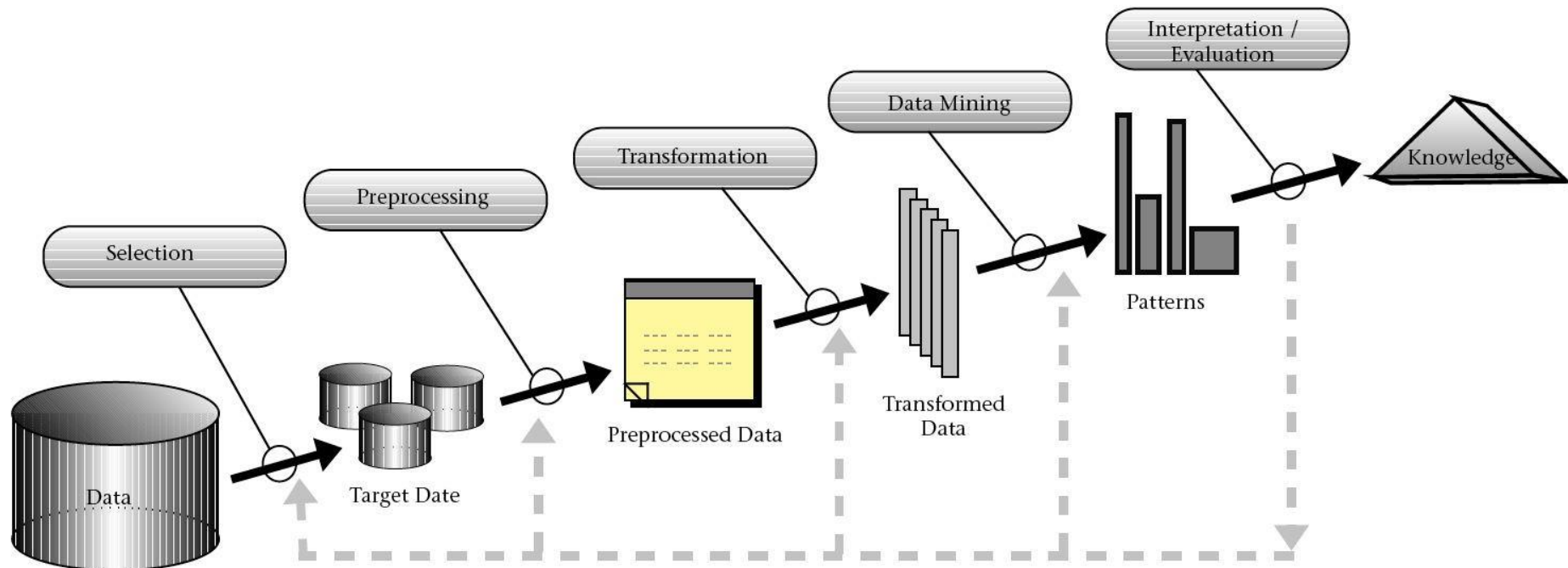
OLTP	OLAP
Garantizar la integridad de los datos y consistencia de los datos (ACID) del negocio.	Analizar los datos para dar respuesta a las necesidades del negocio.
Almacena todos los datos al mayor nivel de detalle posible.	Contiene solamente los datos que se requiere analizar, a un nivel de detalle definido.
Constantemente actualiza. Se tienen los datos más recientes.	Tiene los datos volcados hasta un momento determinado.
Orientada a gestionar transacciones (INSERT, UPDATE, DELETE).	Orientada a consultas (SELECT).
Las consultas que requieren de agrupaciones y visitas a multitud de tablas puede ser ineficientes.	Aumenta el rendimiento en consultas.
Modelo normalizado (3FN)	Modelo desnormalizado. Suele existir redundancia.

Resumen y conclusiones: Tecnologías, DW y OLAP

- Diferentes tipos de tecnologías disponibles para OLAP: MOLAP, ROLAP, HOLAP.
- En ROLAP, los DW pueden ser el apoyo para el análisis OLAP.
 - ¡OJO!. DW no es igual a OLAP. Pueden existir DW sin OLAP y viceversa.
 - No obstante, suelen estar íntimamente ligados.
- Los DW se pueden dividir en diferentes Data Marts.
- Podemos hacer uso de los SGBDR para construir DW, dando soporte a ROLAP.
 - Se usan estructuras relacionales (tablas).
 - Tienen la ventaja de poder almacenarse en el mismo sistema que las Bases de Datos relacionales que contienen los datos de origen.

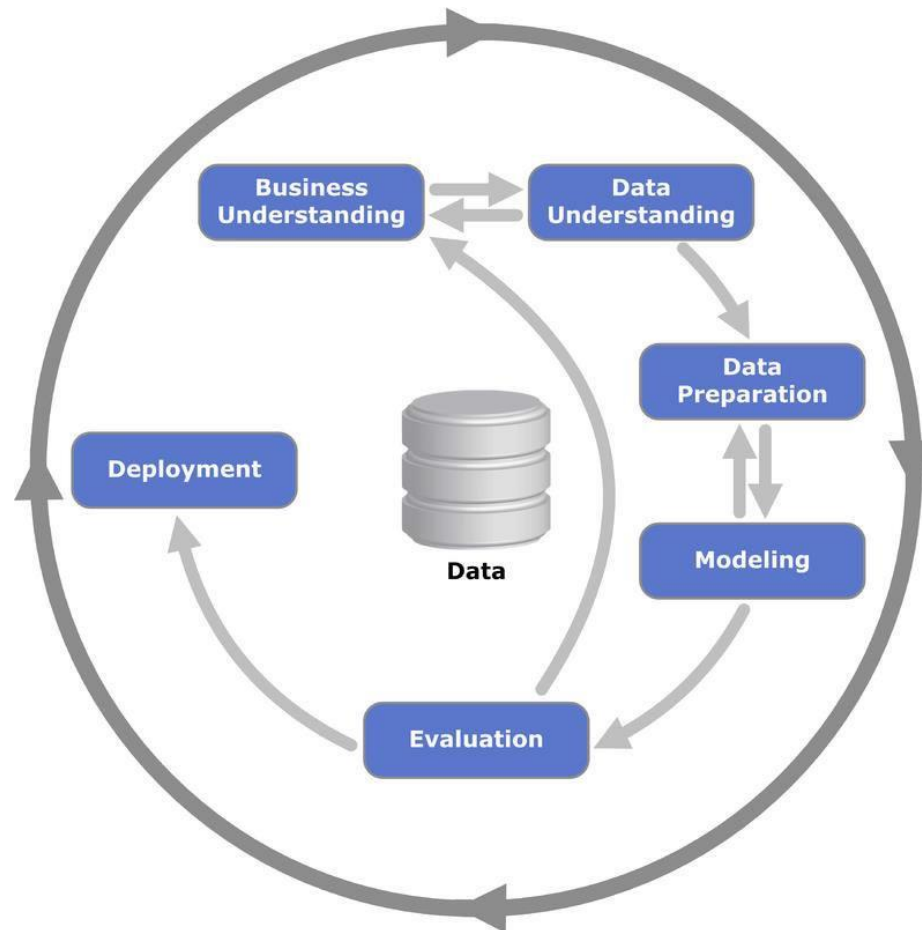
Resumen y conclusiones: ¿en dónde estamos dentro del proceso KDD?

- Proceso KDD (Knowledge Discovery on Databases):



Resumen y conclusiones: ¿en dónde estamos dentro del proceso CRISP-DM ?

- Proceso CRISP-DM (Cross Industry Standard Process for Data Mining):



OLAP: On-line Analytical Processing

Cubos OLAP.

Tecnologías OLAP.

ROLAP, data warehouse y data marts.

Comparativa OLAP vs OLTP.

Extensión SQL-OLAP y ejemplos.

Extensión OLAP-SQL

- El estándar SQL99 incluyó una serie de funciones, asociadas a la cláusula GROUP BY, para dar soporte a OLAP: CUBE, ROLLUP, GROUPING, GROUPING SETS.
- En 2003 (SQL2003), se añadieron además 34 nuevas funciones:
 - 7 funciones numéricas.
 - 16 funciones de agregación.
 - 5 funciones ventana.
 - 4 funciones para muestreados
 - 2 funciones de distribución inversa.
- Los diferentes SGBDR han ido implementando, cada uno a su ritmo, estas funciones, además de otras propias.
 - En el caso de MySQL, funciones como CUBE no se encuentran implementadas, y otras como las funciones ventana no fueron añadidas hasta verano de 2017.
 - Los siguientes ejemplos están basados en SQL Server 2014, que implementa las funciones OLAP del estándar.

Notación GROUP BY con extensión OLAP

GROUP BY <grouping spec>

<grouping spec>::=

<grouping column ref list> |

ROLLUP (<grouping column ref list>) |

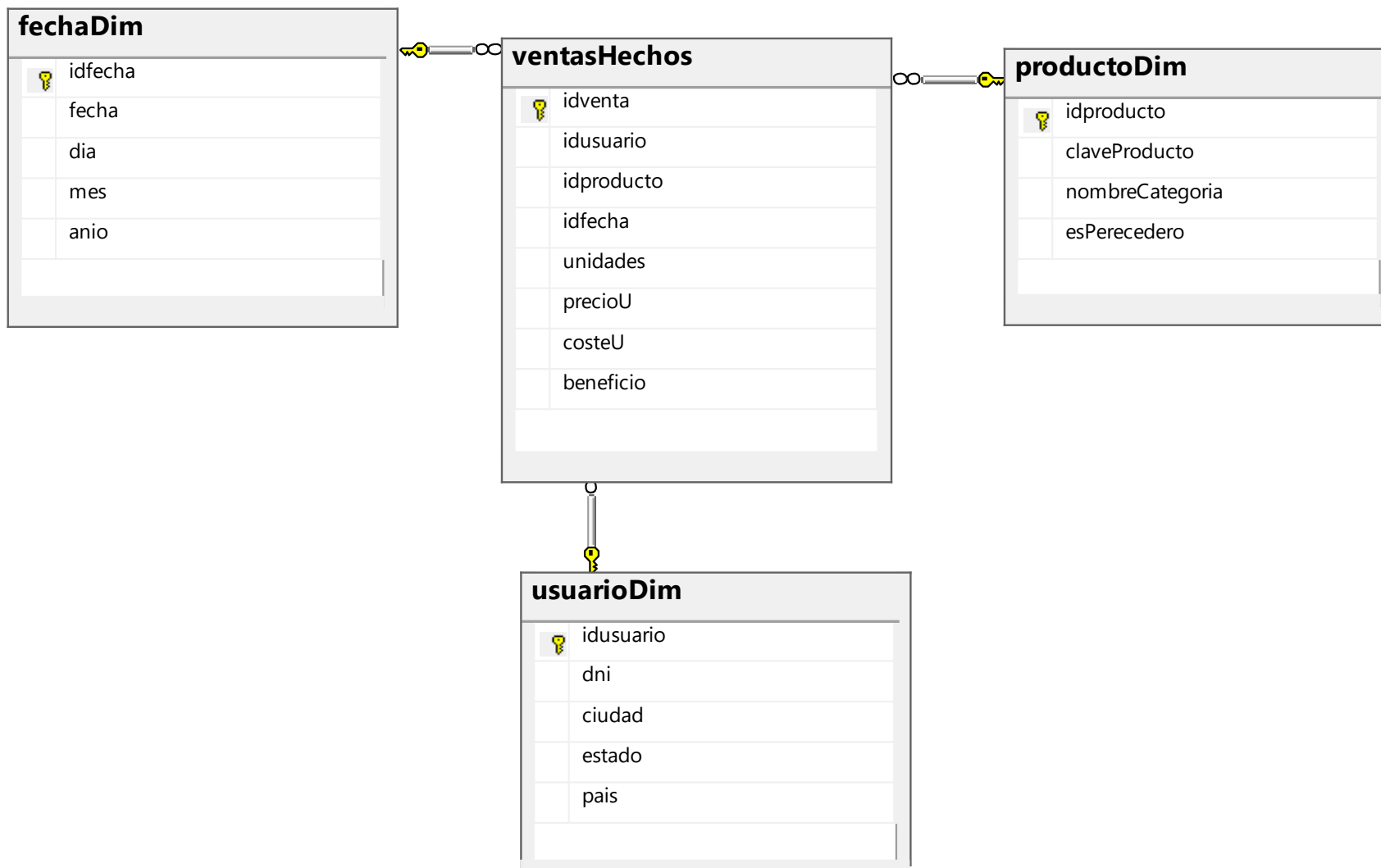
CUBE (<grouping column ref list>) |

GROUPING SETS (<grouping set list>) |

()

Columnas en la agregación

DW para los siguientes ejemplos



Ejemplo de GROUP BY “simple”

- Devolver el beneficio total obtenido por año:

```
SELECT fdim.anio, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
GROUP BY fdim.anio;
```



anio	beneficioPorAnio
2014	79
2015	155
2016	76
2017	64

Ejemplo de GROUP BY “simple”

- Devolver el beneficio total obtenido por año y categoría de producto:

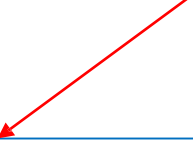
```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY fdim.anio, pDim.nombreCategoria;
```



anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28

Operador CUBE

```
GROUP BY <grouping spec>
```



```
<grouping spec>::=  
<grouping column ref list> |  
ROLLUP (<grouping column ref list>) |  
CUBE (<grouping column ref list>) |  
GROUPING SETS (<grouping set list>) |  
()
```

Añade todos los
superagregados de
las columnas del
agrupamiento.

Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año junto con el beneficio total:

```
SELECT fdim.anio, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
GROUP BY CUBE(fdim.anio);
```



anio	beneficioPorAnio
2014	79
2015	155
2016	76
2017	64
NULL	374

Los superagregados
aparecen como
NULL

Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año y categoría de producto junto con el beneficio sólo por categoría, sólo por año, y total:

```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY CUBE(fdim.anio, pDim.nombreCategoria);
```

anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
NULL	alimentación	202
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28
NULL	limpieza	172
NULL	NULL	374
2014	NULL	79
2015	NULL	155
2016	NULL	76
2017	NULL	64

Los superagregados
aparecen como
NULL

Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año y categoría de producto junto con el beneficio sólo por categoría, sólo por año, y total:

```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY CUBE(fdim.anio, pDim.nombreCategoria);
```

Beneficios por año
y categoria

anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
NULL	alimentación	202
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28
	limpieza	172
	NULL	374
	NULL	79
	NULL	155
	NULL	76
	NULL	64

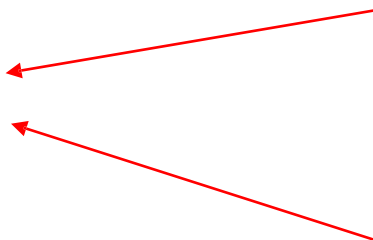
Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año y categoría de producto junto con el beneficio sólo por categoría, sólo por año, y total:

```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY CUBE(fdim.anio, pDim.nombreCategoria);
```

anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
NULL	alimentación	202
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28
	limpieza	172
	NULL	374
	NULL	79
	NULL	155
	NULL	76
	NULL	64

Beneficios por
categoria



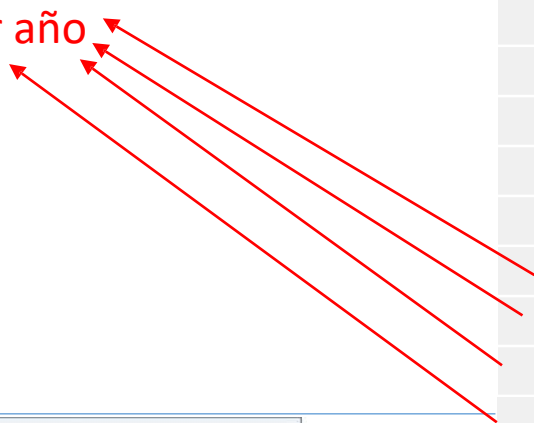
Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año y categoría de producto junto con el beneficio sólo por categoría, sólo por año, y total:

```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY CUBE(fdim.anio, pDim.nombreCategoria);
```

anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
NULL	alimentación	202
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28
	limpieza	172
	NULL	374
	NULL	79
	NULL	155
	NULL	76
	NULL	64

Beneficios por año



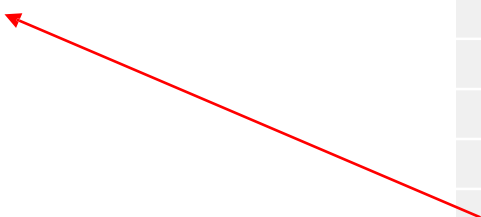
Ejemplo de GROUP BY con CUBE

- Devolver el beneficio total obtenido por año y categoría de producto junto con el beneficio sólo por categoría, sólo por año, y total:

```
SELECT fdim.anio, pDim.nombreCategoria, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
GROUP BY CUBE(fdim.anio, pDim.nombreCategoria);
```

anio	nombreCategoria	beneficioPorAnio
2014	alimentación	27
2015	alimentación	139
2017	alimentación	36
NULL	alimentación	202
2014	limpieza	52
2015	limpieza	16
2016	limpieza	76
2017	limpieza	28
	limpieza	172
	NULL	374
	NULL	79
	NULL	155
	NULL	76
	NULL	64

Beneficio total



Operador ROLLUP

```
GROUP BY <grouping spec>
```

 <grouping spec>::=

<grouping column ref list> |

ROLLUP (<grouping column ref list>) |

CUBE (<grouping column ref list>) |

GROUPING SETS (<grouping set list>) |

()

Añade los superagregados de la primera columna (o columnas) del GROUP BY

Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
M2017	NULL	NULL	64
NULL	NULL	NULL	374

Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
    INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
    INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
2017	NULL	NULL	64
NULL	NULL	NULL	374

Agregados por las 3 columnas

Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
2017	NULL	NULL	64
NULL	NULL	NULL	374

Agregados año y categoría

Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
2017	NULL	NULL	64
NULL	NULL	NULL	374

Agregados año

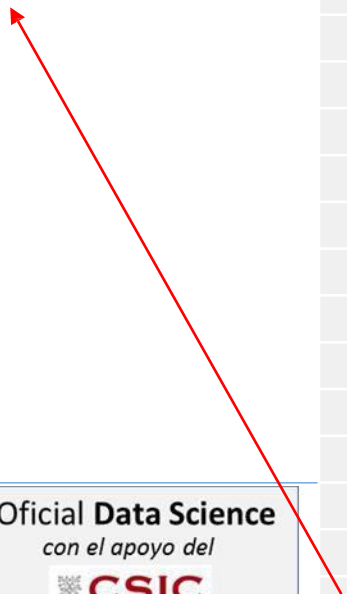
Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
2017	NULL	NULL	64
NULL	NULL	NULL	374

Total



Ejemplo de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
    INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
    INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP(fdim.anio, pDim.nombreCategoria, uDim.pais);
```

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2014	NULL	NULL	79
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2015	NULL	NULL	155
2016	limpieza	España	76
2016	limpieza	NULL	76
2016	NULL	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
2017	NULL	NULL	64
NULL	NULL	NULL	374

No hay agregados ni sólo por país, ni sólo por categoría. En todos, excepto en el total, aparece el año.

Ejemplo (otro) de GROUP BY con ROLLUP

- Devuelve los beneficios obtenidos por año, categoría de producto y país del cliente, además de los superagregado por año y categoría:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
    INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
    INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY ROLLUP((fdim.anio, pDim.nombreCategoria), uDim.pais);
```

Sólo aparecen
agregados por
año y categoría
juntos y el
total.

anio	nombreCategoria	pais	beneficioPorAnio
2014	alimentación	España	27
2014	alimentación	NULL	27
2014	limpieza	España	52
2014	limpieza	NULL	52
2015	alimentación	España	139
2015	alimentación	NULL	139
2015	limpieza	España	16
2015	limpieza	NULL	16
2016	limpieza	España	76
2016	limpieza	NULL	76
2017	alimentación	España	36
2017	alimentación	NULL	36
2017	limpieza	España	28
2017	limpieza	NULL	28
NULL	NULL	NULL	374

Operador GROUPING SETS

```
GROUP BY <grouping spec>
```

 <grouping spec>::=

<grouping column ref list> |

ROLLUP (<grouping column ref list>) |

CUBE (<grouping column ref list>) |

GROUPING SETS (<grouping set list>) |

()

Permite incluir en una sola consulta varios grupos de agregación

Ejemplo de GROUP BY con GROUPING SETS

- Devuelve los beneficios obtenidos categoría y año, y por categoria y país:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha =
vh.idfecha INNER JOIN productoDim pDim ON pDim.idproducto =
vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY GROUPING SETS((fdim.anio, pDim.nombreCategoria), (uDim.pais,
pDim.nombreCategoria))
```

Por
categoría
y país

Por año y
categoría

anio	nombreCategoria	pais	beneficioPorAnio
NULL	alimentación	España	202
NULL	limpieza	España	172
2014	alimentación	NULL	27
2015	alimentación	NULL	139
2017	alimentación	NULL	36
2014	limpieza	NULL	52
2015	limpieza	NULL	16
2016	limpieza	NULL	76
2017	limpieza	NULL	28

Ejemplo de GROUP BY con GROUPING SETS

- GROUPING SETS puede devolver los mismos resultados que CUBE si se contemplan todas las opciones:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as
beneficioPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
INNER JOIN productoDim pDim ON pDim.idproducto = vh.idproducto
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario
GROUP BY GROUPING SETS((fdim.anio, pDim.nombreCategoria, uDim.pais),
(fdim.anio, pDim.nombreCategoria), (uDim.pais, pDim.nombreCategoria),
(fdim.anio, uDim.pais), (uDim.pais), (pDim.nombreCategoria), (fdim.anio),
());
```



Vacío, para que devuelva el total

Ejemplo de GROUP BY con cláusula GROUPING

- La cláusula grouping (no confundir con grouping by) permite diferenciar los NULL de los superagregados de los NULL como valor de las columnas almacenadas:

```
SELECT fdim.anio, pDim.nombreCategoria, uDim.pais, sum(vh.beneficio) as  
beneficioPorAnio, grouping(uDim.pais) as nullPais, grouping(fdim.anio) as nullAnio  
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha =  
vh.idfecha INNER JOIN productoDim pDim ON pDim.idproducto =  
vh.idproducto  
INNER JOIN usuarioDim uDim ON uDim.idusuario = vh.idusuario  
GROUP BY GROUPING SETS((fdim.anio, pDim.nombreCategoria),  
                          (uDim.pais, pDim.nombreCategoria));
```

Un 1 indica que el valor NULL proviene de la superagregación.

anio	nombreCategoria	pais	beneficioPorAnio	nullPais	nullAnio
NULL	alimentación	España	202	0	1
NULL	limpieza	España	172	0	1
2014	alimentación	NULL	27	1	0
2015	alimentación	NULL	139	1	0
2017	alimentación	NULL	36	1	0
2014	limpieza	NULL	52	1	0
2015	limpieza	NULL	16	1	0
2016	limpieza	NULL	76	1	0
2017	limpieza	NULL	28	1	0

Funciones ventana: definición

- Operan sobre una ventana de la tabla.
- En vez de devolver una fila por cada ventana, devuelve todas las filas de la ventana, y en una nueva columna el valor agregado de cada ventana.
- Las ventanas se definen en con las cláusula OVER, que a su vez contiene las siguientes cláusulas:
 - PARTITION BY: criterio por el que se definen las ventanas (partición).
 - ORDER BY: orden de las filas dentro de la partición.
 - ROWS BETWEEN <number> AND <number> FOLLOWING: permite definir sobre que rangos de fila anteriores y posteriores a la actual se realiza la agregación.

Funciones ventana: ejemplo

```
SELECT fdim.anio,
sum(vh.beneficio) as
beneficioPorAnio

FROM fechaDim fdim INNER JOIN
ventasHechos vh ON fdim.idfecha =
vh.idfecha
GROUP BY fdim.anio;
```

anio	beneficioPorAnio
2014	79
2015	155
2016	76
2017	64

```
SELECT fdim.anio, vh.idproducto,
vh.idusuario, beneficio,
sum(vh.beneficio) OVER
(PARTITION BY fdim.anio) as
beneficioPorAnio

FROM fechaDim fdim INNER JOIN
ventasHechos vh ON fdim.idfecha
= vh.idfecha;
```

anio	idproducto	idusuario	beneficio	beneficioPorAnio
2014	4	4	24	79
2014	8	4	52	79
2014	3	4	3	79
2015	3	3	113	155
2015	7	3	26	155
2015	2	3	3	155
2015	6	3	13	155
2016	2	2	36	76
2016	6	2	24	76
2016	1	2	3	76
2016	5	2	13	76
2017	1	1	13	64
2017	5	1	15	64
2017	9	1	13	64
2017	4	1	23	64

Funciones ventana: ranking

- Las funciones de ranking pueden usarse con OVER y ORDER BY para clasificar y ordenar resultados:

```
SELECT idventa, beneficio,  
rank() over (order by beneficio desc) as rank, dense_rank()  
over (order by beneficio desc) as dense_rank, cume_dist()  
over (order by beneficio desc) as cume_dist FROM  
ventasHechos  
ORDER BY beneficio desc;
```

idventa	beneficio	rank	dense_rank	cume_dist
3	113	1	1	0,06666667
8	52	2	2	0,13333333
2	36	3	3	0,2
7	26	4	4	0,26666667
6	24	5	5	0,4
4	24	5	5	0,4
5	15	7	6	0,53333333
15	15	7	6	0,53333333
1	13	9	7	0,66666667
9	13	9	7	0,66666667
10	7	11	8	0,86666667
13	7	11	8	0,86666667
14	7	11	8	0,86666667
11	3	14	9	1
12	3	14	9	1

Funciones ventana: ranking

- También se pueden hacer rankings por partición/ventana

```
SELECT idventa, nombreCategoria, beneficio,  
rank() over (partition by categoria order by beneficio desc) as rank, dense_rank() over  
(partition by categoria order by beneficio desc) as dense_rank, cume_dist() over  
(partition by categoria order by beneficio desc) as cume_dist FROM ventasHechos vh INNER  
JOIN productoDIM pd on vh.idproducto = pd.idproducto  
ORDER BY nombreCategoria, beneficio desc;
```

idventa	categoria	beneficio	rank	dense_rank	cume_dist
3	alimentación	113	1	1	0,16666667
7	alimentación	26	2	2	0,33333333
4	alimentación	24	3	3	0,5
9	alimentación	13	4	4	0,66666667
13	alimentación	7	5	5	0,83333333
12	alimentación	3	6	6	1
8	limpieza	52	1	1	0,11111111
2	limpieza	36	2	2	0,22222222
6	limpieza	24	3	3	0,33333333
15	limpieza	15	4	4	0,55555556
5	limpieza	15	4	4	0,55555556
1	limpieza	13	6	5	0,66666667
10	limpieza	7	7	6	0,88888889
14	limpieza	7	7	6	0,88888889
11	limpieza	3	9	7	1

Funciones ventana: más ejemplos

- Las funciones ventana pueden usarse con casi cualquier función agregada (COUNT, SUM, AVG, MAX, MIN, EVERY, SOME, ANY...)

```
SELECT fdim.anio, vh.idproducto, vh.idusuario, beneficio, sum(vh.beneficio)
      OVER (PARTITION BY fdim.anio) as sumaPorAnio, avg(vh.beneficio)
      OVER (PARTITION BY fdim.anio) as mediaPorAnio, count(*) OVER
      (PARTITION BY fdim.anio) as transaccionesPorAnio,
      max(vh.beneficio) OVER (PARTITION BY fdim.anio) as maxPorAnio,
      min(vh.beneficio) OVER (PARTITION BY fdim.anio) as minPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha;
```

anio	idproducto	idusuario	beneficio	sumaPorAnio	mediaPorAnio	transacPorAnio	maxPorAnio	minPorAnio
2014	4	4	24	79	26,3333	3	52	3
2014	8	4	52	79	26,3333	3	52	3
2014	3	4	3	79	26,3333	3	52	3
2015	3	3	113	157	39,25	4	113	3
2015	7	3	26	157	39,25	4	113	3
2015	2	3	3	157	39,25	4	113	3
2015	6	3	15	157	39,25	4	113	3
2016	2	2	36	74	18,5	4	36	7
2016	6	2	24	74	18,5	4	36	7
2016	1	2	7	74	18,5	4	36	7
2016	5	2	7	74	18,5	4	36	7
2017	1	1	13	48	12	4	15	7
2017	5	1	15	48	12	4	15	7
2017	9	1	13	48	12	4	15	7
2017	4	1	7	48	12	4	15	7

Funciones ventana: más ejemplos

- Podemos usar las funciones ventana para agregar por diferentes campos y mostrar informes completos:

```
SELECT fdim.anio, pd.categoria, ud.regionUsuario, vh.idproducto, vh.idusuario, beneficio,  
avg(vh.beneficio) OVER (PARTITION BY fdim.anio) as mediaPorAnio,  
avg(vh.beneficio) OVER (PARTITION BY pd.categoria) as mediaPorCat,  
avg(vh.beneficio) OVER (PARTITION BY ud.regionUsuario) as mediaPorEst  
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha  
INNER JOIN productoDim pd ON pd.idproducto=vh.idproducto  
INNER JOIN usuarioDim ud ON ud.idusuario=vh.idusuario;
```

anio	categoria	estado	idproducto	idusuario	beneficio	mediaPorAnio	mediaPorCat	mediaPorEst
2017	limpieza	Cantabria	1	1	13	12	19,1111	23,25
2016	limpieza	Cantabria	1	2	7	18,5	19,1111	23,25
2016	limpieza	Cantabria	2	2	36	18,5	19,1111	23,25
2015	limpieza	Cantabria	2	3	3	39,25	19,1111	23,25
2015	alimentación	Cantabria	3	3	113	39,25	31	23,25
2014	alimentación	La Rioja	3	4	3	26,3333	31	26,3333
2014	alimentación	La Rioja	4	4	24	26,3333	31	26,3333
2017	alimentación	Cantabria	4	1	7	12	31	23,25
2017	limpieza	Cantabria	5	1	15	12	19,1111	23,25
2016	limpieza	Cantabria	5	2	7	18,5	19,1111	23,25
2016	limpieza	Cantabria	6	2	24	18,5	19,1111	23,25
2015	limpieza	Cantabria	6	3	15	39,25	19,1111	23,25
2015	alimentación	Cantabria	7	3	26	39,25	31	23,25
2014	limpieza	La Rioja	8	4	52	26,3333	19,1111	26,3333
2017	alimentación	Cantabria	9	1	13	12	31	23,25

Funciones ventana: más ejemplos

- También se pueden usar funciones de agregación dentro de la cláusula OVER. En este caso, es necesario usar el ORDER BY.

```
SELECT fdim.anio, pdim.categoria, sum(vh.beneficio) as beneficioAnioCategoria,
sum(sum(vh.beneficio)) OVER (PARTITION BY pdim.categoria ORDER BY fdim.anio) as
sumaPorAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha =
vh.idfecha INNER JOIN productoDim pdim ON pdim.idproducto = vh.idproducto
GROUP BY fdim.anio, pdim.categoria;
```

anio	categoria	beneficioAnioCategoria	sumaPorAnio
2014	limpieza	52	52
2015	limpieza	18	70
2016	limpieza	74	144
2017	limpieza	28	172
2014	alimentación	27	27
2015	alimentación	139	166
...

→ Beneficio limpieza en 2014

→ Beneficio alimentación en 2014 + 2015

→ Beneficio alimentación en 2014 + 2015 + 2016

...

Funciones ventana: más ejemplos

- También se pueden usar funciones de agregación dentro de la cláusula OVER. En este caso, es necesario usar el ORDER BY.

```
SELECT fdim.anio, pdim.categoria, sum(vh.beneficio) as beneficioAnioCat,  
sum(sum(vh.beneficio)) OVER (PARTITION BY pdim.categoria ORDER BY fdim.anio) as sumaPorAnio,  
avg(sum(vh.beneficio)) OVER (PARTITION BY pdim.categoria ORDER BY fdim.anio) as avgPorAnio,  
max(sum(vh.beneficio)) OVER (PARTITION BY pdim.categoria ORDER BY fdim.anio) as maxPorAnio FROM  
fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha  
INNER JOIN productoDim pdim ON pdim.idproducto = vh.idproducto  
GROUP BY fdim.anio, pdim.categoria;
```

Máximo encontrado “hasta el momento” por año en cada categoría

anio	categoria	beneficioAnioCat	sumaPorAnio	avgPorAnio	maxPorAnio
2014	alimentación	27	27	27	27
2015	alimentación	139	166	83	139
2017	alimentación	20	186	62	139
2014	limpieza	52	52	52	52
2015	limpieza	18	70	35	52
2016	limpieza	74	144	48	74
2017	limpieza	28	172	43	74

Funciones ventana: más ejemplos

- E incluso las funciones de agregación pueden aparecer en el ORDER BY de las funciones ventana:

```
SELECT fdim.anio, sum(vh.beneficio) as  
beneficioAnio, rank() OVER (ORDER BY  
sum(vh.beneficio)) as rank  
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha  
= vh.idfecha  
GROUP BY fdim.anio  
ORDER BY rank;
```

anio	beneficioAnio	rank
2017	96	1
2016	174	2
2015	594	3
2014	851	4

Funciones ventana: más ejemplos

- BETWEEN, PRECEDING y FOLLOWING pueden usarse para agregar solamente los valores de un subconjunto de filas anteriores y posteriores a la actual. Importante el ORDER BY para que las filas no se ordenen de forma aleatoria. La siguiente consulta devuelve el beneficio de cada venta junto con la media de las ventas inmediatamente anteriores y posteriores (ordenadas según su id):

```
SELECT vh.idventa, beneficio,  
avg(beneficio) OVER (ORDER BY vh.idventa ROWS BETWEEN 1 PRECEDING AND 1 FOLLOWING) as mediaAntPos  
FROM ventasHechos vh  
ORDER BY vh.idventa;
```

idventa	beneficio	mediaConAntPos
1	13	24,5
2	36	54
3	113	57,6666
4	24	50,6666
5	15	21
6	24	21,6666
7	26	34
8	52	30,3333
9	13	24
10	7	7,6666
11	3	4,3333
12	3	4,3333
13	7	5,6666
14	7	9,6666
15	15	11

Funciones ventana: más ejemplos

- Podrían acumularse sólo los anteriores:

```
SELECT vh.idventa, beneficio,  
avg(beneficio) OVER (ORDER BY vh.idventa ROWS BETWEEN 1 PRECEDING AND 0 FOLLOWING) as mediaAnt  
FROM ventasHechos vh  
ORDER BY vh.idventa;
```

idventa	beneficio	mediaConAnt
1	13	13
2	36	24,5
3	113	74,5
4	24	68,5
5	15	19,5
6	24	19,5
7	26	25
8	52	39
9	13	32,5
10	7	10
11	3	5
12	3	3
13	7	5
14	7	7
15	15	11

Funciones ventana: más ejemplos

- Podrían acumularse sólo los posteriores:

```
SELECT vh.idventa, beneficio,  
avg(beneficio) OVER (ORDER BY vh.idventa ROWS BETWEEN 0 PRECEDING AND 2 FOLLOWING) as mediaPos  
FROM ventasHechos vh  
ORDER BY vh.idventa;
```

idventa	beneficio	mediaConPos
1	13	24,5
2	36	74,5
3	113	68,5
4	24	19,5
5	15	19,5
6	24	25
7	26	39
8	52	32,5
9	13	10
10	7	5
11	3	3
12	3	5
13	7	7
14	7	11
15	15	15

Funciones ventana: más ejemplos

- Incluso los 2, 3... anteriores y los 2, 3... posteriores:

```
SELECT vh.idventa, beneficio,  
avg(beneficio) OVER (ORDER BY vh.idventa ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) as media2AntPos  
FROM ventasHechos vh  
ORDER BY vh.idventa;
```

idventa	beneficio	media2AntPos
1	13	54
2	36	46,5
3	113	40,2
4	24	42,4
5	15	40,4
6	24	28,2
7	26	26
8	52	24,4
9	13	20,2
10	7	15,6
11	3	6,6
12	3	5,4
13	7	7
14	7	8
15	15	9,6666

Funciones ventana: más ejemplos

- BETWEEN, PRECEDING y FOLLOWING también pueden usarse con agregados de agregados. Media sólo con anterior:

```
SELECT vh.idproducto,    sum(beneficio) as beneficioPorProd,
avg(sum(vh.beneficio)) OVER (ORDER BY idproducto ROWS BETWEEN 1 PRECEDING AND 0 FOLLOWING) as
                                                                    mediaConAnt
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
GROUP BY vh.idproducto
ORDER BY idproducto;
```

idproducto	beneficioPorProd	mediaConAnt
1	20	20
2	39	29,5
3	116	77,5
4	31	73,5
5	22	26,5
6	39	30,5
7	26	32,5
8	52	39
9	13	32,5

Funciones ventana: más ejemplos

- BETWEEN, PRECEDING y FOLLOWING también pueden usarse con agregados de agregados. Varios:

```
SELECT vh.idproducto, sum(beneficio) as beneficioPorProd,
avg(sum(vh.beneficio)) OVER (ORDER BY idproducto ROWS BETWEEN 1 PRECEDING AND 0 FOLLOWING) as mediaConAnt,
avg(sum(vh.beneficio)) OVER (ORDER BY idproducto ROWS BETWEEN 0 PRECEDING AND 1 FOLLOWING) as mediaConPos,
avg(sum(vh.beneficio)) OVER (ORDER BY idproducto ROWS BETWEEN 2 PRECEDING AND 2 FOLLOWING) as
mediaCon2Ant2Pos,
sum(sum(vh.beneficio)) OVER (ORDER BY idproducto ROWS BETWEEN 9 PRECEDING AND 0 FOLLOWING) as sumaAcumulada
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
GROUP BY vh.idproducto
ORDER BY idproducto;
```

idproducto	beneficioPorProd	mediaConAnt	mediaConPos	mediaCon2Ant2Pos	sumaAcumulada
1	20	20	29,5	58,3333	20
2	39	29,5	77,5	51,5	59
3	116	77,5	73,5	45,6	175
4	31	73,5	26,5	49,4	206
5	22	26,5	30,5	46,8	228
6	39	30,5	32,5	34	267
7	26	32,5	39	30,4	293
8	52	39	32,5	32,5	345
9	13	32,5	13	30,3333	358

Agregaciones usando expresión de tabla común (CTE)

- La siguiente consulta devuelve los beneficios por mes y acumulados por año:

```
WITH tabla_temporal AS
(SELECT fdim.mes, fdim.anio, sum(beneficio) as beneficioMesAnio
FROM fechaDim fdim INNER JOIN ventasHechos vh ON fdim.idfecha = vh.idfecha
GROUP BY fdim.mes, fdim.anio)
SELECT tt.anio, tt.mes, tt.beneficioMesAnio,
sum(tt.beneficioMesAnio) OVER (PARTITION BY tt.anio ORDER BY tt.mes) as AcumMesAnio
FROM tabla_temporal tt
ORDER BY tt.anio, tt.mes asc;
```

anio	mes	beneficioMesAnio	AcumMesAnio
2014	2	176	176
2014	3	45	221
2014	4	45	266
2014	5	135	401
2014	6	135	536
2014	7	135	671
2014	8	135	806
2014	9	45	851
...