# Universidad Tecnológica de Puebla

## Técnico Superior Universitario en Tecnologías de la Información y Comunicación


Materia: Base de Datos II

Mayo – Agosto 2015

Colaboradores:

Ricardo Salvador Martínez Castillo

Francisco Torres Chávez

Enrique Villa Ruano

# Tabla de contenido

# Unidad I. Modelo Relacional


El Modelo Entidad-Relación permite abstraer la información de la vida real para comprender como almacenar los datos, sin embargo, este modelo no se puede pasar directamente al gestor de bases de datos, es necesario convertirlo al Modelo Relacional para almacenar la información sin redundancias y que permita recuperarla fácilmente. Por lo que, el Modelo E-R se debe pasar a Modelo Relacional y éste a su vez al gestor de bases de datos.

## 1.1 Introducción

El modelo E-R se considera un modelo conceptual ya que permite a un nivel alto el ver con claridad la información utilizada en algún problema o negocio.

Primero nos concentraremos en desarrollar un buen modelo "lógico" que se conoce como "esquema de la base de datos" (*database schema*) a partir del cual se podrá realizar el modelado físico en el SGBD, es importante mencionar que es un paso necesario, no se puede partir de un modelo conceptual para realizar un físico. (Figura 1)

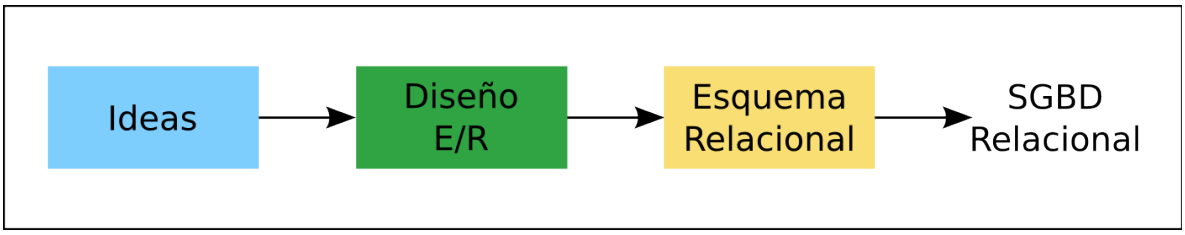



**Figura 1 Base de Datos Relacional**

## 1.2 ¿Por qué "modelo relacional"?

Puede resultar confuso el concepto de modelo Entidad-Relación versus Modelo Relacional, quizás porque ambos comparten casi las mismas palabras. Como se mencionó anteriormente, el objetivo del modelo relacional es crear un "esquema" (*schema*), lo cual como se mencionará posteriormente consiste de un conjunto de "tablas" que representan "relaciones", relaciones entre los datos.

Estas tablas, pueden ser construidas de diversas maneras:

- Creando un conjunto de tablas iniciales y aplicar operaciones de normalización hasta conseguir el esquema más óptimo. Las técnicas de normalización se explican más adelante en este capítulo.
- Convertir el diagrama e-r a tablas y posteriormente aplicar también operaciones de normalización hasta conseguir el esquema óptimo.

La primer técnica fue de las primeras en existir y, como es de suponerse, la segunda al ser más reciente es mucho más conveniente en varios aspectos:

- El partir de un diagrama visual es muy útil para apreciar los detalles, de ahí que se llame modelo conceptual.
- El crear las tablas iniciales es mucho más simple a través de las reglas de conversión.
- Se podría pensar que es lo mismo porque finalmente hay que "normalizar" las tablas de todas formas, pero la ventaja de partir del modelo e-r es que la "normalización" es mínima por lo general.
- Lo anterior tiene otra ventaja, aún cuando se normalice de manera deficiente, se garantiza un esquema aceptable, en la primer técnica no es así.

## 1.3 Conceptos básicos

### 1.3.1 Tablas

El modelo relacional proporciona un manera simple de representar los datos: una tabla bidimensional llamada relación, (figura 2).

| Título | Año | duración | Tipo |
|---|---|---|---|
| Star Wars | 1977 | 124 | color |
| Mighty Ducks | 1991 | 104 | color |
| Wayne's World | 1992 | 95 | color |

**Tabla 1 Relación Películas**

La relación Películas tiene la intención de manejar la información de las instancias en la entidad Películas, cada renglón corresponde a una entidad película y cada columna corresponde a uno de los atributos de la entidad. Sin embargo las relaciones pueden representar más que entidades, como se explicará más adelante.

#### 1.3.2 Atributos

Los atributos son las columnas de un relación y describen características particulares de ella.

### 1.3.3 Esquemas

Es el nombre que se le da a una relación y el conjunto de atributos en ella.

| Películas (título, año, duración, tipo) |
|---|

En un modelo relación, un diseño consiste de uno o más esquemas, a este conjunto se le conoce como "esquema relacional de base de datos" (relational database schema) o simplemente "esquema de base de datos" (database schema).

### 1.3.4 Tuplas

Cada uno de los renglones en una relación conteniendo valores para cada uno de los atributos.

| (Star Wars, 1977, 124, color) |
|---|

### 1.3.5 Dominios

Se debe considerar que cada atributo (columna) debe ser atómico, es decir, que no sea divisible, no se puede pensar en un atributo como un "registro" o "estructura" de datos.

### 1.3.6 Representaciones equivalentes de una relación

Las relaciones son un conjunto de tuplas, no una lista de tuplas. El orden en que aparecen las tuplas es irrelevante. Así mismo el orden de los atributos tampoco es relevante. (figura 3)

| Año | Título | Tipo | Duración |
|---|---|---|---|
| 1991 | Mighty Ducks | color | 104 |
| 1992 | Wayne's World | color | 95 |
| 1977 | Star Wars | color | 124 |

Tabla 2 Relación Películas (segunda representación)

## 1.4 Conversión del modelo E-R a un esquema de base de datos (Conversión a tablas)

El modelo es una representación visual que gráficamente nos da una perspectiva de como se encuentran los datos involucrados en un proyecto u organización.

Pero el modelo no nos presenta propiamente una instancia de los datos, un ejemplo que muestre con claridad algunas datos de muestra y como se relacionan en realidad. Por eso es conveniente crear un "esquema", el cual consiste de tablas las cuales en sus renglones (tuplas) contienen instancias de los datos.

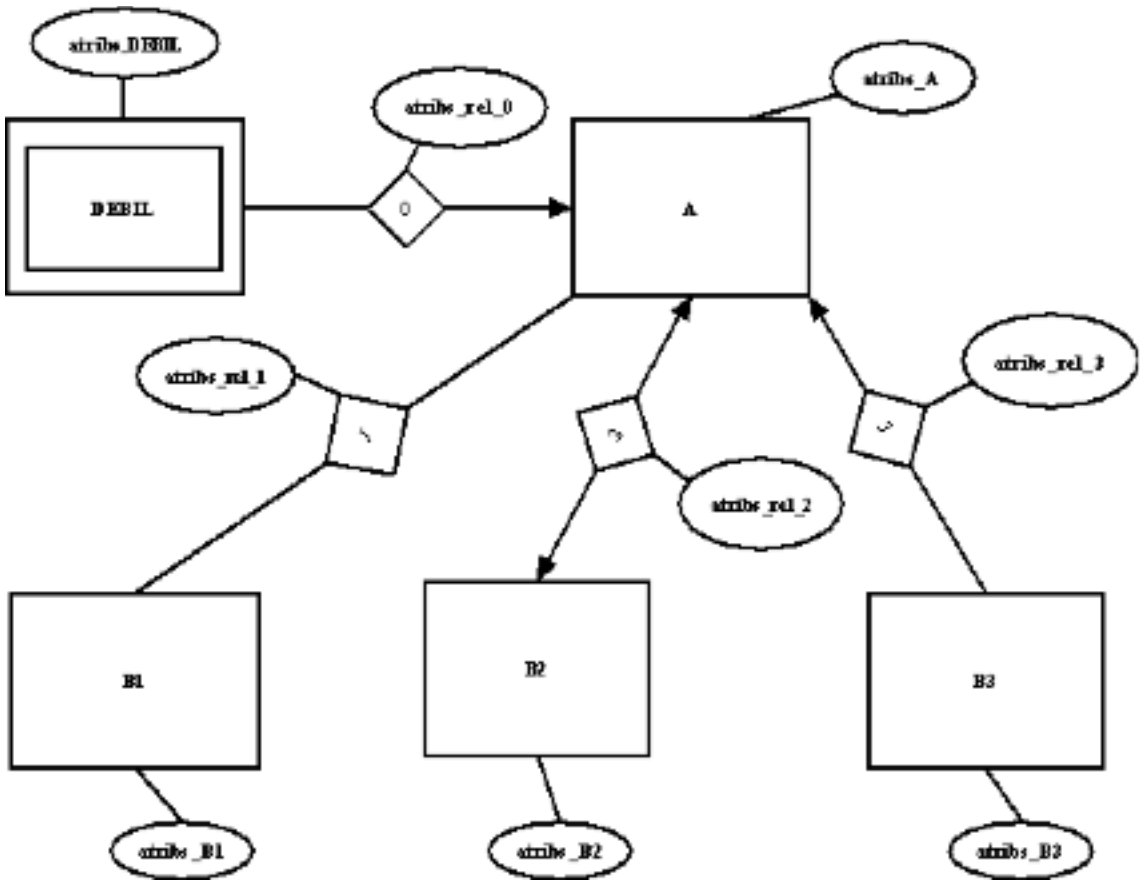
### 1.4.1 Conversión a tablas desde un modelo

Las tablas siguientes muestran las reglas que se deben seguir para poder crear dicho esquema.

**Modelo E-R conversión a tablas**

- Una tabla por cada conjunto de entidadeso nombre de tabla = nombre de conjunto de entidades
- Una tabla por cada conjunto de relaciones m-mo nombre de tabla = nombre de conjunto de relaciones
- Definición de columnas para cada tabla
  - Conjuntos fuertes de entidades
    - columnas = nombre de atributos
  - Conjuntos débiles de entidades
    - columnas = llave_primaria (dominante) U atributos (subordinado)
  - Conjunto de relaciones R (m-m) entre A, B
    - columnas (R) = llave_primaria (A) U llave_primaria (B) U atributos(R)
  - Conjunto de relaciones R (1-1) entre A y B
    - columnas (A) = atributos (A) U llave primaria (B) U atributos(R)
  - Conjunto de relaciones R (1-m) entre A y B
    - columnas (B) = atributos (B) U llave primaria (A) U atributos

(R)




El diagrama anterior se convierte al siguiente esquema:

Débil (Atributos_Debil, LlavePrimaria_A, Attributos_Relación_0)

A (LLavePrimaria_A, Attributos_A)

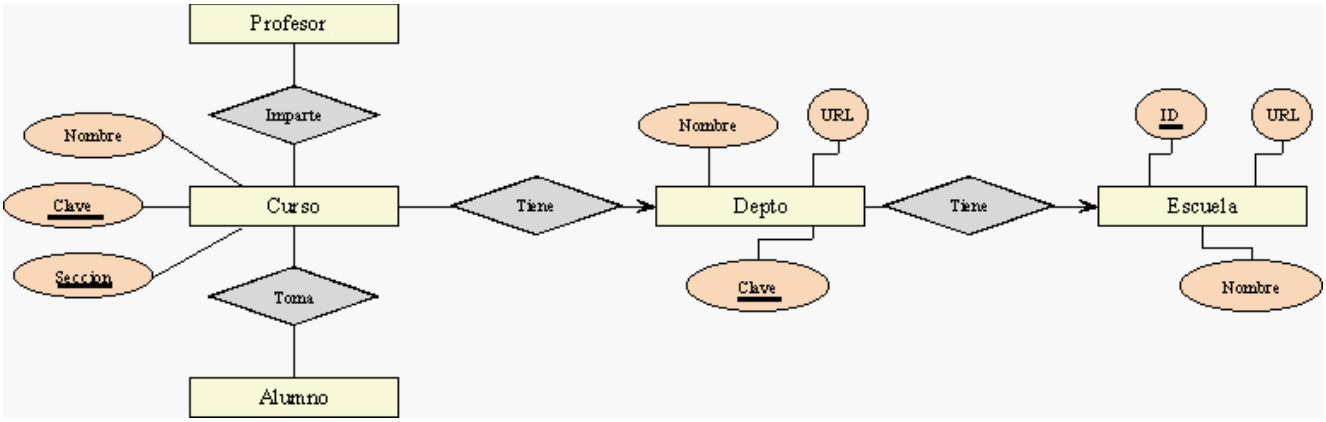
B1 (LlavePrimaria_B1, Atributos_B1)

B2 (LlavePrimaria_B2, LlavePrimaria_A, AtributosRelación_2)

B3 (LlavePrimaria_B3, LlavePrimaria_A, AtributosRelación_3)

A_B1 (LlavePrimaria_A, LlavePrimariaB, AtributosRelación_1)

Ejemplo:




Para el ejemplo de la figura tendríamos el esquema:

Escuela (id, url, nombre)

Departamento (clave, url, nombre, id_escuela)

Curso (clave, sección, nombre, clave_depto)

Profesor (id, nombre, extensión)

Estudiante (id, nombre, carrera)

Profesor_curso(id_prof, clave_curso, sección_curso)

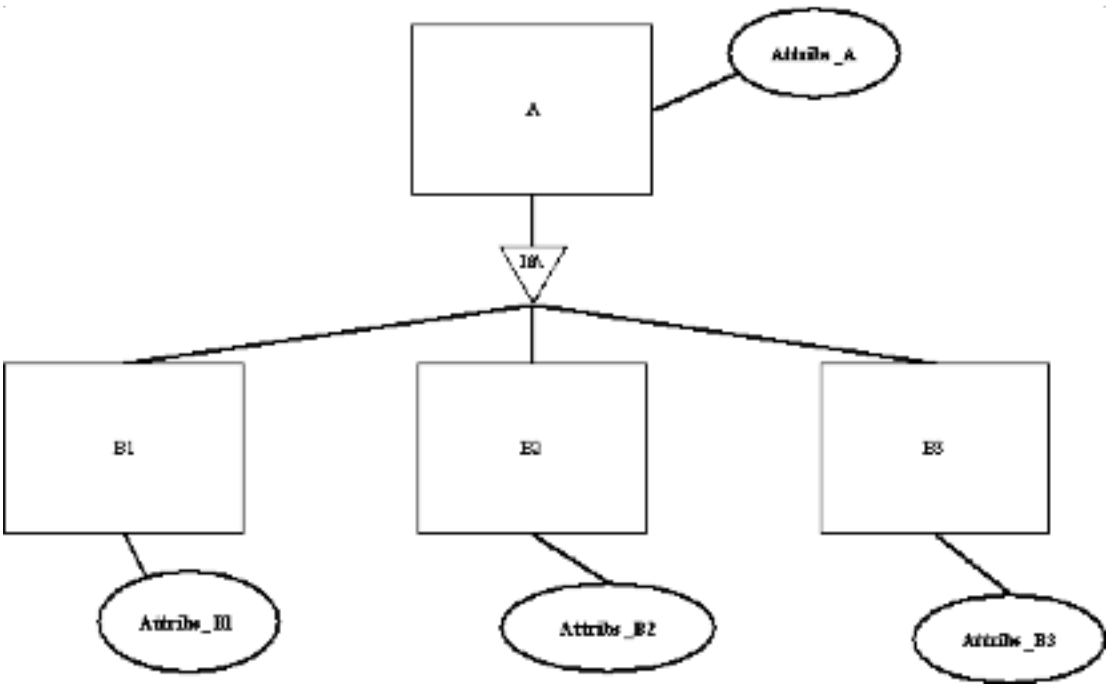
Estudiante_curso (id_estud, clave_curso, sección_curso)

## 1.4.2 Conversión a tablas desde un modelo con generalización

El modelo E-R de generalización a tablas dos posibilidades:

1. Posibilidad número 1
   - Crear una tabla para el conjunto de entidades A de mayor nivel
     - columnas (A) = atributos(A)
   - Para cada conjunto de entidades B de menor nivel, crear una tabla tal

que:
    - columns (B) = atributos (B) U llave_primaria (A)
2. Posibilidad número 2
    - si A es un conjunto de entidades de mayor nivel para cada conjunto de entidades B de menor nivel, crear una tabla tal que:
        - columnas (B) = atributos (B) U atributos (A)




La generalización se convierte al siguiente esquema:

Posibilidad 1

A (LlavePrimaria_A, Atributos_A)

B1 (Atributos_B1, LlavePrimaria_A)

B2 (Atributos_B2, LlavePrimaria_A)

B3 (Atributos_B3, LlavePrimaria_A)

Posibilidad 2

| |
|---|
| B1 (Atributos_B1, LlavePrimaria_A, Atributos_A) |
| B2 (Atributos_B2, LlavePrimaria_A, Atributos_A) |
| B3 (Atributos_B3, LlavePrimaria_A, Atributos_A) |

Es importante mencionar que a pesar de que existen 2 métodos para convertir una generalización a tablas, no hay una regla exacta de cual usar en determinado caso. A continuación se mencionan algunos consejos útiles para la determinación de cual método emplear:

3. Si la entidad de nivel superior está relacionada con otra(s) entidades puede sugerirse emplear el método (1) ya que de esa manera la tabla (A) será la única involucrada en la relación, de otra forma se tendrían tres tablas (B1,B2 y B3) formando parte de la relación
4. Es importante tomar en cuenta la pertenencia de instancias, si se considera que hablamos de una generalización disjunta, donde no se puede pertenecer a varias entidades de nivel inferior, quizás sea recomendable el método (1), en otro caso se podría pensar en el método (2).
5. También es importante analizar ambos casos con respecto a las "consultas" que se deseen realizar ya que esto también determina en muchos casos el método a emplear.

### 1.4.3 Descubrimiento de llaves en las relaciones

Las llaves resultantes en las relaciones de un esquema se pueden inferir de la siguiente manera:

1) Cada tabla que provenga de una entidad contiene por si misma una llave
2) Para las tablas resultado de una relación se toman las llaves primarias de ambas entidades y éstas conforman la nueva llave primaria, excepto en un caso como el que sigue:

**FALTA IMAGEN**

Podemos observar que existe una relación m-m entre "actor" y "serie",

demostrando que un actor puede actuar en muchas series y que muchas series tendrán a los mismos actores.

La tabla a crear actor_serie es la siguiente:

| id_actor | id_serie | id_personaje |
|---|---|---|
| Joaquín Pardavé | Génesis | Abel hermano bueno |
| Evita Muñoz (Chachita) | Qué bonita familia | Dulce abuelita |
| Joaquín Pardavé | Génesis | Caín hermano malo |
| Evita Muñoz (Chachita) | Hermelinda linda | Bruja Hermelinda |

**Tabla 3 Tabla de Autor**

si se considera "id_actor, id_serie" como la llave primaria caemos en un problema porque esa combinación no identifica de manera única a la tupla, como es el caso de "Joaquín Pardavé, Génesis" ya que en la primer tupla tenemos que determina a "Abel hermano bueno" y en la tercera a "Caín hermano malo".

La relación es correcta ya que un actor puede representar a varios personajes en la misma obra, pero entonces la llave "id_actor, id_serie" no es la correcta y en este caso lo más recomendable sería emplear los tres atributos de la relación "id_actor, id_serie, id_personaje".

# Unidad II. Diccionario de Datos


El diccionario de datos también es conocido como catálogo del sistema y en éste debemos almacenar:

- El nombre de las relaciones.
- El nombre de los atributos de cada relación.
- El dominio y la longitud de los atributos.
- El nombre de las vistas definidas en la base de datos, y la definición de esas vistas.
- Las restricciones de integridad.

## 2.1 Definición de un diccionario de datos

Un diccionario de datos es un conjunto de metadatos[1] que contiene las características lógicas de los datos que se van a utilizar en el sistema que se programa, incluyendo nombre, descripción, alias, contenido y organización.

Estos diccionarios se desarrollan durante el análisis de flujo de datos y ayuda a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño del proyecto.

Identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información, se desarrolla durante el análisis de flujo de datos y auxilia a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño.

En un diccionario de datos se encuentra la lista de todos los elementos que forman parte del flujo de datos de todo el sistema. Los elementos más importantes son flujos de datos, almacenes de datos y procesos. El diccionario de datos guarda los detalles y descripción de todos estos elementos.

Ejemplos

```
Nombre = Título + Primer-nombre + Apellido-paterno + Apellido-materno

Título = [ Sr | Sra | Dr | Ing]

Primer-nombre = {caracter}

Apellido-paterno = {caracter}

Apellido-materno = {caracter}

caracter = [A-Z|a-z| |’] a
```

1 Los metadatos son datos que definen datos, es decir, al crear una tabla y especificar los campos, sus tipos de datos, llaves primarias, llaves foráneas, etc., se está haciendo la definición de la estructura de la tabla y a su vez define lo que va a contener.

Contiene las características lógicas de los sitios donde se almacenan los datos del sistema, incluyendo nombre, descripción, alias, contenido y organización. Identifica los procesos donde se emplean los datos y los sitios donde se necesita el acceso inmediato a la información, se desarrolla durante el análisis de flujo de datos y auxilia a los analistas que participan en la determinación de los requerimientos del sistema, su contenido también se emplea durante el diseño.

## 2.2 Razones para su utilización

Las principales razones para utilizar un diccionario de datos se enlistan a continuación:

1. Para manejar los detalles en sistemas muy grandes, ya que tienen enormes cantidades de datos, aun en los sistemas más chicos hay gran cantidad de datos. Los sistemas al sufrir cambios continuos, es muy difícil manejar todos los detalles. Por eso se registra la información, ya sea sobre hoja de papel o usando procesadores de texto. Los analistas mas organizados usan el diccionario de datos automatizados diseñados específicamente para el análisis y diseño de software.
2. Para asignarle sólo un significado a cada uno de los elementos y actividades del sistema. Los diccionarios de datos proporcionan asistencia para asegurar significados comunes para los elementos y actividades del sistema y registrando detalles adicionales relacionados con el flujo de datos en el sistema, de tal manera que todo pueda localizarse con rapidez.
3. Para documentar las características del sistema, incluyendo partes o componentes así como los aspectos que los distinguen. También es necesario saber bajo qué circunstancias se lleva a cabo cada proceso y con qué frecuencia ocurren. Produciendo una comprensión más completa. Una vez que las características están articuladas y registradas, todos los participantes en el proyecto tendrán una fuente común de información con respecto al sistema.
4. Para facilitar el análisis de los detalles con la finalidad de evaluar las características y determinar donde efectuar cambios en el sistema.

Determina si son necesarias nuevas características o si están en orden los cambios de cualquier tipo. Se abordan las siguientes características:

- Naturaleza de las transacciones: las actividades de la empresa que se llevan a cabo mientras se emplea el sistema.
- Preguntas: solicitudes para la recuperación o procesamiento de información para generar una respuesta específica.
- Archivos y bases de datos: detalles de las transacciones y registros maestros que son de interés para la organización.
- Capacidad del sistema: Habilidad del sistema para aceptar, procesar y almacenar transacciones y datos

5. Localizar errores y omisiones en el sistema, detectan dificultades, y las presentan en un informe. Aun en los manuales, se revelan errores.

## 2.3 Contenido de un registro del diccionario

El diccionario tiene dos tipos de descripciones para el flujo de datos del sistema, son los elementos datos y estructura de datos.

- **Elemento dato**: son los bloques básicos para todos los demás datos del sistema, por si mismos no le dan un significado suficiente al usuario. Se agrupan para formar una estructura de datos. Se combinan varios elementos de datos para hacer los records o "data structures". Ejemplo: nombre, dirección, seguro social.
- **Descripción**: Cada entrada en el diccionario consiste de un conjunto de detalles que describen los datos utilizados o producidos por el sistema. Cada uno está identificado con:
  - Nombre: para distinguir un dato de otro.
  - Descripción: indica lo que representa en el sistema.
  - Alias: porque un dato puede recibir varios nombres, dependiendo de quién uso este dato.
  - Longitud: porque es de importancia de saber la cantidad de espacio necesario para cada dato.
  - Valores de los datos: porque en algunos procesos solo son permitidos valores muy específicos para los datos. Si los valores de los datos están restringidos a un intervalo especifico, esto debe estar en la entrada del diccionario.

- **Estructura de datos:** es un grupo de datos que están relacionados con otros y que en conjunto describen un componente del sistema. También se conocen como record. Es la combinación de elementos de datos relacionados que se incluye en un flujo de datos o se retiene en un "*data store*".
- **Descripción:** Se construyen sobre cuatro relaciones de componentes.
- **Relación secuencial:** define los componentes que siempre se incluyen en una estructura de datos.
    - Relación de selección: (uno u otro), define las alternativas para datos o estructuras de datos incluidos en una estructura de datos.
    - Relación de iteración: (repetitiva), define la repetición de un componente.
    - Relación opcional: los datos pueden o no estar incluidos, o sea, una o ninguna iteración.
- **Notación:** Los analistas usan símbolos especiales con la finalidad de no usar demasiada cantidad de texto para la descripción de las relaciones entre datos y mostrar con claridad las relaciones estructurales. En algunos casos se emplean términos diferentes para describir la misma entidad (alias) estos se representan con un signo igual (=) que vincula los datos.

El diccionario de datos guarda y organiza los detalles del Diagrama de Flujo de Datos (DFD). Es el segundo componente del análisis estructurado. También se conoce como "*Data Repository*". Incluye el contenido de los flujos de datos (*data flow*), los almacenamientos de datos (*data store*), las entidades externas y los procesos.

## 2.4 Documentación.

Datos de los elementos (*Data elements*). Las características que se describen en el diccionario de datos son:

1. ***Name*:** Es el nombre del elemento de datos; debe ser significativo.
2. ***Alias:*** Cualquier otro nombre que se pueda usar para referirse al elemento de datos. Por ejemplo, el nombre de un elemento de datos puede ser Balance actual, y el alias puede ser Deuda. Solo se incluye el alias si

realmente es necesario utilizarlo.

3. ***Type* (tipo) y *Size* (tamaño):** tipo se refiere a si el elemento de datos contiene valor numérico, caracteres o alfabético. Tamaño se refiere al máximo de caracteres o de dígitos que puede tener el elemento de datos.
4. ***Output format or edit mask:*** Indica cómo se presenta el dato al mostrarse en pantalla o al imprimirse en un reporte. Por ejemplo, el número de teléfono del cliente se puede guardar en el disco usando solo números 7878889999, pero presentarse editado en la pantalla o en el reporte (787) 888-9999.

6. ***Default value*:** Es el valor que el elemento de datos tiene si no se cambia entrando otro valor.

7. ***Prompt, column header or field caption*** - Es el nombre que se presenta en la pantalla o el título del dato en el reporte.

8. ***Source:*** De dónde se origina el valor del elemento de datos. Puede ser una forma, un departamento, otro sistema, etc.

9. ***Security:*** Identifica los individuos o departamentos que pueden modificar el elemento de datos. Por ejemplo, la línea de crédito puede ser cambiada por el gerente de crédito.

10. ***Responsible user(s):*** Identifica el (los) usuarios responsables de entrar o cambiar los valores del elemento de datos.

11. ***Acceptable Data and Data validation:*** Se especifica el dominio o valores permitidos. Pueden ser valores específicos, una lista de valores, los valores que se encuentren en otro archivo, etc. El valor puede tener reglas de validación; por ejemplo, el salario debe estar entre lo permitido para la posición que el empleado ocupa.

12. ***Derivation formula:*** Si el valor es el resultado de un cálculo, se muestra la fórmula que se utiliza.

13. ***Description or comments:*** Para proveer información adicional, notas o

descripciones.

Flujo de datos (*Data flows*). Las características que se describen en el flujo de datos son:

1. ***Name:*** El nombre del flujo de datos tal y como aparece en el DFD.

5. ***Alias:*** Otro nombre con que se conozca el flujo de datos.

6. ***Abbreviation or ID:*** Código que provee acceso rápido al flujo de datos en un diccionario de datos automatizado.

7. ***Description:*** Describe el flujo de datos y su propósito.

8. ***Origin:*** De donde sale (la fuente) el flujo de datos. Puede ser un proceso, un “data store” o una entidad.

9. ***Destination:*** El punto final del flujo de datos en el DFD. Puede ser un proceso, un “data store” o una entidad.

10. ***Record:*** Cada flujo de datos representa un grupo de elementos de datos relacionados, o un record. Los records y los flujos de datos se definen por separado para que más de un flujo de datos o “data store” pueda hacer referencia al mismo record.

11. ***Volume and frequency:*** Describe el número esperado de ocurrencias para el flujo de datos por unidad de tiempo.

Almacenamiento de datos (*Data Store*). Las características que se describen en el “data store” son:

1. ***Name:*** El nombre del “data store” según aparece en el DFD.

2. ***Alias:*** Otro nombre con el que se pueda llamar al “data store”.

3. ***Abbreviation or ID:*** Código que provee un acceso rápido al “data store” en un diccionario de datos automatizado.

4. ***Description:*** Describe el "*data store*" y su propósito.

5. ***Input data flows:*** Los nombres de los flujos de datos que entran al "data store".

6. ***Output data flows:*** Los nombres de los flujos de datos que salen del "data store".

7. ***Record:*** El nombre del record en el diccionario de datos para el "data store".

8. ***Volume and Frequency:*** El número estimado de records guardados en el "data store", al igual que el aumento o cambio esperado.

Proceso (*Process*). Se documenta cada función primitiva. Se incluye:

1. ***Process name or label:*** El nombre del proceso como aparece en el DFD.
2. ***Purpose or description:*** Un resumen del propósito general del proceso. Los detalles se documentan en el Process Description.
3. ***Process number:*** Número de referencia que identifica el proceso y su relación con los niveles del sistema.
4. ***Input data flows:*** Los nombres de los flujos de datos que entran al proceso.
5. ***Output data flows:*** Los nombres de los flujos de datos que salen del proceso.
6. ***Process Description:*** Se explican los detalles del proceso.
    - Entidades Externas – Las características que se describen son:
        - ***Name:***
        - ***Alias:***
        - ***Description:*** Describe a la entidad y su propósito.
        - ***Input data flow:***
        - ***Output data flow:***
    - ***Record:*** Se describe lo siguiente:

- ***Record name:***
- ***Alias:***
- ***Description:***
- ***Record content or composition:*** Una lista de todos los elementos de datos incluidos en el record. Se identifica cualquier “key” primario, o sea un elemento de datos en el record que identifica en forma única al record.

# Unidad III. Restricciones de las Bases de Datos


Las restricciones de integridad garantizan que las modificaciones realizadas por los usuarios no alteren la consistencia de la base de datos, es decir, se debe asegurar que no se dañen accidentalmente los datos que se encuentran almacenados en la base de datos.

## 3.1 Restricciones de Integridad

Considere el ejemplo que se muestra en la siguiente tabla:

| Id_lib | Título | Tipo | Autor_id |
|---|---|---|---|
| LIB_000016 | Crónica de una muerte anunciada | Novela | GAGA |
| LIB_000017 | ¿? | Teatro | GAGA |
| LIB_000018 | Doce cuentos peregrinos | Cuento | GAGA |
| LIB_000001 | El club de los suicidas | Azul | ROST |
| LIB_000001 | Poemas | Poesía | XXXX |

**Tabla 4 Tabla de Libro**

Las siguientes son preguntas que se pueden realizar al observar el contenido de la tabla Libro:

- ¿Puede haber dos libros con el mismo valor en id_lib?
- ¿Puede haber un libro sin valor en título?
- ¿Es posible el valor “XXXX” en el atributo autor_id?
- ¿Tiene sentido el valor “Azul” en el atributo tipo?

Para responder las preguntas anteriores, se deben estudiar y aplicar los aspectos que se enlistan a continuación:

- Definición de dominios.
- Restricción de unicidad.
- Restricción de valor no nulo.
- Definición de clave primaria.
- Definición de claves ajenas.
- Restricciones de integridad generales.

Se especifican junto con el esquema de la base de datos y el responsable de que se cumplan es el sistema gestor de bases de datos.

### 3.1.1 Restricciones sobre atributos

#### Dominio.

El asociar un dominio a cada atributo restringe el conjunto de valores que puede tomar ese atributo.
Ejemplo:

“El tipo de publicación únicamente puede ser Novela, Cuento, Teatro o Poesía”
Dominios:Dom_tipo : {Novela, Cuento, Teatro, Poesía, ...}

Para Publicaciones:

Publicación (id_lib: dom_id_lib, título: dom_título, tipo: dom_tipo, autor_id:
dom_autor_id);

### Restricción de Valor No Nulo

VNN: {Ao,..., Ap}

La definición de una restricción de valor no nulo sobre un conjunto de atributos K de la relación R expresa la siguiente propiedad: “No debe haber en R una tupla que tenga el valor nulo en algún atributo de K”.

Ejemplo:

VNN: { título }

“No debe haber en Publicación una tupla que tenga el valor nulo en algún atributo de título”.

Formalmente esta restricción se define como:

∀ t (Publicación(t)-nulo(t.título))

## 3.2 Restricción de unicidad

La definición de una restricción de unicidad sobre un conjunto de atributos K de la relación R expresa la siguiente propiedad: “no debe haber en R dos tuplas que tengan el mismo valor en todos los atributos del conjunto K”.

Uni: {Ao,..., Ap}

Ejemplo:

Uni: {id_lib}

“No debe haber en Publicación dos tuplas que tengan el mismo valor en el atributo id_lib”.

Formalmente esta restricción se define como:

¬t1 t2 (Publicación(t1) Publicación(t2) t1≠t2 t1.id_lib = t2.id_lib
¬ nulo(t1.id_lib) ¬nulo( t2.id_lib))

## 3.3 Concepto de clave primaria

Una clave primaria de una relación es un conjunto de atributos de su esquema que son elegidos para servir de identificador unívoco de sus tuplas:

CP: {Ao,..., Ap}

- Deberá ser minimal.

- Sus atributos deberán tener siempre un valor para cada tupla (restricción de valor no nulo).
- Este valor deberá ser único para cada tupla (restricción de unicidad).

Ejemplo:

CP: {id_lib}
“id_lib es un atributo clave primaria de Publicaciones”

Formalmente esto se puede describir como:

¬t1 t2 (Publicación(t1) Publicación(t2) t1≠t2 t1.id_lib = t2.id_lib)
)∀ t (Publicación(t) )-nulo(t.id_lib))

Dado un conjunto de atributos CP que se ha definido como clave primaria de R, se dice que R satisface la restricción de integridad de clave primaria si se cumplen las siguientes propiedades:

1. R satisface una restricción de valor no nulo sobre CP
2. R satisface una restricción de unicidad sobre CP

en caso contrario R viola esta restricción. Además, CP debe ser minimal; es decir, no debe tener ningún subconjunto propio que pueda ser a su vez clave primaria de R.

## 3.4 Concepto de clave ajena

El uso de claves ajenas es el mecanismo que proporciona el modelo relacional para expresar asociaciones entre los objetos representados en el esquema de la base de datos. Este mecanismo se define para que dichas asociaciones, si se realizan, se hagan siempre adecuadamente.

CAj: {Ao,..., Ap} ⊔ S

Con este objetivo, se añade al esquema de una relación, R, un conjunto de atributos que hagan referencia a un conjunto de atributos de una relación S. A ese conjunto de atributos se les denomina clave ajena de la relación R que hace referencia a la relación S.

| id_lib | título | tipo | autor_id |
|---|---|---|---|
| LIB-000016 | Crónica de una muerte anunciada | Novela | GAGA |
| LIB-000017 | Siempre NO | Teatro | GAGA |
| LIB-000008 | Doce cuentos peregrinos | Cuento | GAGA |
| LIB-000001 | El club de los suicidas | Novela | ROST |
| LIB-000004 | Poemas | Poesía | BERU |

| autor_id | nombre |
|---|---|
| GAGA | Gálamo Gante |
| ROST | Robert Steinball |
| BERU | Bertrand Rusbelt |

Dada una clave ajena CA de R que hace referencia a S, ésta se define como:

- Un subconjunto K = {Ai, Aj,..., Ak} del esquema de R
- Una biyección f: K ⊔ J tal que:
  - J es un conjunto de atributos de S
  - J tiene restricción de unicidad
  - V Ai (Ai K)  Ai y f(Ai) tienen el mismo dominio.
- Un tipo de integridad referencial.

Esta integridad referencial puede ser: débil, parcial o completa.
R satisface la restricción de integridad referencial sobre CA si, según el tipo elegido, se cumple la propiedad que se especifica:

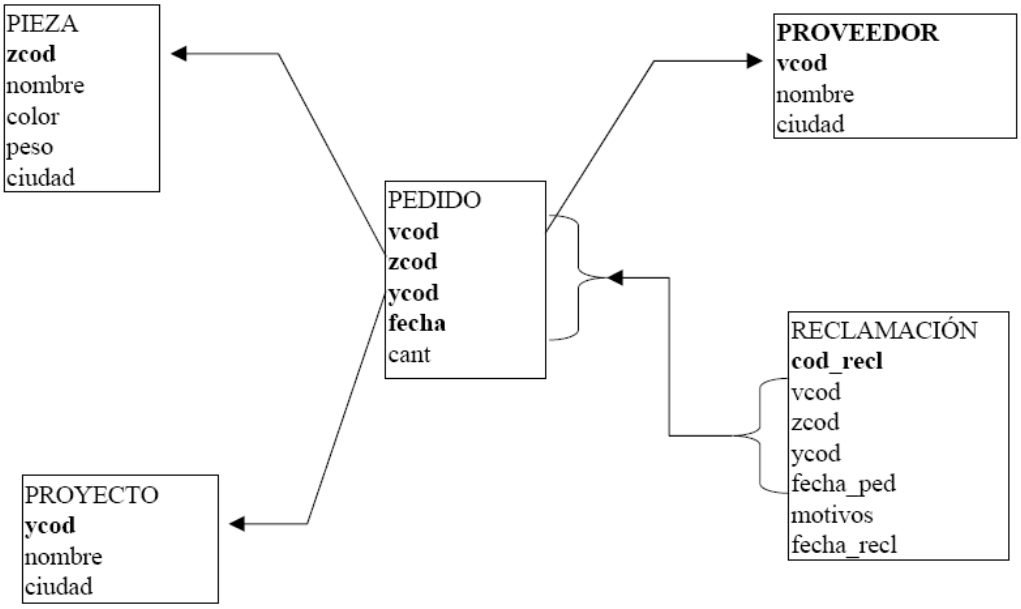
- I. R. Débil: si en una tupla de R todos los valores de los atributos de K tienen un valor que no es el nulo, entonces debe existir una tupla en S que tome esos mismos valores en los atributos de J.
- I. R. Parcial: si en una tupla de R algún atributo de K no toma el valor nulo, entonces debe existir una tupla en S que tome en los atributos de J los mismos valores que los atributos de K con valor no nulo.
- I. R. Completa: en una tupla de R todos los atributos de K deben tener el valor nulo o bien todos tienen un valor que no es el nulo y entonces debe existir una tupla en S que tome en los atributos de J los mismos valores que toman los de K.

## Clave Ajena: Notación Simplificada

- La biyección f: K ⊏ J se puede omitir cuando J es la clave primaria de S y se da uno de los siguientes casos:
  - El conjunto K contiene un único atributo, o
  - La biyección está definida por la igualdad sintáctica entre los nombres de atributo de K y de J.
- El tipo de integridad referencial se puede omitir en cualquiera de estos casos:
  - La clave ajena K conste sólo de un atributo, o
  - Cuando todos ellos sufran restricción de valor no nulo, ya que en estos casos los tres tipos de integridad referencial coinciden.

Ejemplo

PROVEEDOR (vcod: d_vcod, nombre: d_nom1, ciudad: d_ciu) CP:{vcod}
PIEZA (zcod: d_zcod, nombre: d_nom2, color: d_color, peso: d_peso, ciudad: d_ciu)
CP: {zcod}
PROYECTO(ycod: d_ycod, nombre: d_nom3, ciudad: d_ciu) CP: {ycod}
PEDIDO (vcod: d_vcod, zcod: d_zcod, ycod: d_ycod, fecha: d_fecha, cant: d_cant)
CP: {vcod, zcod, ycod, fecha}
CAj: {vcod} ⊏ PROVEEDOR
CAj: {zcod} ⊏ PIEZA
CAj: {ycod} ⊏ PROYECTO
RECLAMACIÓN (cod_recl: d_rcod, vcod: d_vcod, zcod: d_zcod, ycod: d_ycod, fecha_ped: d_fecha, motivo: d_mot, fecha_recl: d_fecha)
CP: {cod_recl}
CAj:{vcod, zcod, ycod, fecha_ped} ⊏ PEDIDO f(fecha_ped) = fecha




**PEDIDO**

| vcod | zcod | ycod | fecha | cant |
|---|---|---|---|---|
| Ford | Volante | Focus | 1/1/99 | 100 |
| Ford | Rueda | Ka | 2/1/99 | 300 |
| Ford | Rueda | Ka | ? | 50 |
| Ford | Volante | Focus | 1/1/99 | 500 |

**RECLAMACIÓN**

| cod_recl | vcod | zcod | ycod | fecha_ped | motivos | fecha_recl |
|---|---|---|---|---|---|---|
| 1 | Ford | Volante | Focus | 1/1/99 | Volante Cuadrado | 5/2/99 |
| 2 | Ford | Rueda | ? | 3/1/99 | Ruedas pinchadas | 7/2/99 |
| 3 | Ford | Rueda | ? | 2/1/99 | Ovoidales | 7/3/99 |
| 4 | ? | ? | ? | ? | No se pidió | 13/3/99 |
| 5 | Ford | Puerta | Ka | 1/1/99 | Entra frío | 14/3/99 |

# Unidad IV. Consultas Avanzadas


Las consultas son una de las partes fundamentales de las bases de datos ya que permiten obtener información para las empresas u organizaciones, que les sirve para tomar decisiones.

## 4.1 Combinaciones.

Las consultas mutitabla son llamadas así porque están basadas en más de una tabla. Otra manera de llamarlas es como Combinaciones.
Las combinaciones permiten recuperar datos de dos o más tablas según las relaciones lógicas entre ellas. Las combinaciones indican cómo debe usar Microsoft SQL Server los datos de una tabla para seleccionar las filas de otra tabla
Una condición de combinación define la forma en la que dos tablas se relacionan en una consulta al:

- Especificar la columna de cada tabla que debe usarse para la combinación. Una condición de combinación típica especifica una clave externa de una tabla y su clave asociada en otra tabla.
- Especificar un operador lógico (por ejemplo, = o <>) para usarlo en los valores de comparación de las columnas.

Las combinaciones internas se pueden especificar en las cláusulas FROM o WHERE. Las combinaciones externas sólo se pueden especificar en la cláusula FROM. Las condiciones de combinación se combinan con las condiciones de búsqueda de WHERE y HAVING para controlar cuáles son las filas seleccionadas de las tablas base a las que se hace referencia en la cláusula FROM.

### 4.1.1 Join

Las condiciones de la combinación en la cláusula FROM ayuda a separarlas de cualquier otra condición de búsqueda que se pueda especificar en una cláusula WHERE; es el método recomendado para especificar combinaciones. La sintaxis simplificada de la combinación de la cláusula FROM es:

```
FROM first_table join_type second_table [ON (join_condition)]
```

*join_type* especifica el tipo de combinación que se lleva a cabo: interior, exterior o cruzada. *join_condition* define el predicado que se va a evaluar en cada par de filas combinadas. Por ejemplo:

```
FROM Purchasing.ProductVendor JOIN Purchasing.Vendor
        ON (ProductVendor.VendorID = Vendor.VendorID)
```

El ejemplo completo quedaría de la siguiente manera:

```
SELECT ProductID, Purchasing.Vendor.VendorID, Name
FROM Purchasing.ProductVendor JOIN Purchasing.Vendor
```

```
        ON (Purchasing.ProductVendor.VendorID = Purchasing.Vendor.VendorID)
WHERE StandardPrice > $10
AND Name LIKE 'F%'
```

Cuyo resultado es la tabla que devuelve la información de los productos y proveedores de cualquier combinación de partes suministrada por una empresa cuyo nombre empieza por la letra F y el precio del producto es superior a 10 $
La siguiente consulta contiene la misma condición de combinación pero ahora especificada en la cláusula WHERE:

```
SELECT pv.ProductID, v.VendorID, v.Name
FROM Purchasing.ProductVendor pv, Purchasing.Vendor v
WHERE pv.VendorID = v.VendorID
        AND StandardPrice > $10
        AND Name LIKE N'F%'
```

Como ya habíamos mencionado anteriormente las condiciones de combinación se pueden especificar en las cláusulas FROM o WHERE, aunque se recomienda que se especifiquen en la cláusula FROM. Las cláusulas WHERE y HAVING pueden contener también condiciones de búsqueda para filtrar aún más las filas seleccionadas por las condiciones de combinación.
Las combinaciones se pueden clasificar en:

- Combinaciones internas (la operación de combinación típica, que usa algunos operadores de comparación como = o <>). En este tipo se incluyen las combinaciones equivalentes y las combinaciones naturales. Las combinaciones internas usan un operador de comparación para hacer coincidir las filas de dos tablas según los valores de las columnas comunes de cada tabla. Un ejemplo sería recuperar todas las filas en las que el número de identificación de estudiante es el mismo en las tablas students y courses.
- Combinaciones externas. Puede ser una combinación externa izquierda, derecha o completa. Las combinaciones externas se especifican en la cláusula FROM con uno de los siguientes conjuntos de palabras clave:
    - LEFT JOIN o LEFT OUTER JOIN. El conjunto de resultados de una combinación externa izquierda incluye todas las filas de la tabla de la izquierda especificada en la cláusula LEFT OUTER y no sólo aquellas en las que coincidan las columnas combinadas. Cuando

una fila de la tabla de la izquierda no tiene filas coincidentes en la tabla de la derecha, la fila asociada del conjunto de resultados contiene valores NULL en todas las columnas de la lista de selección que procedan de la tabla de la derecha.

  - RIGHT JOIN o RIGHT OUTER JOIN.Una combinación externa derecha es lo contrario de una combinación externa izquierda. Se devuelven todas las filas de la tabla de la derecha. Cada vez que una fila de la tabla de la derecha no tenga correspondencia en la tabla de la izquierda, se devuelven valores NULL para la tabla de la izquierda.
  - FULL JOIN o FULL OUTER JOIN. Una combinación externa completa devuelve todas las filas de las tablas de la izquierda y la derecha. Cada vez que una fila no tenga coincidencia en la otra tabla, las columnas de la lista de selección de la otra tabla contendrán valores NULL. Cuando haya una coincidencia entre las tablas, la fila completa del conjunto de resultados contendrá los valores de datos de las tablas base.

- Combinaciones cruzadas. Las combinaciones cruzadas devuelven todas las filas de la tabla izquierda. Cada fila de la tabla izquierda se combina con todas las filas de la tabla de la derecha. Las combinaciones cruzadas se llaman también productos cartesianos.

### 4.1.2 Combinaciones internas

Una combinación interna es aquélla en la que los valores de las columnas que se están combinando se comparan mediante un operador de comparación.
Ejemplo 1: Combinación interna.

```
SELECT *
FROM HumanResources.Employee AS e
        INNER JOIN Person.Contact AS c ON e.ContactID = c.ContactID
ORDER BY c.LastName
```

Esta combinación interna se conoce como una combinación equivalente. Devuelve todas las columnas de ambas tablas y sólo devuelve las filas en las que haya un valor igual en la columna de la combinación.
Ejemplo 2: Se utiliza una combinación menor que (<) para buscar precios de

ventas del producto 718 que sean inferiores al precio de lista recomendado para ese producto

```
SELECT DISTINCT p.ProductID, p.Name, p.ListPrice, sd.UnitPrice AS 'Selling
      Price'
FROM Sales.SalesOrderDetail AS sd JOIN Production.Product AS p
       ON sd.ProductID = p.ProductID AND sd.UnitPrice < p.ListPrice
WHERE p.ProductID = 718;
```

Ejemplo 3: Combinar más de dos tablas. La siguiente consulta busca los nombres de todos los productos de una subcategoría determinada y los nombres de sus proveedores

```
SELECT p.Name, v.Name
FROM Production.Product p
       JOIN Purchasing.ProductVendor pv ON p.ProductID = pv.ProductID
       JOIN Purchasing.Vendor v ON pv.VendorID = v.VendorID
WHERE ProductSubcategoryID = 15
ORDER BY v.Name
```

Ejemplo 4: Se utiliza una combinación no igual que se combina con una autocombinación para buscar todas las filas de la tabla ProductVendor en la que dos o más filas tengan el mismo ProductID pero distintos números de VendorID (es decir, productos que tienen más de un proveedor):

```
SELECT DISTINCT p1.VendorID, p1.ProductID
FROM Purchasing.ProductVendor p1 INNER JOIN Purchasing.ProductVendor p2
       ON p1.ProductID = p2.ProductID
WHERE p1.VendorID <> p2.VendorID
ORDER BY p1.VendorID
```

Ejemplo 5: Una combinación cruzada que no tenga una cláusula WHERE genera el producto cartesiano de las tablas involucradas en la combinación

```
SELECT p.SalesPersonID, t.Name AS Territory
FROM Sales.SalesPerson p
CROSS JOIN Sales.SalesTerritory t
ORDER BY p.SalesPersonID;
```

Ejemplo 6: Al agregar una cláusula WHERE, la combinación cruzada se comporta como una combinación interna.

```
SELECT p.SalesPersonID, t.Name AS Territory
FROM Sales.SalesPerson p CROSS JOIN Sales.SalesTerritory t
WHERE p.TerritoryID = t.TerritoryID
ORDER BY p.SalesPersonID;
```

## 4.2 Subconsultas

Una subconsulta es una consulta anidada en una instrucción SELECT, INSERT,

UPDATE o DELETE, o bien en otra subconsulta. Las subconsultas se pueden utilizar en cualquier parte en la que se permita una expresión. En este ejemplo, se utiliza una subconsulta como una expresión de columna llamada MaxUnitPrice en una instrucción SELECT.

### 4.2.1 Reglas de las subconsultas

Las subconsultas están sujetas a las restricciones siguientes:

- La lista de selección de una subconsulta que se especifica con un operador de comparación, sólo puede incluir un nombre de expresión o columna (excepto EXISTS e IN, que operan en SELECT * o en una lista respectivamente).
- Si la cláusula WHERE de una consulta externa incluye un nombre de columna, debe ser compatible con una combinación con la columna indicada en la lista de selección de la subconsulta.
- Los tipos de datos ntext, text y image no están permitidos en las listas de selección de subconsultas.
- Puesto que deben devolver un solo valor, las subconsultas que se especifican con un operador de comparación sin modificar (no seguido de la palabra clave ANY o ALL) no pueden incluir las cláusulas GROUP BY y HAVING.
- La palabra clave DISTINCT no se puede usar con subconsultas que incluyan GROUP BY.
- Sólo se puede especificar ORDER BY si se especifica también TOP.
- Una vista creada con una subconsulta no se puede actualizar.
- La lista de selección de una subconsulta especificada con EXISTS, por convención, tiene un asterisco (*) en lugar de un solo nombre de columna. Las reglas de una subconsulta especificada con EXISTS son idénticas a las de una lista de selección estándar, porque este tipo de subconsulta crea una prueba de existencia y devuelve TRUE o FALSE en lugar de datos.

### 4.2.2 Tipos de subconsultas

Las subconsultas se pueden especificar en muchos sitios:

- Con alias

- Con IN o NOT IN.
- En las instrucciones UPDATE, DELETE e INSERT.
- Con los operadores de comparación.
- Con ANY, SOME o ALL.
- Con EXISTS o NOT EXISTS.
- En lugar de una expresión.

### 4.2.3 Ejemplos

Ejemplo 1: En este ejemplo, se utiliza una subconsulta como una expresión de columna llamada MaxUnitPrice en una instrucción SELECT

```
SELECT Ord.SalesOrderID, Ord.OrderDate,
        (SELECT MAX(OrdDet.UnitPrice)
                FROM AdventureWorks.Sales.SalesOrderDetail AS OrdDet
                WHERE Ord.SalesOrderID = OrdDet.SalesOrderID) AS MaxUnitPrice
FROM AdventureWorks.Sales.SalesOrderHeader AS Ord
```

Ejemplo2: A continuación aparece un ejemplo que muestra una subconsulta SELECT:

```
SELECT Name
FROM AdventureWorks.Production.Product
WHERE ListPrice = (SELECT ListPrice
                                FROM AdventureWorks.Production.Product
                                WHERE Name = 'Chainring Bolts')
```

Ejemplo 3: La columna CustomerID de la cláusula WHERE de la consulta externa está calificada implícitamente por el nombre de tabla de la cláusula FROM de la consulta externa, Sales.Store. La referencia a CustomerID en la lista de selección de la subconsulta está calificada por la cláusula FROM de la subconsulta, es decir, por la tabla Sales.Customer:

```
SELECT Name
FROM Sales.Store
WHERE CustomerID NOT IN (SELECT CustomerID
                                        FROM Sales.Customer
                                        WHERE TerritoryID = 5)
```

Ejemplo 4: En una instrucción se puede anidar cualquier número de subconsultas. En éste ejemplo se buscan los nombres de los empleados que también son vendedores:

```
SELECT LastName, FirstName
FROM Person.Contact
WHERE ContactID IN (SELECT ContactID
                                FROM HumanResources.Employee
                                WHERE EmployeeID IN (SELECT SalesPersonID
                                                                FROM Sales.SalesPerson)
```

## 4.3 Operaciones con conjuntos

### 4.3.1 Unión

Esta operación se utiliza cuando tenemos dos tablas con las mismas columnas y queremos obtener una nueva tabla con las filas de la primera y las filas de la segunda. En este caso la tabla resultante tiene las mismas columnas que la primera tabla (que son las mismas que las de la segunda tabla).
Cuando hablamos de tablas pueden ser tablas reales almacenadas en la base de datos o tablas lógicas (resultados de una consulta), esto nos permite utilizar la operación con más frecuencia ya que pocas veces tenemos en una base de datos tablas idénticas en cuanto a columnas. El resultado es siempre una tabla lógica.
A continuación se muestran las reglas básicas para combinar los conjuntos de resultados de dos consultas con UNION:

- El número y el orden de las columnas deben ser idénticos en todas las consultas.
- Los tipos de datos deben ser compatibles.

#### 4.3.1.1Ejemplos de Unión

Ejemplo 1: Por ejemplo queremos en un sólo listado los productos cuyas existencias sean iguales a cero y también los productos que aparecen en pedidos del año 90. En este caso tenemos unos productos en la tabla de productos y los otros en la tabla de pedidos, las tablas no tienen las mismas columnas no se puede hacer una unión de ellas pero lo que interesa realmente es el identificador del producto (idfab, idproducto), luego por una parte sacamos los códigos de los productos con existencias cero (con una consulta), por otra parte los códigos de los productos que aparecen en pedidos del año 90 (con otra consulta), y luego unimos estas dos tablas lógicas.

```
SELECT idfab,idproducto
FROM productos
WHERE existencias = 0
UNION ALL
SELECT fab,producto
FROM pedidos
WHERE year(fechapedido) = 1990
ORDER BY idproducto
```

Ejemplo 2: El conjunto de resultados incluye el contenido de las columnas

ProductModelID y Name de las tablas ProductModel y Gloves.

```
SELECT ProductModelID, Name
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves
ORDER BY Name ;
```

Ejemplo 3: La cláusula INTO de la segunda instrucción SELECT especifica que la tabla denominada ProductResults contiene el conjunto final de resultados de la unión de las columnas designadas de las tablas ProductModel y Gloves.

```
SELECT ProductModelID, Name
INTO ProductResults
FROM Production.ProductModel
WHERE ProductModelID NOT IN (3, 4)
UNION
SELECT ProductModelID, Name
FROM dbo.Gloves ;
```

Ejemplo 4: El orden de algunos parámetros utilizados con la cláusula UNION es importante. En éste ejemplo se muestra el uso correcto e incorrecto de UNION en dos instrucciones SELECT en las que se va a cambiar el nombre de una columna en el resultado.

| /* INCORRECTO */<br>USE AdventureWorks ;<br>GO<br>SELECT ProductModelID, Name<br>FROM Production.ProductModel<br>WHERE ProductModelID NOT IN (3, 4)<br>ORDER BY Name<br>UNION<br>SELECT ProductModelID, Name<br>FROM dbo.Gloves ;<br>GO | /* CORRECTO */<br>USE AdventureWorks ;<br>GO<br>SELECT ProductModelID, Name<br>FROM Production.ProductModel<br>WHERE ProductModelID NOT IN (3, 4)<br>UNION<br>SELECT ProductModelID, Name<br>FROM dbo.Gloves<br>ORDER BY Name ;<br>GO |
|---|---|

### 4.3.2 Intersección y complemento

Las Instrucciones EXCEPT e INTERSECT Devuelven valores distintos al comparar los resultados de dos consultas.EXCEPT devuelve los valores distintos de la consulta izquierda que no se encuentran en la consulta derecha.
INTERSECT devuelve los valores distintos devueltos por las consultas situadas a

los lados izquierdo y derecho del operando INTERSECT.
Las reglas básicas para combinar los conjuntos de resultados de dos consultas que utilizan EXCEPT o INTERSECT son las siguientes:

- El número y el orden de las columnas debe ser el mismo en todas las consultas.
- Los tipos de datos deben ser compatibles.

Cuando los tipos de datos de columnas comparables devueltos por las consultas situadas a la izquierda y a la derecha de los operandos EXCEPT o INTERSECT son tipos de datos de caracteres con intercalaciones diferentes, la comparación requerida se realiza conforme a las reglas de prioridad de intercalación.
Cuando se comparan filas para determinar valores distintos, dos valores NULL se consideran equivalentes.
Los nombres de columna del conjunto de resultados devueltos por EXCEPT o INTERSECT son los mismos que han sido devueltos por la consulta situada en el lado izquierdo del operando.
Los nombres o alias de columna de las cláusulas ORDER BY deben hacer referencia a los nombres de columna devueltos por la consulta del lado izquierdo.
La capacidad de aceptar valores NULL de cualquier columna del conjunto de resultados devueltos por EXCEPT o INTERSECT es la misma que la de la columna correspondiente devuelta por la consulta situada en el lado izquierdo del operando.
Si EXCEPT o INTERSECT se utilizan con otros operadores en una expresión, ésta se evalúa en el contexto de la siguiente prioridad:

1. Expresiones entre paréntesis.
2. El operando INTERSECT
3. EXCEPT y UNION se evalúan de izquierda a derecha según su posición en la expresión

### 4.3.2.1Ejemplos.

Ejemplo 1: La siguiente consulta devuelve los valores distintos devueltos por las consultas situadas a los lados izquierdo y derecho del operando INTERSECT

```
SELECT ProductID
FROM Production.Product
INTERSECT
```

```
SELECT ProductID
FROM Production.WorkOrder ;
```

Ejemplo 2: Devuelve los valores distintos de la consulta situados a la izquierda del operando EXCEPT que no se encuentran en la consulta derecha.

```
SELECT ProductID
FROM Production.Product
EXCEPT
SELECT ProductID
FROM Production.WorkOrder ;
```

Ejemplo 3: Devuelve los valores distintos de la consulta situados a la izquierda del operando EXCEPT que no se encuentran en la consulta derecha. Las tablas se invierten respecto al ejemplo anterior.

```
SELECT ProductID
FROM Production.WorkOrder
EXCEPT
SELECT ProductID
FROM Production.Product ;
```

# Unidad V. Procedimientos Almacenados y Disparadores


Los procedimientos almacenados son un conjunto de instrucciones que se compilan una sola vez y se almacenan en el sistema gestor de bases de datos para que se puedan ejecutar posteriormente.

## 5.1 Procedimientos almacenados

Un procedimiento almacenado es un grupo de instrucciones Transact-SQL que compila una vez, se almacena en el servidor y se ejecuta varias veces. Los procedimientos almacenados son un método para encapsular tareas repetitivas. Admiten variables declaradas por el usuario, ejecución condicional y otras características de programación muy eficaces.

### 5.1.1 Sintaxis General

```
CREATE PROC [ EDURE ] nombreProcedimiento [ ; número ]
[ { @tipoDatos procedimiento }
[ VARYING ] [ = predeterminado ] [ OUTPUT ] ] [ ,...n ]
[ WITH { RECOMPILE | ENCRYPTION | RECOMPILE , ENCRYPTION } ] [ FOR
      REPLICATION ]
AS instrucciónSql [ ...n ]
```

### 5.1.2 Reglas para programar procedimientos almacenados

De entre las reglas para programar procedimientos almacenados, cabe citar las siguientes:

- Puede crear otros objetos de base de datos dentro de un procedimiento almacenado. Puede hacer referencia a un objeto creado en el mismo procedimiento almacenado, siempre que se haya creado antes de hacer la referencia.
- Puede hacer referencia a tablas temporales dentro de un procedimiento almacenado.
- Si crea una tabla temporal local dentro de un procedimiento almacenado, ésta existirá únicamente para los fines del procedimiento y desaparecerá en cuanto éste finalice.
- Si ejecuta un procedimiento almacenado que llama otro procedimiento almacenado, este último puede tener acceso a todos los objetos creados por el primero, incluidas las tablas temporales.
- El número máximo de parámetros en un procedimiento almacenado es de 2100.
- El número máximo de variables locales en un procedimiento almacenado

está limitado únicamente por la memoria disponible.

- En función de la memoria disponible, el tamaño máximo de un procedimiento almacenado es de 128 megabytes (MB).

### 5.1.3 Declaración de variables.

Una variable local de Transact-SQL es un objeto que contiene un valor individual de datos de un tipo específico. Normalmente, las variables se utilizan en lotes y scripts:

- Como contadores, para contar el número de veces que se realiza un bucle o controlar cuántas veces debe ejecutarse.
- Para contener un valor de datos que desea probar mediante una instrucción de control de flujo.
- Para guardar el valor de un dato que se va a devolver en un código de retorno de un procedimiento almacenado o un valor devuelto de una función.
- Los nombres de algunas funciones del sistema de Transact-SQL empiezan con dos arrobas (@@). A pesar de que en versiones anteriores de Microsoft SQL Server se hacía referencia a las funciones que empiezan por @@ como variables globales, no son variables y no tienen el mismo comportamiento que las variables. Las funciones que empiezan por @@ son funciones del sistema, y el uso de su sintaxis sigue las reglas de las funciones.
- La instrucción DECLARE inicializa una variable de Transact-SQL al:
  - Asignar un nombre. El nombre debe tener un único @ como primer carácter.
  - Asignar un tipo de datos suministrado por el sistema o definido por el usuario y una longitud. Para las variables numéricas, se asignan también una precisión y una escala. Para las variables del tipo XML, puede asignarse una colección de esquemas opcional.
  - Establecer el valor a NULL.

Por ejemplo, la siguiente instrucción DECLARE crea una variable local llamada @mycounter con un tipo de datos int.

```
DECLARE @MyCounter int;
```

Para declarar más de una variable local, use una coma después de la primera variable local definida y, a continuación, especifique el nombre y tipo de datos de la siguiente variable local.
Por ejemplo, la siguiente instrucción DECLARE crea tres variables locales llamadas @LastName, @FirstName y @StateProvince, e inicializa cada una de ellas a NULL:

```
DECLARE @LastName nvarchar(30), @FirstName nvarchar(20), @StateProvince
nchar(2);
```

### 5.1.4 Ejecución de procedimientos Almacenados.

Para ejecutar un procedimiento almacenado, utilice la instrucción EXECUTE de Transact-SQL. También puede ejecutar un procedimiento almacenado sin necesidad de utilizar la palabra clave EXECUTE si el procedimiento almacenado es la primera instrucción del lote.
Es posible suministrar los valores de los parámetros si se escribe un procedimiento almacenado que los acepte. El valor suministrado debe ser una constante o una variable; no puede especificar un nombre de función como valor de parámetro. Las variables pueden ser definidas por el usuario o ser variables del sistema.
Si especifica los parámetros con el formato @parámetro =*value*, puede proporcionarlos en cualquier orden. También puede omitir los parámetros para los que se hayan especificado valores predeterminados. Si sólo especifica un parámetro con el formato @parámetro =*value*, deberá proporcionar todos los parámetros subsiguientes del mismo modo. Si no especifica los parámetros con el formato @parámetro =*value*, deberá especificarlos en el orden que se ha seguido en la instrucción CREATE PROCEDURE.
Cuando ejecute un procedimiento almacenado, el servidor rechazará todos los parámetros que no se incluyeron en la lista de parámetros durante la creación del procedimiento. No se aceptará ningún parámetro pasado por referencia (el nombre del parámetro se pasa explícitamente) si el nombre del parámetro no coincide
Aunque puede omitir los parámetros para los que se hayan especificado valores predeterminados, sólo puede truncar la lista de parámetros. Por ejemplo, si en un procedimiento almacenado hay cinco parámetros, puede omitir el cuarto y el

quinto, pero no puede omitir el cuarto e incluir el quinto a menos que suministre los parámetros con el formato @parámetro =*value*.
El valor predeterminado de un parámetro, si se ha definido para el parámetro del procedimiento almacenado, se utiliza cuando:

- No existe ningún valor especificado para el parámetro en el momento de ejecutar el procedimiento almacenado.
- Se especifica la palabra clave DEFAULT como valor para el parámetro.

5.1.5 Ejemplos.
Ejemplo 1: Crear un procedimiento almacenado que utilice parámetros Este ejemplo crea un procedimiento almacenado de gran utilidad para la base de datos pubs. A partir del nombre y primer apellido de un autor, el procedimiento almacenado muestra el título y el editor de cada uno de sus libros.

```
CREATE PROC au_info @lastname varchar(40), @firstname varchar(20)
AS
SELECT au_lname, au_fname, title, pub_name
FROM authors INNER JOIN titleauthor ON authors.au_id = titleauthor.au_id JOIN titles ON titleauthor.title_id = titles.title_id JOIN publishers ON titles.pub_id = publishers.pub_id
WHERE au_fname = @firstnameAND au_lname = @lastname
GO
```

Ahora ejecute el procedimiento almacenado au_info:

```
EXECUTE au_info Ringer, Anne
GO
```

Ejemplo 2: Crear un procedimiento almacenado que utilice valores predeterminados para los parámetros.
Este ejemplo crea un procedimiento almacenado, pub_info2, que muestra los nombres de todos los autores que han escrito algún libro publicado por el editor que se especifica como parámetro. Si no se facilita el editor, el procedimiento almacenado muestra los autores cuyos libros han sido editados por Algodata Infosystems.

```
CREATE PROC pub_info2@pubname varchar(40) = 'Algodata Infosystems'
AS
SELECT au_lname, au_fname, pub_nameFROM authors a INNER JOIN titleauthor ta ON a.au_id = ta.au_id JOIN titles t ON ta.title_id = t.title_id JOIN publishers p ON t.pub_id = p.pub_id
WHERE @pubname = p.pub_name
```

```
GO
```

Ejecute pub_info2 sin especificar ningún parámetro:

```
EXECUTE pub_info2
GO
```

Ejemplo 3: Ejecutar un procedimiento almacenado que suplante el valor predeterminado de un parámetro con un valor explícito.

En este ejemplo, el procedimiento almacenado, showind2, el valor predeterminado para el parámetro @table es titles.

```
CREATE PROC showind2 @table varchar(30) = 'titles'
AS
SELECT TABLE_NAME = sysobjects.name,INDEX_NAME = sysindexes.name,
INDEX_ID = indid
FROM sysindexes INNER JOIN sysobjects ON sysobjects.id = sysindexes.id
WHERE sysobjects.name = @table
GO
```

Los títulos de las columnas (por ejemplo, TABLE_NAME) facilitan la lectura de los resultados. Esto es lo que muestra el procedimiento almacenado para la tabla authors:

```
EXECUTE showind2 authors
GO
```

Si no especifica ningún valor, SQL Server utiliza la tabla predeterminada, titles:

```
EXECUTE showind2
GO
```

Ejemplo4: Crear un procedimiento almacenado que utilice un valor predeterminado NULL para los parámetros.

El valor predeterminado para los parámetros puede ser NULL. En este caso, si el usuario no especifica ningún parámetro, SQL Server ejecuta el procedimiento almacenado de acuerdo con el resto de sus instrucciones. No se muestra ningún mensaje de error.

La definición del procedimiento también puede especificar que se realice alguna otra acción si no proporciona ningún parámetro. Por ejemplo:

```
CREATE PROC showind3 @table varchar(30) = NULL
AS
IF @table IS NULL
PRINT 'Give a table name'
ELSE
SELECT TABLE_NAME = sysobjects.name, INDEX_NAME = sysindexes.name,
INDEX_ID = indid
FROM sysindexes INNER JOIN sysobjectsON sysobjects.id = sysindexes.id
```

```
WHERE sysobjects.name = @table
GO
```

Ejemplo5: Crear un procedimiento almacenado que utilice un valor predeterminado para parámetros que incluya caracteres comodín.

El valor predeterminado puede incluir caracteres comodín (%, _, [] y [^]), si el procedimiento almacenado utiliza el parámetro con la palabra clave LIKE. Por ejemplo, es posible modificar showind para que se muestre información acerca de las tablas del sistema, si no especifica ningún parámetro:

```
CREATE PROC showind4 @table varchar(30) = 'sys%'
AS
SELECT TABLE_NAME = sysobjects.name, INDEX_NAME = sysindexes.name,
INDEX_ID = indid
FROM sysindexes INNER JOIN sysobjectsON sysobjects.id = sysindexes.id
WHERE sysobjects.name LIKE @table
GO
```

La siguiente variación del procedimiento almacenado au_info utiliza valores predeterminados con caracteres comodín para ambos parámetros:

```
CREATE PROC au_info2 @lastname varchar(30) = 'D%', @firstname varchar(18)
= '%'
AS
SELECT au_lname, au_fname, title, pub_name
FROM authors INNER JOIN titleauthor ON authors.au_id = titleauthor.au_id JOIN
titles ON titleauthor.title_id = titles.title_id JOIN publishers ON titles.pub_id =
publishers.pub_id
WHERE au_fname LIKE @firstname AND au_lname LIKE @lastname
GO
```

Si se ejecuta au_info2 sin parámetros, se muestran todos los autores cuyo apellido empiece por la letra D:

```
EXECUTE au_info2
GO
```

En este ejemplo se omite el segundo parámetro cuando se han definido valores predeterminados para dos parámetros, de modo que se pueden buscar los libros y los editores de todos los autores cuyo apellido sea Ringer:

```
EXECUTE au_info2 Ringer;
GO
```

## 5.2 Desencadenadores

Un desencadenador o disparador es un tipo especial de procedimiento

almacenado al que no llama directamente el usuario. Cuando se crea el desencadenador, se define para que se inicie automáticamente cuando se ejecuta un evento de lenguaje. SQL Server incluye tres tipos generales de desencadenadores: DML, DDL y de inicio de sesión.

Los desencadenadores DDL se invocan cuando tiene lugar un evento de lenguaje de definición de datos (DDL) en el servidor o la base de datos. Los desencadenadores de inicio de sesión activan procedimientos almacenados en respuesta a un evento LOGON. Este evento se genera cuando se establece una sesión de usuario con una instancia de SQL Server. Los desencadenadores DML se invocan cuando un evento de lenguaje de manipulación de datos (DML) tiene lugar en la base de datos. Los eventos DML incluyen instrucciones INSERT, UPDATE o DELETE que modifican datos en una tabla o vista especificada.

Un desencadenador DML puede consultar otras tablas e incluir instrucciones Transact-SQL complejas. El desencadenador y la instrucción que lo activa se tratan como una sola transacción, que puede revertirse desde el desencadenador. Si se detecta un error grave (por ejemplo, no hay suficiente espacio en disco), se revierte automáticamente toda la transacción.

### 5.2.1 Sintaxis general:

```
CREATE TRIGGER trigger_name
ON { table | view }
[ WITH ENCRYPTION ]
{
        { { FOR | AFTER | INSTEAD OF } { [ INSERT ] [ , ] [ UPDATE ] }
        [ WITH APPEND ]
        [ NOT FOR REPLICATION ]
        AS
        [ { IF UPDATE ( column )
        [ { AND | OR } UPDATE ( column ) ]
        [ ...n ]
        | IF ( COLUMNS_UPDATED ( ) { bitwise_operator } updated_bitmask )
        { comparison_operator } column_bitmask [ ...n ]
        }]
        sql_statement [ ...n ]
        }
}
```

### 5.2.2 Desencadenadores DML

Los desencadenadores DML tienen varias utilidades:

- Pueden realizar cambios en cascada mediante tablas relacionadas de la base de datos; sin embargo, estos cambios pueden ejecutarse de manera más eficaz mediante restricciones de integridad referencial en cascada.
- Pueden proteger contra operaciones INSERT, UPDATE y DELETE incorrectas o dañinas, y exigir otras restricciones que sean más complejas que las definidas con restricciones CHECK. A diferencia de éstas, los desencadenadores DML pueden hacer referencia a columnas de otras tablas. Por ejemplo, un desencadenador puede utilizar una instrucción SELECT de otra tabla para comparar con los datos insertados o actualizados y para realizar acciones adicionales, como modificar los datos o mostrar un mensaje de error definido por el usuario.
- Pueden evaluar el estado de una tabla antes y después de realizar una modificación de datos y actuar en función de esa diferencia.
- Varios desencadenadores DML del mismo tipo (INSERT, UPDATE o DELETE) en una tabla permiten realizar distintas acciones en respuesta a una misma instrucción de modificación.

### 5.2.2 Tipos de desencadenadores DML

Puede programar los siguientes tipos de desencadenadores DML:

- Desencadenadores AFTER. Los desencadenadores AFTER se ejecutan después de llevar a cabo la acción de las instrucciones INSERT, UPDATE o DELETE. La especificación de AFTER produce el mismo efecto que especificar FOR, que es la única opción disponible en las versiones anteriores de Microsoft SQL Server. El desencadenador AFTER sólo puede especificarse en tablas.
- Desencadenadores INSTEAD OF. Se ejecutan los desencadenadores INSTEAD OF en lugar de la acción habitual de desencadenamiento. También se pueden definir desencadenadores INSTEAD OF en vistas con una o más tablas base, donde se pueden ampliar los tipos de actualizaciones que puede admitir una vista.

- Desencadenadores CLR. Un desencadenador CLR puede ser un desencadenador AFTER o INSTEAD OF. Un desencadenador CLR también puede ser un desencadenador DDL. En lugar de ejecutar un procedimiento almacenado Transact-SQL, un desencadenador CLR ejecuta uno o más métodos escritos en código administrado que son miembros de un ensamblado creado en .NET Framework y cargado en SQL Server

### 5.2.3 Ejemplos

Ejemplo 1: Cuando se inserte un pedido, entrará en funcionamiento el trigger ActualizaVentasEmpleado y se ejecutará las instrucción UPDATE. La tabla empleados sumará a las ventas del empleado (ventas) mas el importe del pedido insertado (inserted.importe), y sólo actualizará el empleado cuyo numemp coincida con el campo rep del pedido insertado (WHERE numemp=inserted.rep)

```
CREATE TRIGGER ActualizaVentasEmpleadosON pedidos FOR INSERT
AS
UPDATE empleados SET ventas=ventas+inserted.importe FROM empleados,
inserted WHERE numemp=inserted.rep;
GO
```

# Unidad VI. Índices


Los índices de bases de datos son muy similares a los índices de los libros, facilitan las búsquedas en función de los ordenamientos que se realicen a través del índice.

## 6.1 Concepto de índices

Al igual que el índice de un libro, el índice de una base de datos permite encontrar rápidamente información específica en una tabla o vista indizada. Un índice contiene claves generadas a partir de una o varias columnas de la tabla o la vista y punteros que asignan la ubicación de almacenamiento de los datos especificados. Puede mejorar notablemente el rendimiento de las aplicaciones y consultas de bases de datos creando índices correctamente diseñados para que sean compatibles con las consultas. Los índices pueden reducir la cantidad de datos que se deben leer para devolver el conjunto de resultados de la consulta. Los índices también pueden exigir la unicidad en las filas de una tabla, lo que se garantiza la integridad de los datos de la tabla.

Un índice es una estructura de disco asociada con una tabla o una vista que acelera la recuperación de filas de la tabla o de la vista. Un índice contiene claves generadas a partir de una o varias columnas de la tabla o la vista. Dichas claves están almacenadas en una estructura (árbol b) que permite que SQL Server busque de forma rápida y eficiente la fila o filas asociadas a los valores de cada clave.

### 6.2 Utilización de índices

Los índices bien diseñados pueden reducir las operaciones de E/S de disco y consumen menos recursos del sistema, con lo que mejoran el rendimiento de la consulta. Los índices pueden ser útiles para diversas consultas que contienen instrucciones SELECT, UPDATE, DELETE o MERGE. Fíjese en la consulta SELECT Title, HireDate FROM HumanResources.Employee WHERE EmployeeID = 250 en la base de datos AdventureWorks. Cuando se ejecuta la consulta, el optimizador de consultas evalúa cada método disponible para recuperar datos y selecciona el método más eficiente. El método puede ser un recorrido de la tabla o puede ser recorrer uno o más índices si existen

Al realizar un recorrido de la tabla, el optimizador de consultas leerá todas las filas de la tabla y extraerá las filas que cumplen con los criterios de la consulta. Un recorrido de la tabla genera muchas operaciones de E/S de disco y puede consumir recursos. No obstante, puede ser el método más eficaz si, por ejemplo,

el conjunto de resultados de la consulta es un porcentaje elevado de filas de la tabla.
Cuando el optimizador de consultas utiliza un índice, busca en las columnas de clave de índice, busca la ubicación de almacenamiento de las filas que necesita la consulta y extrae las filas coincidentes de esa ubicación. Generalmente, la búsqueda del índice es mucho más rápida que la búsqueda de la tabla porque, a diferencia de la tabla, un índice frecuentemente contiene muy pocas columnas por fila y las filas están ordenadas.
El optimizador de consultas normalmente selecciona el método más eficaz cuando ejecuta consultas. No obstante, si no hay índices disponibles, el optimizador de consultas debe utilizar un recorrido de la tabla. Su tarea consiste en diseñar y crear los índices más apropiados para su entorno de forma que el optimizador de consultas disponga de una selección de índices eficaces entre los que elegir.

## 6.3 Tipos de índices

Los tipos de índice disponibles en SQL Server son los siguientes:

- Agrupado: Un índice agrupado ordena y almacena las filas de datos de la tabla o vista por orden en función de la clave del índice agrupado. El índice agrupado se implementa como una estructura de árbol b que admite la recuperación rápida de las filas a partir de los valores de las claves del índice agrupado.
- No agrupado: Los índices no agrupados se pueden definir en una tabla o vista con un índice agrupado o en un montón. Cada fila del índice no agrupado contiene un valor de clave no agrupada y un localizador de fila. Este localizador apunta a la fila de datos del índice agrupado o el montón que contiene el valor de clave. Las filas del índice se almacenan en el mismo orden que los valores de la clave del índice, pero no se garantiza que las filas de datos estén en un determinado orden a menos que se cree un índice agrupado en la tabla.
- Único: Un índice único garantiza que la clave de índice no contenga valores duplicados y, por tanto, cada fila de la tabla o vista es en cierta forma única. Tanto los índices agrupados como los no agrupados pueden ser únicos.

- Índice con columnas incluidas: Índice no agrupado que se extiende para incluir columnas sin clave además de las columnas de clave.
- Vistas indizadas: Un índice en una vista materializa (ejecuta) la vista, y el conjunto de resultados se almacena de forma permanente en un índice agrupado único, del mismo modo que se almacena una tabla con un índice agrupado. Los índices no agrupados de la vista se pueden agregar una vez creado el índice agrupado.
- Texto: Tipo especial de índice funcional basado en símbolos (token) que crea y mantiene el motor de texto completo de Microsoft para SQL Server. Proporciona la compatibilidad adecuada para búsquedas de texto complejas en datos de cadenas de caracteres.
- Espacial: Un índice espacial proporciona la capacidad de realizar de forma más eficaz determinadas operaciones en objetos espaciales (*datos espaciales*) en una columna del tipo de datos geometry. El índice espacial reduce el número de objetos a los que es necesario aplicar las operaciones espaciales, que son relativamente costosas.
- Filtrado: Índice no clúster optimizado, especialmente indicado para cubrir consultas que seleccionan de un subconjunto bien definido de datos. Utiliza un predicado de filtro para indizar una parte de las filas de la tabla. Un índice filtrado bien diseñado puede mejorar el rendimiento de las consultas, reducir los costos de mantenimiento y de almacenamiento del índice en relación con los índices de tabla completa.
- • XML: Representación dividida y permanente de los objetos XML binarios grandes (BLOB) de la columna de tipo de datos XML.

## 6.4 Creación de índices

Los índices se crean automáticamente cuando las restricciones PRIMARY KEY y UNIQUE se definen en las columnas de tabla. Por ejemplo, cuando cree una tabla e identifique una determinada columna como la clave primaria, Database Engine (Motor de base de datos) crea automáticamente una restricción PRIMARY KEY y un índice en esa columna.
Los siguientes pasos forman parte de la estrategia recomendada para crear índices:

1. Diseñar el índice. El diseño de índices es una tarea crítica. El diseño de índices incluye la determinación de las columnas que se utilizarán, la selección del tipo de índice (por ejemplo, agrupado o no agrupado), la selección de opciones de índice adecuadas y la determinación de grupos de archivos o de la ubicación de esquemas de partición.
2. Determinar el mejor método de creación. Los índices se crean de las siguientes maneras:
    a. Definiendo una restricción PRIMARY KEY o UNIQUE en una columna mediante CREATE TABLE o ALTER TABLE: SQL Server Database Engine (Motor de base de datos de SQL Server) crea automáticamente un índice único para hacer cumplir los requisitos de unicidad de una restricción PRIMARY KEY o UNIQUE. De forma predeterminada se crea un índice clúster único para hacer cumplir una restricción PRIMARY KEY, a menos que ya exista un índice clúster en la tabla o que usted especifique un índice no agrupado único. De forma predeterminada se crea un índice único no agrupado para hacer cumplir una restricción UNIQUE a menos que se especifique explícitamente un índice clúster único y no exista un índice clúster en la tabla. También se pueden especificar las opciones de índice, la ubicación del índice, el grupo de archivos o el esquema de la partición. Un índice creado como parte de una restricción PRIMARY KEY o UNIQUE recibe automáticamente el mismo nombre que la restricción.
    b. Creando un índice independiente de una restricción utilizando la instrucción CREATE INDEX, o el cuadro de diálogo Nuevo índice en el Explorador de objetos de SQL Server Management Studio: Debe especificar el nombre del índice, de la tabla y de las columnas a las que se aplica el índice. También se pueden especificar las opciones de índice, la ubicación del índice, el grupo de archivos o el esquema de la partición. De forma predeterminada, se crea un índice que no es único y no está agrupado si no se especifican las opciones únicas o agrupadas. Para crear un índice filtrado, use la cláusula opcional

WHERE.

3. Crear el índice. Un factor importante que debe tenerse en cuenta es si el índice se creará en una tabla vacía o en una tabla con datos. La creación de un índice en una tabla vacía no tiene implicaciones de rendimiento en el momento de creación del índice; sin embargo, el rendimiento se verá afectado cuando se agreguen los datos a la tabla. La creación de índices en tablas grandes debe planearse con cuidado para que el rendimiento de la base de datos no se vea afectado. La mejor manera de crear índices en tablas de gran tamaño es empezar con el índice clúster y, a continuación, generar los índices no agrupados. Considere la posibilidad de establecer la opción ONLINE en ON cuando cree índices en tablas existentes. Cuando se establece en ON, los bloqueos a largo plazo no se retienen, lo que permite que continúen consultas o actualizaciones a la tabla subyacente.

## 6.5 Sintaxis General

```
CREATE [ UNIQUE ] [ CLUSTERED | NONCLUSTERED ] INDEX index_name ON
<object> ( column [ ASC | DESC ] [ ,...n ] )
□ [ INCLUDE ( column_name [ ,...n ] ) ]
□ [ WHERE <filter_predicate> ]
□ [ WITH ( <relational_index_option> [ ,...n ] ) ]
□ [ ON { partition_scheme_name ( column_name ) | filegroup_name |
default} ]
□ [ FILESTREAM_ON { filestream_filegroup_name |
partition_scheme_name | " NULL" } ] [;]
```

- UNIQUE: Crea un índice único en una tabla o vista. Un índice único es aquel en el que no se permite que dos filas tengan el mismo valor de clave del índice. El índice clúster de una vista debe ser único. Database Engine (Motor de base de datos) no admite la creación de un índice único sobre columnas que ya contengan valores duplicados, independientemente de si se ha establecido o no IGNORE_DUP_KEY en ON. Si se intenta, Database Engine (Motor de base de datos) muestra un mensaje de error. Se deben quitar los valores duplicados para poder crear un índice único en la columna o columnas. Las columnas que se utilizan en un índice único se deben

establecer en NOT NULL, dado que varios valores NULL se consideran duplicados cuando se crea un índice único.

- CLUSTERED: Crea un índice en el que el orden lógico de los valores de clave determina el orden físico de las filas correspondientes de la tabla. El nivel inferior, u hoja, de un índice clúster contiene las filas de datos reales de la tabla. Una tabla o vista permite un índice clúster al mismo tiempo. Una vista con un índice clúster único se denomina vista indizada. La creación de un índice clúster único en una vista materializa físicamente la vista. Es necesario crear un índice clúster único en una vista para poder definir otros índices en la misma vista. Cree el índice clúster antes de crear los índices no clúster. Los índices no clúster existentes en las tablas se vuelven a generar al crear un índice clúster. Si no se especifica CLUSTERED, se crea un índice no clúster.
- NONCLUSTERED: Crea un índice que especifica la ordenación lógica de una tabla. Con un índice no clúster, el orden físico de las filas de datos es independiente del orden indizado. Cada tabla puede tener hasta 999 índices no clúster, independientemente de cómo se crean: de forma implícita con las restricciones PRIMARY KEY y UNIQUE, o explícita con CREATE INDEX. Para las vistas indizadas, sólo se pueden crear índices no clúster en una vista que ya tenga definido un índice clúster único. El valor predeterminado es NONCLUSTERED.
- *index_name:* Es el nombre del índice. Los nombres de índice deben ser únicos en una tabla o vista, pero no es necesario que sean únicos en una base de datos. Los nombres de índice deben seguir las reglas de los identificadores.
- *Column:* Es la columna o columnas en las que se basa el índice. Especifique dos o más nombres de columna para crear un índice compuesto sobre los valores combinados de las columnas especificadas. Enumere las columnas que desee incluir en el índice compuesto (en orden de prioridad) entre paréntesis después de *table_or_view_name*. Se pueden combinar hasta 16 columnas en la clave de un único índice compuesto. Todas las columnas de una clave del índice compuesto deben encontrarse

en la misma tabla o vista. El tamaño máximo permitido de los valores de índice combinado es 900 bytes. Las columnas de tipos de datos de objetos grandes (LOB) ntext, text, varchar(max), nvarchar(max), varbinary(max), xml o image no se pueden especificar como columnas de clave de un índice. Además, una definición de vista no puede incluir columnas ntext, text ni image, aunque no se haga referencia a ellas en la instrucción CREATE INDEX. Puede crear índices en columnas de tipo definido por el usuario CLR si el tipo admite el orden binario. También puede crear índices en columnas calculadas que están definidas como invocaciones de método de una columna de tipo definido por el usuario, siempre que los métodos estén marcados como deterministas y no realicen operaciones de acceso a datos.

- [ ASC | DESC ]: Determina la dirección ascendente o descendente del orden de la columna de índice determinada. El valor predeterminado es ASC.
- INCLUDE (*column* [ ,... *n* ] ): Especifica las columnas sin clave que se agregarán en el nivel hoja del índice no clúster. El índice no clúster puede ser único o no único. Los nombres de columna no se pueden repetir en la lista INCLUDE y no se pueden utilizar simultáneamente como columnas con y sin clave. Los índices no clúster siempre contienen las columnas de índice clúster si se define un índice clúster en la tabla. Se admiten todos los tipos de datos, a excepción de text, ntext e image. El índice se debe crear o regenerar sin conexión (ONLINE = OFF) si el tipo de datos de alguna de las columnas sin clave especificadas es varchar(max), nvarchar(max) o varbinary(max). Las columnas calculadas que son deterministas, y precisas o imprecisas, pueden ser columnas incluidas. Las columnas calculadas derivadas de los tipos de datos image, ntext, text, varchar(max), nvarchar(max), varbinary(max) y xml pueden ser columnas sin clave incluidas, siempre que los tipos de datos de las columnas calculadas sean aceptables como columna incluida.
- WHERE <filter_predicate>: Crea un índice filtrado especificando qué filas se van a incluir en el índice. El índice filtrado debe ser un índice no clúster en una tabla. Crea las estadísticas filtradas para las filas de datos en el índice

filtrado.

- ON *partition_scheme_name*(*column_name*): Especifica el esquema de partición que define los grupos de archivos a los que se asignarán las particiones de un índice con particiones. El esquema de partición debe existir en la base de datos al ejecutar CREATE PARTITION SCHEME o ALTER PARTITION SCHEME. *column_name* especifica la columna en la que se crearán particiones del índice con particiones. Esta columna debe coincidir con el tipo de datos, la longitud y la precisión del argumento de la función de partición que utiliza *partition_scheme_name*. *column_name* no se restringe a las columnas en la definición de índice. Se pueden especificar todas las columnas de la tabla base, excepto en el caso de partición de un índice UNIQUE en el que se debe elegir un valor para *column_name* entre las columnas utilizadas como clave única. Esta restricción permite que Database Engine (Motor de base de datos) compruebe la unicidad de los valores de clave en una única partición solamente.
- ON *filegroup_name:* Crea el índice especificado en el grupo de archivos especificado. Si no se ha especificado una ubicación y la tabla o vista no tiene particiones, el índice utiliza el mismo grupo de archivos que la tabla o vista subyacente. El grupo de archivos debe existir previamente.
- ON "default": Crea el índice especificado en el grupo de archivos predeterminado. El término predeterminado (default), en este contexto, no es una palabra clave. Es un identificador para el grupo de archivos predeterminado y debe delimitarse, como en ON"default" o en ON [default]. Si se especifica "default", la opción QUOTED_IDENTIFIER debe establecerse en ON en la sesión actual. Ésta es la configuración predeterminada.
- [ FILESTREAM_ON { *filestream_filegroup_name* | *partition_scheme_name* | "NULL" } ]: Especifica la posición de datos FILESTREAM para la tabla cuando se crea un índice clúster. La cláusula FILESTREAM_ON permite mover los datos FILESTREAM a otro esquema de partición o a otro grupo de archivos FILESTREAM.
- *filestream_filegroup_name:* es el nombre de un grupo de archivos

FILESTREAM. El grupo de archivos debe tener un archivo definido para el grupo de archivos, utilizando para ello las instrucciones CREATE DATABASE o ALTER DATABASE; de lo contrario, se producirá un error.

## 6.6 Ejemplos.

Ejemplo 1: Crear un índice llamado IX_1 por la columna ID, indicando que la tabla Datos1 se ordenará físicamente por el índice solicitado:

```
CREATE CLUSTERED INDEX IX_1 ON Datos1 (ID)
```

Ejemplo2: Podemos ahora preguntarnos que pasaría si además es necesario realizar búsquedas por otro campo, supongamos por el campo Numero, en este caso no podremos reordenar la tabla físicamente por Numero, ya que al hacer esto perderíamos el orden físico que ya habíamos establecido por el campo ID, es claro que el orden físico puede establecerse solo para una clave (ya sea compuesta por un solo o varios campos). Para este caso creamos un índice, llamado IX_2, del tipo non-clustered, ya que no modifican el orden físico de los registros en la tabla original, estos índices guardarán en otra estructura una copia de los valores involucrados en la clave y un puntero al registro original de la tabla.

```
CREATE INDEX IX_2 ON Datos1 (Numero)
```

Ejemplo3: Crear un índice IX_3 non-clustered que incluya columnas que no son parte del mismo.

```
CREATE INDEX IX_3 ON Datos1 (Numero) INCLUDE (Descripcion, ID)
```

Ejemplo 4: Crear un índice filtrado, en éste caso solo se aplica a un grupo de datos.

```
CREATE INDEX IX_4 ON Datos1 (Numero,ID,Descripcion) WHERE Numero < 100
```

## 6.7 Eliminación de índices

La instrucción DROP INDEX elimina los índices de la base de datos de SQL Server actual. No puede usar la instrucción DROP INDEX para quitar un índice que tenga una restricción PRIMARY KEY o UNIQUE. Para quitar la restricción y después eliminar el índice, use ALTER TABLE con la cláusula DROP CONSTRAINT.

Si desea usar la instrucción DROP INDEX para eliminar un índice clúster y mover la tabla resultante a otro grupo de archivos o esquema de partición, especifique la opción MOVE TO

Ejemplo1: borra el índice IX_2, creado anteriormente.

```
DROP INDEX IX_2 ON Datos1
```

Ejemplo2: Elimina un índice clúster, PK_MyClusteredIndex, en línea y se mueven la tabla resultante (montón) y los datos FILESTREAM al esquema de partición MyPartitionScheme mediante las cláusulas MOVE TO y FILESTREAM ON

```
DROP INDEX PK_MyClusteredIndex
ON dbo.MyTable
MOVE TO MyPartitionScheme
FILESTREAM_ON MyPartitionScheme;
```

# Unidad VII. Vistas


Las vistas se encuentran en el Nivel de Vista de las bases de datos y permiten personalizar la información a la que debe tener acceso cada uno de los usuarios que se conecten a la base de datos.

## 7.1 Concepto de vistas

Una vista es una tabla virtual cuyo contenido está definido por una consulta. Al igual que una tabla real, una vista consta de un conjunto de columnas y filas de datos con un nombre. Sin embargo, a menos que esté indizada, una vista no existe como conjunto de valores de datos almacenados en una base de datos. Las filas y las columnas de datos proceden de tablas a las que se hace referencia en la consulta que define la vista y se producen de forma dinámica cuando se hace referencia a la vista.

Una vista actúa como filtro de las tablas subyacentes a las que se hace referencia en ella. La consulta que define la vista puede provenir de una o de varias tablas, o bien de otras vistas de la base de datos actual u otras bases de datos. Asimismo, es posible utilizar las consultas distribuidas para definir vistas que utilicen datos de orígenes heterogéneos. Esto puede resultar de utilidad, por ejemplo, si desea combinar datos de estructura similar que proceden de distintos servidores, cada uno de los cuales almacena los datos para una región distinta de la organización.

No existe ninguna restricción a la hora de consultar vistas y muy pocas restricciones a la hora de modificar los datos de éstas.

En la siguiente ilustración se muestra una vista basada en dos tablas.

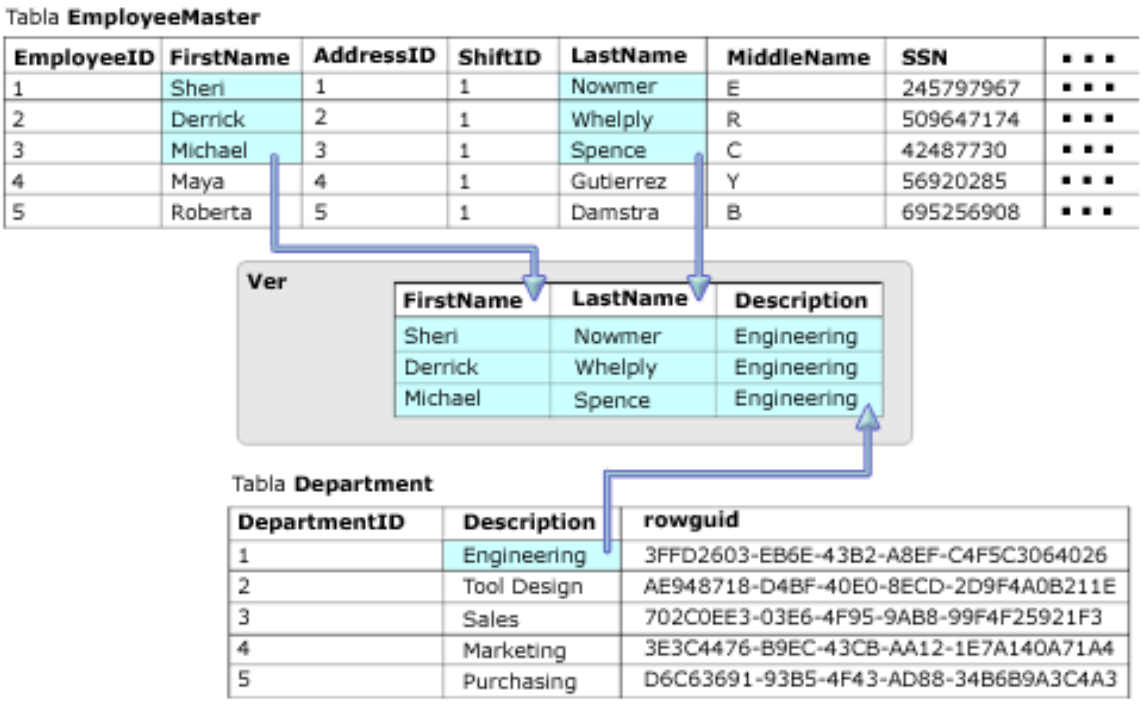



**Ilustración 1 Ejemplo de vista**

## 7.2 Tipos de vistas

- Vistas estándar : La combinación de datos de una o más tablas mediante una vista estándar permite satisfacer la mayor parte de las ventajas de utilizar vistas. Éstas incluyen centrarse en datos específicos y simplificar la manipulación de datos. Estas ventajas se describen pormenorizadamente en Escenarios de utilización de vistas.
- Vistas indizadas : Una vista indizada es una vista que se ha materializado. Esto significa que se ha calculado y almacenado. Se puede indizar una vista creando un índice agrupado único en ella. Las vistas indizadas mejoran de forma considerable el rendimiento de algunos tipos de consultas. Las vistas indizadas funcionan mejor para consultas que agregan muchas filas. No son adecuadas para conjuntos de datos subyacentes que se actualizan frecuentemente. Para obtener más información, vea Diseñar vistas indizadas.
- Vistas con particiones: Una vista con particiones junta datos horizontales con particiones de un conjunto de tablas miembro en uno o más servidores.

Esto hace que los datos aparezcan como si fueran de una tabla. Una vista que junta tablas miembro en la misma instancia de SQL Server es una vista con particiones local. Cuando una vista junta datos de tablas de servidores, es una vista con particiones distribuida. Las vistas con particiones distribuidas se usan para implementar una federación de servidores de bases de datos. Una federación es un grupo de servidores que se administran independientemente, pero que colaboran para compartir la carga de proceso de un sistema. Formar una federación de servidores de base de datos mediante la partición de datos es el mecanismo que permite ampliar horizontalmente un conjunto de servidores para admitir los requisitos de procesamiento de sitios Web de varios niveles y de gran tamaño.

## 7.3 Utilización de una vista.

Las vistas suelen utilizarse para centrar, simplificar y personalizar la percepción de la base de datos para cada usuario. Las vistas pueden emplearse como mecanismos de seguridad, que permiten a los usuarios obtener acceso a los datos por medio de la vista, pero no les conceden el permiso de obtener acceso directo a las tablas base subyacentes de la vista. Las vistas pueden utilizarse para proporcionar una interfaz compatible con versiones anteriores con el fin de emular una tabla que existía pero cuyo esquema ha cambiado. También pueden utilizarse para copiar datos entre Microsoft SQL Server a fin de mejorar el rendimiento y crear particiones de los datos.

- Las vistas permiten a los usuarios centrarse en datos de su interés y en tareas específicas de las que son responsables. Los datos innecesarios o sensibles pueden quedar fuera de la vista.
- Las vistas permiten simplificar la forma en que los usuarios trabajan con los datos. Las combinaciones, proyecciones, consultas UNION y consultas SELECT que se utilizan con frecuencia pueden definirse como vistas para que los usuarios no tengan que especificar todas las condiciones y calificaciones cada vez que realicen una operación adicional en los datos. Por ejemplo, es posible crear como vista una consulta compleja que se

utilice para la elaboración de informes y que realice subconsultas, combinaciones externas y agregaciones para recuperar datos de un grupo de tablas. La vista simplifica el acceso a los datos ya que evita la necesidad de escribir o enviar la consulta subyacente cada vez que se genera el informe; en lugar de eso, se realiza una consulta en la vista.

- Las vistas permiten crear una interfaz compatible con versiones anteriores para una tabla cuando su esquema cambia.
- Las vistas permiten que varios usuarios puedan ver los datos de modo distinto, aunque estén utilizando los mismos simultáneamente. Esto resulta de gran utilidad cuando usuarios que tienen distintos intereses y calificaciones trabajan con la misma base de datos. Por ejemplo, es posible crear una vista que recupere únicamente los datos para los clientes con los que trabaja el responsable comercial de una cuenta. La vista puede determinar qué datos deben recuperarse en función del Id. de inicio de sesión del responsable comercial que utilice la vista.
- Es posible utilizar vistas para exportar datos a otras aplicaciones.
- El operador de conjuntos UNION de Transact-SQL puede utilizarse dentro de una vista para combinar los resultados de dos o más consultas de tablas distintas en un solo conjunto de resultados. Esta combinación se muestra al usuario como una tabla única denominada vista con particiones. Por ejemplo, si una tabla contiene datos de ventas de Washington y otra tabla contiene datos de ventas de California, podría crearse una vista a partir de la UNION de ambas tablas. La vista representada incluye los datos de ventas de ambas zonas

## 7.4 Sintaxis General

```
CREATE VIEW [ schema_name . ] view_name [ (column [ ,...n ] ) ]
[ WITH <view_attribute>[ ,...n ] ]
AS
select_statement
[ WITH CHECK OPTION ] [ ; ]
<view_attribute> ::=
{
   [ ENCRYPTION ]
   [ SCHEMABINDING ]
```

```
   [ VIEW_METADATA ]
}
```

- *schema_name:* Es el nombre del esquema al que pertenece la vista. *view_name:* Es el nombre de la vista. Los nombres de las vistas deben cumplir las reglas de losidentificadores. La especificación del nombre del propietario de la vista es opcional.
- *Column:* Es el nombre que se va a utilizar para una columna en una vista. Sólo se necesita un nombre de columna cuando una columna proviene de una expresión aritmética, una función o una constante; cuando dos o más columnas puedan tener el mismo nombre, normalmente debido a una combinación; o cuando una columna de una vista recibe un nombre distinto al de la columna de la que proviene. Los nombres de columna se pueden asignar también en la instrucción SELECT. Si no se especifica el parámetro column, las columnas de la vista adquieren los mismos nombres que las columnas de la instrucción SELECT.
- AS*:* Especifica las acciones que va a llevar a cabo la vista.
- *select_statement:* Es la instrucción SELECT que define la vista. Dicha instrucción puede utilizar más de una tabla y otras vistas. Se necesitan permisos adecuados para seleccionar los objetos a los que se hace referencia en la cláusula SELECT de la vista que se ha creado. Una vista no tiene por qué ser un simple subconjunto de filas y de columnas de una tabla determinada. Es posible crear una vista que utilice más de una tabla u otras vistas mediante una cláusula SELECT de cualquier complejidad. En una definición de vista indizada, la instrucción SELECT debe ser una instrucción de una única tabla o una instrucción JOIN de varias tablas con agregación opcional. Las cláusulas SELECT de una definición de vista no pueden incluir lo siguiente: Una cláusula ORDER BY, a menos que también haya una cláusula TOP en la lista de selección de la instrucción SELECT.
- CHECK OPTION: Exige que todas las instrucciones de modificación de datos ejecutadas en la vista sigan los criterios establecidos en *select_statement*. Cuando una fila se modifica mediante una vista, WITH CHECK OPTION garantiza que los datos permanezcan visibles en toda la

vista después de confirmar la modificación.

- ENCRYPTION: Cifra las entradas de sys.syscomments que contienen el texto de la instrucción CREATE VIEW. El uso de WITH ENCRYPTION evita que la vista se publique como parte de la replicación de SQL Server.
- SCHEMABINDING: Enlaza la vista al esquema de las tablas subyacentes. Cuando se especifica SCHEMABINDING, las tablas base no se pueden modificar de una forma que afecte a la definición de la vista. En primer lugar, se debe modificar o quitar la propia definición de la vista para quitar las dependencias en la tabla que se va a modificar. Cuando se utiliza SCHEMABINDING, *select_statement* debe incluir los nombres de dos partes (*schema*.*object*) de las tablas, vistas o funciones definidas por el usuario a las que se hace referencia. Todos los objetos a los que se hace referencia se deben encontrar en la misma base de datos. Las vistas o las tablas que participan en una vista creada con la cláusula SCHEMABINDING no se pueden quitar a menos que se quite o cambie esa vista de forma que deje de tener un enlace de esquema. En caso contrario, Database Engine (Motor de base de datos) genera un error. Además, la ejecución de las instrucciones ALTER TABLE en tablas que participan en vistas que tienen enlaces de esquema provoca un error si estas instrucciones afectan a la definición de la vista.No es posible especificar SCHEMABINDING si la vista contiene columnas de tipo de datos de alias.
- VIEW_METADATA: Especifica que la instancia de SQL Server devolverá a las API de DB-Library, ODBC y OLE DB la información de metadatos sobre la vista en vez de las tablas base cuando se soliciten los metadatos del modo de exploración para una consulta que hace referencia a la vista.

## 7.5 Ejemplos

Ejemplo 1: La vista vBikes en la base de datos de ejemplo AdventureWorks permitiría a un usuario ver las existencias de bicicletas disponibles actualmente. La vista filtra todos los campos de la tabla Product salvo Name y sólo devuelve los nombres de las bicicletas completas en lugar de los componentes de bicicleta

```
CREATE VIEW vBikes
AS
```

```
SELECT DISTINCT p.[Name]
FROM Production.Product p JOIN Production.ProductInventory i ON p.ProductID = i.ProductID JOIN Production.ProductSubCategory ps ON p.ProductSubcategoryID = ps.ProductSubCategoryID JOIN Production.ProductCategory pc ON (ps.ProductCategoryID = pc.ProductCategoryID AND pc.Name = N'Bikes') AND i.Quantity > 0
```

Ejemplo 2: Una aplicación puede haber hecho referencia a una tabla no normalizada que tiene el siguiente esquema:

```
Employee(Name, BirthDate, Salary, Department, BuildingName)
```

Para evitar el almacenamiento redundante de datos en la base de datos, puede normalizar la tabla dividiéndola en las dos siguientes tablas:

```
Employee2(Name, BirthDate, Salary, DeptId)
Department(DeptId, BuildingName)
```

Para proporcionar una interfaz compatible con versiones anteriores que siga haciendo referencia a los datos de Employee, puede eliminar la tabla Employee antigua y reemplazarla por la siguiente vista:

```
CREATE VIEW Employee
AS
SELECT Name, BirthDate, Salary, BuildingName
FROM Employee2 e, Department d
WHERE e.DeptId = d.DeptId
```

Las aplicaciones que realizaban consultas en la tabla Employee, ahora pueden obtener sus datos desde la vista Employee. No es necesario cambiar la aplicación si sólo lee desde Employee.

## 7.6 Modificar una vista

Después de definir una vista, puede cambiar su nombre o modificar su definición sin tener que quitar la vista ni volver a crearla. Quitar una vista y volver a crearla provoca que se pierdan los permisos asociados a la ella. Cuando vaya a cambiar el nombre de una vista, tenga en cuenta las siguientes directrices:

- La vista cuyo nombre vaya a cambiar debe encontrarse en la base de datos actual.
- El nuevo nombre debe seguir las reglas definidas para los identificadores.
- Sólo puede cambiar el nombre de las vistas para las que tiene permiso.
- El propietario de la base de datos puede cambiar el nombre de las vistas de

cualquier usuario.

La modificación de una vista no afecta a los objetos dependientes, como pueden ser los procedimientos almacenados o los desencadenadores, a menos que la definición de la vista cambie de tal modo que el objeto dependiente deje de ser válido.
También puede modificar una vista para cifrar su definición o para garantizar que todas las instrucciones que impliquen una modificación de datos y que se ejecuten en la vista, sigan los criterios especificados en la instrucción SELECT que define la vista. Un ejemplo, se define la vista: CREATE VIEW employees_view AS SELECT EmployeeID FROM HumanResources.Employee Se modifica la vista como: ALTER VIEW employees_view AS SELECT LastName FROM Person.Contact c JOIN HumanResources.Employee e ON c.ContactID = e.ContactID

## 7.7 Eliminación de una vista

Después de crear una vista, puede eliminarla si no la necesita o si desea borrar la definición de la vista y los permisos asociados. Cuando se elimina una vista, las tablas y los datos en los que está basada no se ven afectados. Se producen errores al ejecutar las consultas, en el caso de que éstas utilicen objetos que dependen de la vista eliminada y a menos que se cree otra vista con el mismo nombre. No obstante, si la nueva vista no incluye columnas esperadas por los objetos dependientes de la misma, las consultas que utilicen los objetos que dependen de la vista dan lugar a errores cuando se ejecutan.
Ejemplo:

```
DROP VIEW employees_view
```

# Unidad VIII. Reportes


Los reportes son una forma muy especial de concentrar información a partir de consultas a la base de datos, su objetivo es apoyar en la toma de decisiones de las empresas u organizaciones.

## 8.1 SQL Server Reporting Services

Microsoft SQL Server 2008 Reporting Services (SSRS) dispone de una gama completa de herramientas y servicios listos para usar que le ayudarán a crear, implementar y administrar informes para la organización, así como de características de programación que le permitirán extender y personalizar la funcionalidad de los informes.

SQL Server 2008 Reporting Services (SSRS) es una plataforma de creación de informes basada en servidor que ofrece una completa funcionalidad de creación de informes para una gran variedad de orígenes de datos. Reporting Services contiene un completo conjunto de herramientas para crear, administrar y entregar informes, así como interfaces de programación de aplicaciones con las que los desarrolladores podrán integrar o extender el procesamiento de los datos y los informes en aplicaciones personalizadas. Las herramientas de Reporting Services trabajan en el entorno de Microsoft Visual Studio y están totalmente integradas con las herramientas y los componentes de SQL Server.

Con Reporting Services, puede crear informes interactivos, tabulares, gráficos o de forma libre a partir de orígenes de datos relacionales, multidimensionales o basados en XML. Puede publicar informes, programar el procesamiento de informes u obtener acceso a informes a petición. Reporting Services también permite crear informes ad hoc basados en modelos predefinidos, así como explorar interactivamente los datos del modelo. Puede elegir entre varios formatos de visualización, exporte informes a otras aplicaciones y suscribirse a los informes publicados. Los informes creados se pueden ver mediante una conexión web o como parte de una aplicación de Microsoft Windows o un sitio de SharePoint. Reporting Services proporciona la llave a sus datos empresariales.

## 8.2 Modo de implementación

SQL Server 2008 Reporting Services admite dos modos de implementación para las instancias del servidor de informes:

- Modo nativo, incluido el modo nativo con elementos web de SharePoint, donde un servidor de informes se ejecuta como un servidor de aplicaciones

que proporciona todas las capacidades de procesamiento y administración exclusivamente a través de los componentes de Reporting Services.

- Modo integrado de SharePoint, donde un servidor de informes se implementa como parte de un conjunto de servidores de SharePoint.

En el modo nativo, un servidor de informes es un servidor de aplicaciones independiente que proporciona todas las operaciones de visualización, administración, procesamiento y entrega de informes y modelos de informe. Se trata del modo predeterminado para las instancias del servidor de informes.
En el modo integrado de SharePoint, un servidor de informes se debe ejecutar en un conjunto de servidores de SharePoint. Un sitio de SharePoint proporciona el acceso front-end al contenido y las operaciones del servidor de informes. El servidor de informes proporciona todo el procesamiento y representación de los informes.
El modo integrado de SharePoint requiere Windows SharePoint Services 3.0 u Office SharePoint Server 2007, el Complemento Reporting Services para las tecnologías de SharePoint, y un servidor de informes configurado para el modo integrado de SharePoint. Un servidor de informes se configura para este modo si la base de datos del servidor de informes a la que está conectado puede almacenar datos de aplicación en un formato optimizado para sitios no jerárquicos y direccionamiento de documentos situado en una implementación de un producto o tecnología de SharePoint.

## 8.3 Creación de un proyecto de servidor de informes (Reporting Services)

Los proyectos de servidor de informes sirven para crear informes que se ejecutan en servidores de informes.
Pasos para crear un proyecto de servidor de informes

1. Haga clic en Inicio, seleccione Programas, Microsoft SQL Server 2008 y, a continuación, haga clic en Business Intelligence Development Studio.
2. En el menú Archivo, seleccione Nuevo y haga clic en Proyecto.
3. En la lista Tipos de proyecto, haga clic en Proyectos de Business Intelligence.
4. En la lista Plantillas, haga clic en Proyecto de servidor de informes.

5. En Nombre, escriba Tutorial.
6. Haga clic en Aceptar para crear el proyecto.

El proyecto Tutorial se muestra en el Explorador de soluciones. Ahora:

1. En el Explorador de soluciones, haga clic con el botón secundario en Informes, seleccione Agregar y haga clic en Nuevo elemento.
2. En el cuadro de diálogo Agregar nuevo elemento, debajo de Plantillas, haga clic en Informe.
3. En Nombre, escriba Sales Orders.rdl y, a continuación, haga clic en Agregar.

Se abrirá el Diseñador de informes y se mostrará el nuevo archivo .rdl en la vista Diseño.
El Diseñador de informes es un componente de Reporting Services que se ejecuta en Business Intelligence Development Studio. Tiene dos vistas: Diseño y Vista previa. Haga clic en cada ficha para cambiar las vistas.
Defina los datos en el panel Datos deinforme. El diseño del informe se define en la vista Diseño. Puede ejecutar el informe y ver su aspecto en la vista Vista previa.

## 8.4 Especificar información de conexión (Reporting Services)

Después de agregar un informe al proyecto Tutorial, necesita definir un *origen de datos*, que es un conjunto de información de conexión que el informe utiliza para tener acceso a los datos procedentes de una base de datos relacional, una base de datos multidimensional u otro recurso.
En éste ejemplo se usará la base de datos AdventureWorks2008 como origen de datos

1. En el panel Datos de informe, haga clic en Nuevo y, a continuación, haga clic en Origen de datos.
2. En Nombre, escriba AdventureWorks.
3. Asegúrese de que está seleccionado Conexión incrustada.
4. En Tipo, seleccione Microsoft SQL Server.
5. En Cadena de conexión, escriba lo siguiente:

```
Data source=localhost; initial catalog=AdventureWorks2008
```

6. Haga clic en Aceptar. Se agrega un origen de datos denominado AdventureWorks al panel Datos de informe.

## 8.5 Definir un conjunto de datos para el informe de tabla

Después de definir el origen de datos, necesita definir un conjunto de datos. En Reporting Services, los datos que se utilizan para los informes proceden de un *conjunto de datos*. Un conjunto de datos incluye un puntero a un origen de datos y la consulta que usará para el informe, así como campos y variables calculados.
Puede usar el Diseñador de consultas del Diseñador de informes para diseñar la consulta. Vamos a crear una consulta que recupere información sobre pedidos de ventas de la base de datos AdventureWorks2008.

1. En el panel Datos de informe, haga clic en Nuevo y, a continuación, haga clic en Conjunto de datos. Se abre el cuadro de diálogo Propiedades del conjunto de datos.
2. En el cuadro Nombre, escriba AdventureWorksDataset.
3. Asegúrese de que el nombre del origen de datos, AdventureWorks, está en el cuadro de texto Origen de datos y de que el Tipo de consulta es Texto.
4. Escriba, o copie y pegue, la siguiente consulta de Transact-SQL en el cuadro Consulta

```
SELECT soh.OrderDate AS [Date], soh.SalesOrderNumber AS [Order],
pps.Name AS Subcat, pp.Name as Product, SUM(sd.OrderQty) AS Qty,
SUM(sd.LineTotal) AS LineTotal
FROM Sales.SalesPerson sp
INNER JOIN Sales.SalesOrderHeader AS soh
ON sp.BusinessEntityID = soh.SalesPersonID
INNER JOIN Sales.SalesOrderDetail AS sd
ON sd.SalesOrderID = soh.SalesOrderID
INNER JOIN Production.Product AS pp
ON sd.ProductID = pp.ProductID
INNER JOIN Production.ProductSubcategory AS pps
ON pp.ProductSubcategoryID = pps.ProductSubcategoryID
INNER JOIN Production.ProductCategory AS ppc
ON ppc.ProductCategoryID = pps.ProductCategoryID
GROUP BY ppc.Name, soh.OrderDate, soh.SalesOrderNumber,
```

```
pps.Name, pp.Name,    soh.SalesPersonID
HAVING ppc.Name = 'Clothing'
```

5. (Opcional) Haga clic en el botón Diseñador de consultas. La consulta se muestra en el Diseñador de consultas basado en texto. Puede cambiar al diseñador gráfico de consultas haciendo clic en Editar como texto. Para ver los resultados de la consulta, haga clic en el botón Ejecutar (!) de la barra de herramientas del Diseñador de consultas. Verá los datos procedentes de seis campos de cuatro tablas distintas de la base de datos AdventureWorks2008. La consulta utiliza funcionalidad de Transact-SQL como los alias. Por ejemplo, la tabla SalesOrderHeader se denomina soh. Haga clic en Aceptar para salir del Diseñador de consultas.
6. Haga clic en Aceptar salir del cuadro de diálogo Propiedades del conjunto de datos.

Los campos del conjunto de datos AdventureWorksDataset aparecen en el panel Datos de informe

## 8.6 Agregar una tabla al informe

Después de definir un conjunto de datos, puede comenzar a definir el diseño del informe. El diseño del informe se crea arrastrando y colocando en la superficie de diseño regiones de datos, cuadros de texto, imágenes y otros elementos que se desean incluir en el informe.
Los elementos que contienen filas de datos repetidas procedentes de conjuntos de datos subyacentes se denominan *regiones de datos*. Normalmente, los informes solo contienen una región de datos, pero puede agregar más si, por ejemplo, desea agregar un gráfico al informe de tabla. Después de agregar una región de datos, puede agregar campos a la misma

1. En el Cuadro de herramientas, haga clic en Tabla y, a continuación, haga clic en la superficie de diseño. El Diseñador de informes dibuja una región de datos de tabla con tres columnas en el centro de la superficie de diseño
2. En el panel Datos de informe, expanda el conjunto de datos

AdventureWorksDataset para mostrar los campos.

3. Arrastre el campo Date desde el panel Datos de informe hasta la primera columna de la tabla.Al colocar el campo en la primera columna, suceden dos cosas. En primer lugar, la celda de datos mostrará el nombre del campo, que se conoce como la *expresión de campo*, entre corchetes: [Date]. En segundo lugar, se agrega automáticamente un valor de encabezado de columna a la fila Encabezado, inmediatamente encima de la expresión de campo. De forma predeterminada, la columna tiene el nombre del campo. Puede seleccionar el texto de la fila Encabezado y escribir un nuevo nombre.
4. Arrastre el campo Order desde el panel Datos de informe hasta la segunda columna de la tabla.
5. Arrastre el campo Product desde el panel Datos de informe hasta la tercera columna de la tabla.
6. Arrastre el campo Qty hasta el borde derecho de la tercera columna hasta que obtenga un cursor vertical y el puntero del mouse tenga un signo más [+]. Cuando suelte el botón, se creará una cuarta columna para [Qty].
7. Agregue el campo LineTotal de la misma manera, creando una quinta columna.

En el diagrama siguiente se muestra una región de datos de tabla rellenada con estos campos: Date, Order, Product, Qty y LineTotal.

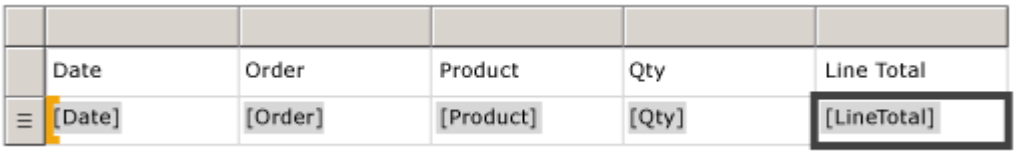



**Ilustración 2 Ejemplo de región de datos**

Al obtener una vista previa de un informe, se puede ver con facilidad el informe representado sin tener que publicarlo antes en un servidor de informes. Es probable que desee obtener frecuentemente una vista previa de un informe durante su diseño

# 8.7 Aplicar formato a un informe (Reporting Services)

## 8.7.1 Dar formato a un campo fecha

1. Haga clic en la ficha Diseño.
2. Haga clic con el botón secundario en la celda con la expresión de campo [Date] y, a continuación, haga clic en Propiedades de cuadro de texto.
3. Haga clic en Número y, a continuación, en el campo Categoría, seleccione Fecha.
4. EnelcuadroTipo,seleccioneJanuary31,2000.
5. Haga clic en Aceptar.

## 8.7.2 Dar formato a un campo moneda

1. Haga clic con el botón secundario en la celda con la expresión de campo [LineTotal] y, a continuación, haga clic en Propiedades de cuadro de texto.
2. Haga clic en Número y, en el campo Categoría, seleccione Moneda.
3. Si la configuración regional es Inglés (Estados Unidos), los valores predeterminados deberían ser:

    a. Decimales: 2
    b. Números negativos: ($12345.00)
    c. Símbolo: $ Inglés (Estados Unidos)

4. Seleccione Usar separador de miles.Si el texto de ejemplo es: $12,345.00, la configuración es correcta.
5. Haga clic en Aceptar.

# 8.8 Agregar grupos y totales (Reporting Services)

## 8.8.1 Agrupar datos en un informe

1. Haga clic en la ficha Diseño.
2. En el panel Datos de informe, arrastre el campo Date hasta el panel Grupos de filas. Sitúelo encima de la fila denominada Detalles.

    Observe que el identificador de fila ahora tiene un corchete para mostrar un grupo. Ahora, la tabla también tiene dos columnas Date, una de ellas en uno de los dos extremos de una línea de puntos vertical.

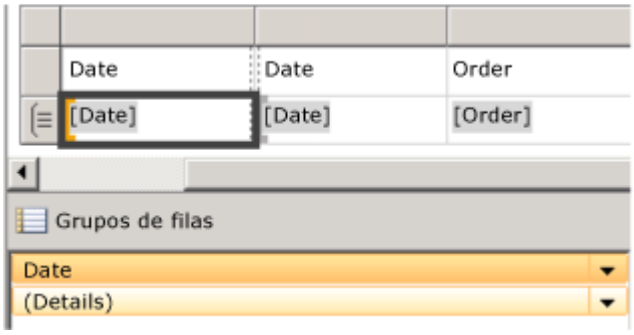



**Ilustración 3 Modificador de línea**

3. En el panel Datos de informe, arrastre el campo Order hasta el panel Grupos de filas. Sitúelo debajo de Date y encima de Detalles. Observe que el identificador de fila ahora tiene dos corchetes para mostrar dos grupos. Ahora, la tabla también tiene dos columnas Order
4. Elimine las columnas Date y Order originales situadas a la derecha de la línea doble. Esta acción quita los valores de este registro para que solo se muestre el valor de grupo. Seleccione los identificadores de las dos columnas y haga clic con el botón secundario en Eliminar columnas.
5. Cambie a la ficha Vista previa para obtener la vista previa del informe.

## 8.8.2 Agregar totales a un informe

1. Cambie a la vista Diseño.
2. Haga clic con el botón secundario en la celda de la región de datos que contiene el campo [LineTotal] y haga clic en Agregar total. Esto agrega una fila con la suma de los importes de los pedidos.
3. Haga clic con el botón secundario en la celda que contiene el campo [Qty] y haga clic en Agregar total. Esto agrega la suma de los importes de los pedidos a la fila de totales.
4. En la celda vacía situada a la izquierda de Sum[Qty], escriba la etiqueta "Order Total".
5. Puede agregar un color de fondo a la fila de totales. Seleccione las dos celdas que contienen las sumas y la celda con la etiqueta.
6. En el menú Formato, haga clic en Color de fondo y, a continuación, haga clic en Gris claro.

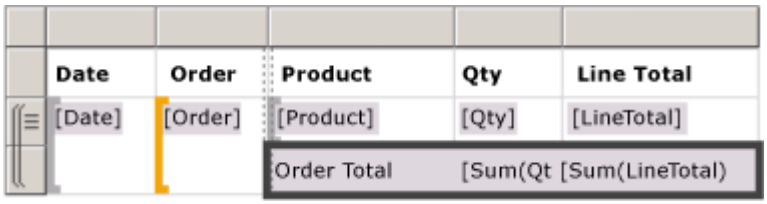



**Ilustración 4 Ejemplo de totales**

# Bibliografía.

http://msdn.microsoft.com/http://www.aulaclic.es/sql/t_3_1.htm

http://msdn.microsoft.com/es-es/library/ms180026.aspx

http://msdn.microsoft.com/es-es/library/ms187953.aspx

http://msdn.microsoft.com/es-es/library/ms189550%28v=SQL.100%29.aspx

http://www.aulaclic.es/sqlserver/t_9_9.htm http://msdn.microsoft.com/es-es/library/ms176118(v=SQL.100).aspx

http://www.programandoamedianoche.com/2009/09/indices-filtrados-en-sql-server-2008/