Capítulo 3: SQL

Fundamentos de Bases de datos, 5ª Edición.

©Silberschatz, Korth y Sudarshan
Consulte www.db-book.com sobre condiciones de uso

Capítulo 3: SQL

- Definición de datos
- Estructura básica de las consultas
- Operaciones sobre conjuntos
- Funciones de agregación
- Valores nulos
- Subconsultas anidadas
- Consultas complejas
- Vistas
- Modificación de la base de datos
- Reunsión de relaciones**

Modelo de Datos

SQL – Definicion del Modelo

PROVEEDORES

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS

PARTES

P#	PNOMBRE	COLOR	PESO	CIUDAD
P1	TUERCA	ROJO	12	LONDRES
P2	PERNO	VERDE	17	PARIS
P3	BIRLO	AZUL	17	ROMA
P4	BIRLO	ROJO	14	LONDRES
P5	LEVA	AZUL	12	PARIS
P6	ENGRANAJE	ROJO	19	LONDRES

ENVIOS

S#	P#	CANT
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	400
S4	P2	200
S4	P4	300
S4	P5	400

©Silberschatz, Korth y Sudarshan

Historia

- IBM Sequel language developed as part of System R project at the IBM San Jose Research Laboratory
  - Nombre Original SEQUEL (by IBM)
  - Prototipo version 1 fue “SYSTEM R”
  - IBM desarrolla una linea de productos para DB2, OS/2, OS/400
  - Varios proveedores comienzan a desarrollar algo similar para sus productos.
- Renamed Structured Query Language (SQL)
- ANSI and ISO standard SQL:
  - SQL-86
  - SQL-89
  - SQL-92
  - SQL:1999 (language name became Y2K compliant!)
  - SQL:2003
- Commercial systems offer most, if not all, SQL-92 features, plus varying feature sets from later standards and special proprietary features.
  - Not all examples here may work on your particular system.

SQL separamos DDL y DML

- DDL (Lenguaje de Definicion de Datos)
  - CREATE TABLE
  - CREATE VIEW
  - CREATE INDEX
  - ALTER TABLE
  - DROP TABLE
  - DROP VIEW
  - DROP INDEX
- DML (Lenguaje de Manipulacion de Datos)
  - SELECT
  - UPDATE
  - DELETE
  - INSERT

©Silberschatz, Korth y Sudarshan

Data Definition Language

DDL no sólo especifica información sobre la relación, sino sobre el conjunto de las relaciones, incluyendo:

- El esquema de cada relación.
- Los valores de dominio asociados a cada atributo.
- Integridad de los datos
- Los indices que mantienen la relación entre las tablas.
- Información sobre la seguridad para el acceso a cada relación.
- La estructura física de cada relación en el disco.

Tipos de dominio en SQL

- **char(n).** Caracter de Longitud fija, especificada en la variable *n.*
- **varchar(n).** Caracter de longitud varialbe, con un máximo especificado de longitud indicado en *n.*
- **int.** Entero (un valor entero especificado que depende de la máquina).
- **smallint.** Entero pequeño (un valor entero especificado que depende de la máquina).
- **numeric(p,d).** Número fijo con decimales, donde el usuario especifica la precisión de *p* digitos, con *n* digitos del punto decimal.
- **real, double precision.** Número de punto flotante y de doble precisión, depende de la precisión de la máquina.
- **float(n).** Número de punto flotante donde el usuario especifica la precisión de *n* digitos.
- Mas datos en cap.4.

©Silberschatz, Korth y Sudarshan

Sentencia Create Table

- Una relacion SQL es definida por el usuario con la sentencia **create table** :

  create table r (A_1 D_1, A_2 D_2, ..., A_n D_n,
  (regla-integridad$_1$),
  ...,
  (regla-integridad$_k$))

  - r es el nombre de la relación
  - cada A_i es el nombre del atributo en el esquema de la relación r
  - D_i es el valor y tipo de datos del atributo A_i

- Ejemplo:

  create table *proveedor*
  (*snum* integer **not null,**
  snombre char(30) **not null,**
  situacion char(1),
  ciudad char(30))

Regla-Integridad en Create Table

- **not null**
- **primary key** (A_1, ..., A_n)

Ejemplo: Para declarar *snum* como una llave primaria de la relación *proveedor*.

create table *proveedor*
(*snum* integer,
snombre char(30) **not null,**
situacion char(1),
ciudad char(30),
primary key (*snum*))

Declarar **primary key** un atributo automaticamente asegura que el atribto es **not null** desde SQL-92, esto debía ser especificado hasta el SQL-89

Sentencia Drop and Alter Table

- La sentencia **drop table** elimina toda la información contenida en la tabla y elimina la tabla de la base de datos.
- El comando **alter table** es usado para agregar atributos a una relación existente:

  alter table *r* **add** *A D*

  donde A es el nombre de los atributos a agregar en la relación *r* y *D* es el dominio de *A.*
  - Todas los nuevos atributos que se agregan en la relación son asignados como que admiten *null* y llenados para todas las tuplas con ese valor.
- El comando **alter table** se puede utilizar para eliminar atributos de una relación :

  alter table *r* **drop** *A*

  donde A es el nombre del atributo que se elimina de la relación *r*
  - Eliminación de atributos no es aceptado por todos los motores bases de datos

Estructura básica de las consultas

- SQL está basado en operaciones relacionales y de conjunto con ciertas modificaciones y mejoras
- Una consulta característica de SQL tiene la forma:

  select $A_1, A_2, ..., A_n$
  from $r_1, r_2, ..., r_m$
  where P

  - A_i representan los atributos
  - r_i representan las relaciones
  - P es un predicado
- Esta consulta es equivalente a la expresión del álgebra relacional.

$$\Pi_{A1, A2, ..., An}(\sigma_P (r_1 \times r_2 \times ... \times r_m))$$

- El resultado de una consulta de SQL es una relación.

La claúsula SELECT

SELECT [distinct] elemento(s)

FROM tabla(s)

[WHERE condicion]

[GROUP BY campo(s)]

[HAVING condicion]

[ORDER BY campo(s)]

Importante: Todo resultado de una consulta de la sentencia SELECT es una TABLA

La cláusula select

- La cláusula **select** se utiliza para dar la relación de los atributos deseados en el resultado de una consulta
  - corresponde a la operación de proyección del álgebra
- Ejemplo: obtener los nombres de todos los proveedores de la realación proveedor:

  select *snombre*
  from *proveedores*
- En la sintaxis del álgebra relacional “puro”, la consulta debería ser:

$$\Pi_{snombre}\,(proveedores)$$

- NOTA: los nombres de SQL son de tipo de letra insensitivo (lo que significa que se puede utilizar las mayúsculas o minúsculas.)
  - Puede ser deseable utilizar las mayúsculas en los lugares en los que utilizamos la fuente en negrita.

La cláusula select (Cont.)

- SQL permite los duplicados en las relaciones además de en los resultados de la consulta.
- Para forzar la eliminación de duplicados, insertar la clave **distinct** después de select.
- Obtener los nombres de todas las ciudades en las relaciones proveedores , y anular los duplicados

  select distinct ciudad
  from *proveedores*

- La clave **all** especifica que los duplicados no se han anulado.

  select all *ciudad*
  from *proveedores*

La cláusula select (Cont.)

- Un asterisco (*) en la cláusula select indica “todos los atributos”

  select *
  from *proveedores*

- La cláusula **select** puede contener expresiones aritméticas que involucran la operación, +, –, * y /, y que funcionan en las constantes o en los atributos de las tuplas.
- La consulta:

  select *snum, snombre, situacion * 100, ciudad*
  from *proveedores*

  volverá a una relación que es la misma que las relaciones *proveedor*, excepto que el atributo *situacion* se multiplica por 100.

La cláusula where

- La cláusula **where** especifica las condiciones que debe satisfacer el resultado
  - Corresponde al predicado de la selección del álgebra relacional.
- La búsqueda de todos los proveedores cuya situacion es mayor que 20 y la ciudad donde se encuentran es igual a “París”.

  select *
  from *proveedores*
  where *ciudad* = *‘París’* **and** *situacion* > 20
- Los resultados de la comparación se pueden combinar utilizando las conectivas lógicas **and, or** y **not.**
- Las comparaciones se pueden aplicar a los resultados de las expresiones aritméticas.

La cláusula where (Cont.)

- SQL incluye un operador de comparación **between**
- Ejemplo: Obtener el nombre de los proveedores cuya situación está entre 15 y 40

select *snombre*
from *proveedores*
where *situacion* **between** 15 **and** 40

La cláusula from

- La cláusula **from** hace una lista de las relaciones que se van a explorar en la evaluación de la expresión
  - Corresponde a la operación del producto cartesiano del álgebra relacional.
- Buscar el producto cartesiano *proveedores X partes*

select ∗
from *proveedores, partes*

- Obtener todas las combinaciones de información de proveedores y partes tales que el proveedor y la parte en cuestión estén situados en la misma ciudad.

select *proveedores.*, partes.**
from *proveedores, partes*
where *proveedores.ciudad =partes.ciudad*

La operación de renombramiento

- SQL permite las relaciones y atributos de renombramiento utilizando la cláusula **as**:

  nombre_antiguo **as** *nombre_nuevo*

- Obtener los datos del proveedor renombrando el snum por numero_proveedor.

select *snum* **as** *numero_proveedor, snombre, situacion, ciudad*
from *proveedores*

Variables tupla

- Las variables tupla se definen en la cláusula **from** mediante el uso de la cláusula **as**.
- Obtener los nombres y situación de los proveedores cuya ciudad sea “Londrés”.

select *p.snombre, p.situacion*
from *proveedores* **as** *P*
where *P.ciudad = “Londres”*

- Obtener todas las parejas de proveedores que viven en la misma ciudad.

select distinct *P1.*, P2.**
from *proveedores* **as** *P1, proveedores* **as** *P2*
where *P1.ciudad = P2.ciudad* **and** *P1.snum < P2.snum ’*

Operaciones con cadenas

- SQL incluye un operador de coincidencia de cadenas para comparaciones de cadenas de caracteres. *El operador “like” utiliza además los siguientes comodines:
  - tanto por ciento(%). El carácter % encaja con cualquier subcadena.
  - guión bajo (_). El carácter _ encaja con cualquier carácter.
- Obtener los nombres de todos los proveedores cuyos nombres incluyan las letras “al”.

  select *snombre*
  from *proveedores*
  where *snombre* **like** ‘%al%’
- SQL soporta una variable de operaciones con cadenas como
  - concatenacion (que utiliza “||”)
  - conversión de mayúscula a minúsculas(y viceversa)
  - Búsqueda de la longitud de la cadena, extracción de subcadena, etc.

Orden en la presentación de las tuplas

- Lista los nombres y situacion de los proveedores, ordenados por situación de menor a mayor

  select *snombre, situacion*
  from *proveedores*
  order by *situacion*

- Se puede especificar la cláusula **desc** para orden descendente o **asc** para orden ascendente, de cada atributo; el orden ascendente es el orden por defecto.
  - Ejemplo: **order by** *situacion* **desc**

Duplicados

- En las relaciones con duplicados, SQL permite definir cuantas copias de las tuplas aparecen en el resultado.
- Las versiones **multiconjunto** de algunos de los operadores del álgebra relacional – dadas las relaciones multiconjunto r_1 y r_2:

  1. $\sigma_\theta(r_1)$**:** Si hay c_1 copias de la tupla t_1 en r_1, y t_1 satisface las selecciones σ_θ, entonces hay c_1 copias de t_1 en $\sigma_\theta(r_1)$.
  2. $\Pi_A(r)$**:** Para cada copia tupla t_1 en r_1, hay una copiade la tupla $\Pi_A(t_1)$ en $\Pi_A(r_1)$ donde $\Pi_A(t_1)$ denota la poryección de la tupla singular t_1.
  3. $r_1 \times r_2$ **:** Si hay c_1 copias de la tupla t_1 en r_1 y c_2 copias de la tupla t_2 en r_2, hay c_1 x c_2 copias de la tupla t_1. t_2 en $r_1 \times r_2$

Duplicados (Cont.)

- Ejemplo: Supóngase que las relaciones multiconjunto r_1 (*A, B*) y r_2 (*C*) son las siguientes:

  $r_1 = \{(1, a)\ (2, a)\}$ $r_2 = \{(2), (3), (3)\}$

- Entonces $\Pi_B(r_1)$ debería ser{(a), (a)}, mientras $\Pi_B(r_1)$ x r_2 debería ser

  $\{(a,2), (a,2), (a,3), (a,3), (a,3), (a,3)\}$

- SQL duplica la semántica:

  select $A_1, A_2, \ldots, A_n$
  from $r_1, r_2, \ldots, r_m$
  where P

  es equivalente a la versión *multiconjunto* del la expresión:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(r_1 \times r_2 \times \ldots \times r_m))$$

Operaciones con conjuntos

- Las operaciones de conjuntos **union**, **intersect**, y **except** operan sobre relaciones y corresponden a las operaciones de álgebra relacional $\cup$, $\cap$, $-$.
- Cada una de las operaciones antes citadas elimina duplicados automáticamente; para retener todos los duplicados se utilizan las versiones de multiconjunto correspondientes **union all**, **intersect all** y **except all**.

  Supóngase que una tupla se produce *m* veces en *r* y *n* veces en *s*, entonces, se produce:

  - *m* + *n* veces en *r* **union all** *s*
  - min(*m,n)* veces en *r* **intersect all** *s*
  - max(0, *m* – *n)* veces en *r* **except all** *s*

Operaciones con conjuntos

- Obtener los proveedores y partes que se encuentran en “Paris”

  (select *snombre* **from** *proveedores* ***where*** *ciudad = “Paris”*)
  union
  (select *pnombre* **from** *partes* ***where*** *ciudad = “Paris”)*

- Obtener todas las ciudades donde hay proveedores y partes.

  (select *ciudad* **from** *proveedores)*
  intersect
  (select *ciudad* **from** *partes)*

- Obtener todos los proveedores que viven en una ciudad donde no hay partes.

  (select *ciudad* **from** *proveedores)*
  except
  (select *ciudad* **from** *partes)*

Funciones de agregación

- Estas funciones operan en el multiconjunto de valores de una columna de una relación, y devuelven un valor

  avg: valor medio
  min: valor mínimo
  max: valor máximo
  sum: suma de valores
  count: número de valores

Funciones de agregación (cont.)

- Obtener el valor promedio de la situación de los proveedores.

  select avg *(situacion)*
  from *proveedores*

- Obtener el número de tuplas de la relación *partes*

  select count (*)
  from *partes*

- Obtener la cantidad de ciudades distintas de la lista de proveedores

  select count (distinct *ciudad***)**
  from *proveedores*

Funciones de agregación – Group By

- Obtener la cantidad de piezas enviadas por cada proveedor.

select *snum,* **sum (***cant)*
from *envios*
group by *snum*

Nota: Los atributos de la cláusula **select** fuera de las funciones de agregación deben aparecer en la lista **group by**

select *p.snum, snombre,* **sum (***cant)*
from *envios* **as** *e, proveedores* **as** *p*
where *e.snum = p.snum*
group by *p.snum, snombre*

©Silberschatz, Korth y Sudarshan

Funciones de agregación – Cláusula Having

- Obtener los nombres de todas las partes cuyos envíos sean superiores a 800 unidades.

select *pnombre,* **sum** (cant)
from *partes* **as** *p, envios* **as** *e*
where *p.pnum = e.pnum*
group by *pnombre*
having sum (*cant*) > 800

Nota: los predicados de la cláusula **having** se aplican después de la formación de grupos mientras que los permitidos en la cláusula **where** se aplican antes de la formación de grupos

Valores nulos

- Es posible que las tuplas tengan un valor nulo, indicado por medio de *null*, en alguno de sus atributos
- *null* significa un valor desconocido o un valor que no existe.
- El predicado **is null** se puede utilizar para comprobar los valores nulos.
  - Ejemplo: obtener todos los proveedores cuya situación sea distinta de null

select *
from *proveedores*
where *situacion* **is null**

- El resultado de la expresión aritmética que involucra a *null* es nulo
  - Ejemplo: 5 + *null* devuelve nulo
- Sin embargo, las funciones de agregación simplemente ignoran los valores nulos
  - Se ofrecerá más información más adelante

Valores nulos y lógica de tres valores

- Cualquier comparación con *null* se convierte en *desconocido*
  - Ejemplo*: 5 < null o null <> null o null = null*
- Lógica de tres valores que utiliza el valor real desconocido:
  - OR: (*desconocido* **or** *cierto*) = *cierto*, (*desconocido* **or** *falso*) = *desconocido,* (*desconocido* **or** *desconocido) = desconocido*
  - AND: *(cierto* **and** *desconocido) = desconocido, (falso* **and** *desconocido) = falso, (desconocido* **and** *desconocido) = desconocido*
  - NOT*: (***not** *desconocido) = desconocido*
  - "*P* **is unknown"** se evalua a *cierto* si el predicado *P* se evalua a *desconocido*
- El resultado del predicado de la cláusula **where** se toma como *falso* si se evalúa en *desconocido*

Valores nulos y agregados

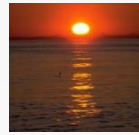
- El total de todas las situaciones de los proveedores

  select sum (*situacion*)
  from *proveedores*

  - La instrucción anterior ignora las cantidades nulas
  - El resultado es *null* si no hay cantidad no nula
- Todas las operaciones agregadas excepto **count(*)** ignoran las tuplas con valores nulos de los atributos agregados.

Subconsultas anidadas

- SQL proporciona un mecanismo para las subconsultas anidadas.
- Una subconsulta es una expresión **select-from-where** que se anida dentro de otra consulta.
- Un uso común de subconsultas es llevar a cabo comprobaciones sobre pertenencia a conjuntos, comparación de conjuntos y cardinalidad de conjuntos.



Ejemplo de consulta

- Obtener todos los nombres de proveedores que han realizado al menos un envío de productos.

```
select distinct snombre
        from proveedores
        where snum in (select snum
                       from envios )
```

- Obtener todos los proveedores que no realizaron ningún envío de mercadería

```
select *
        from proveedores
        where snum not in (select snum
                           from envios)
```

Ejemplo de consulta

- Obtener todos los proveedores de ‘Paris’ que hayan realizado el envío de un ‘Perno’

```
select *
from proveedores
where proveedores.ciudad = ‘Paris’ and
  (snum) in
          (select snum
            from partes as P, envios as E
            where p.pnum = e.pnum and
                p.pnombre = ‘Perno’ )
```

- Note: Se puede escribir la consulta anterior de forma mucho más simple. La formulación anterior es simplemente para ilustrar las características de SQL

- Obtener todos los proveedores que no realizaron ningún envío de la parte ‘Leva’

Comparación de conjuntos

- Obtener los nombres de todos los proveedores que realizaron un envío mayor que alguno de los envíos realizados de la parte ‘P4’

```
select distinct snombre
from proveedores as p, envios as e1, envios as e2
where p.snum = e1.snum and
      e1.cant > e2.cant and e2.pnum = 'P4'
```

- La misma consulta utilizando la clausula > **some**

```
select distinct snombre
from proveedores as p, envios as e1
where p.snum = e1.snum and
      e1.cant >some
            (select e2.cant
             from envios as e2
             where e2.pnum = 'P4' )
```

Definición de la clausula Some

- F <comp> **some** $r \Leftrightarrow \exists\, t \in r$ tal que (F <comp> t)
  donde <comp> puede ser: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **some** | 0 / 5 / 6 |) = verdadero
(leer: 5 < alguna tupla de la relación)

(5 < **some** | 0 / 5 |) = falso

(5 = **some** | 0 / 5 |) = verdadero

(5 ≠ **some** | 0 / 5 |) = verdadero (y que 0 ≠ 5)

(= **some**) ≡ **in**

Sin embargo, (≠ **some**) ≡ **not in**

Consulta ejemplo

- Obtener los nombres de todos los proveedores cuya situación es mayor que la situación de todos los proveedores de Londres.

```
select snombre
from proveedores
        where situacion > all
                (select situacion
                from proveedores
                where ciudad = ‘Londres’)
```

Definición de la cláusula all

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0 / 5 / 6 |) = falso

(5 < **all** | 6 / 10 |) = verdadero

(5 = **all** | 4 / 5 |) = falso

(5 ≠ **all** | 4 / 6 |) = verdadero (ya que 5 ≠ 4 y 5 ≠ 6)

(≠ **all**) ≡ **not in**

Sin embargo, (= **all**) ≠ **in**

©Silberschatz, Korth y Sudarshan

Comprobación de ausencia de tuplas duplicadas

- La construcción **unique** comprueba si una subconsulta tiene alguna tupla duplicada en sus resultados.
- Obtener todos los proveedores que sólo tengan un envío de partes.

select *p.snum*

from *proveedores* **as** *p*

where unique (select *e.snum*

from *envios* **as** *e* **)**

Consulta ejemplo

- Obtener todos los proveedores que al menos hayan realizado dos envíos de partes.

select *p.snum*
from *proveedores* **as** *p*
where unique (select *e.snum*
from *envios* **as** *e* **)**

Relaciones derivadas

- SQL permite utilizar expresiones de subconsulta en la cláusula **from**
- Obtener el número de parte y el envío promedio, en el cual dicho envío sea superior a 250 unidades.

```
select pnum
from ( select pnum, avg( cant )
       from envios
       group by pnum)
       as enviomedio( pnum, promedio )
where promedio > 250
```

Téngase en cuenta que no es necesario utilizar la cláusula **having**, puesto que se calcula la relación temporal (vista) resultado en la cláusula **from**, y los atributos de enviomedio se pueden utilizar directamente en la cláusula **where.**

Modificación de la base de datos– Borrado

DELETE	Eliminar datos de la Tabla
	DELETE **FROM** tabla [**WHERE** condicion]

Modificación de la base de datos– Borrado

- Borrar todos los proveedores de la ciudad de Londres

  delete from *proveedores*
  where *ciudad* = ‘Londres’

- Borrar todos los envíos de los proveedores de Paris.

  delete from *envios*
  where *snum* **in** (**select** *snum*
  from *proveedores*
  where *ciudad* = ‘Paris’)

Consulta ejemplo

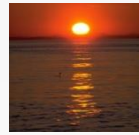
- Borrar todos los envíos inferiores a la media de los envíos.

delete from *envios*
where *cant* < (**select avg** (*cant*)
from *envios*)

- Problema: al borrar tuplas, el saldo medio cambia
- Solución utilizada en SQL:

  1. Primero, calcular el saldo medio **avg** (saldo) de todas las tuplas que se van a borrar

  2. Después, borrar todas las tuplas encontradas antes (sin recalcular avg (saldo) o recomprobando las tuplas)



Modificación de la base de datos– Inserción

INSERT	Agregar datos a la tabla
	INSERT **INTO** tabla [(campo [, campo [...]])] **VALUES** (literal [, literal [....]])

Modificación de la base de datos– Inserción

- Añadir una nueva tupla a *proveedores*

  insert into *proveedores*
  values (‘S6’, ‘Vulcano’,50,’Córdoba’)

  o equivalente

  insert into *proveedores (snum, snombre, situacion, ciudad)*
  values (‘S7’, ‘Vulcano’,50,’Córdoba’)

- Añadir una nueva tupla a la *partes* con *peso en null*

  insert into *partes*
  values (‘P7’,‘Tornillo’, ´Amarillo´, *null, ‘Córdoba’*)

Modificación de la base de datos– Inserción

- Se pueden insertar datos que se toman desde otra tabla

  insert into *envios1* ([*snum*] ,[*pnum*] ,[*cant*])

  select *snum*, *pnum*, *cant* **from** *envios*

- La sentencia **select from where** se evalúa completamente antes de que ninguno de sus resultados se inserte en la relación (de otra forma las consultas como

  insert into *tabla*1 **select** * **from** *tabla*1

  generarían problemas)

Modificación de la base de datos– Actualizaciones

UPDATE	Alterar o Modificar la Tabla
	UPDATE tabla SET campo = valor [, campo = valor] [WHERE condicion]

Modificación de la base de datos– Actualizaciones

- Aumentar todas las situaciones de los proveedores de Londres en un 20% y las situaciones de los proveedores de Paris en un 10%
  - Escribir dos instrucciones **update**:

update *proveedores*
set *situacion* = *situacion* ∗ 1,20
where *ciudad* = ‘Londres’

update *proveedores*
set *situacion* = *situacion* ∗ 1,10
where *ciudad* = ‘Paris’

  - El orden es importante
  - Se puede hacer utilizando la instrucción **case** (siguiente transparencia)

Instrucción case para actualizaciones condicionales

- Misma consulta que la anterior: Aumentar todas las situaciones de los proveedores según se detallo

```
update proveedores
set situacion =  case
                    when ciudad = ´Londres´ then situacion *1,20
                    when ciudad = ´Paris´ then situacion *1,10
                    else   situacion
                end
```

Reunión de relaciones**

- Las **operaciones de reunión** toman dos relaciones y las devuelven como resultado otra relación.
- Estas operaciones adicionales se utilizan generalmente como expresiones de subconsulta de la cláusula **from**
- **Condición de reunión** – define qué tuplas de las dos relaciones coinciden, y qué atributos están presentes en el resultado de la reunión.
- **Tipo de reunión** – define cómo se tratan las tuplas de cada relación que no coincide con ninguna tupla de la otra relación (basada en la condición de reunión).

Tipos de reunión
inner join **left outer join** **right outer join** **full outer join**

Condiciones de reunión
natural **on** <predicado> **using** (A_1, A_2, ..., A_n)

Reunión de relaciones – Conjuntos de datos para ejemplos

- Relación *proveedores*

S#	SNOMBRE	SITUACION	CIUDAD
S1	SALAZAR	20	LONDRES
S2	JAIMES	10	PARIS
S3	BERNAL	30	PARIS
S4	CORONA	20	LONDRES
S5	ALDANA	30	ATENAS

- Relación *envíos*

S#	P#	CANT
S1	P1	300
S1	P2	200
S1	P3	400
S1	P4	200
S1	P5	100
S1	P6	100
S2	P1	300
S2	P2	400
S3	P2	400
S4	P2	200
S4	P4	300
S6	P5	400

- *Nota: no se tiene la información de envíos para el proveedor S5 ni datos de proveedor para el envío para S6-P5-400*

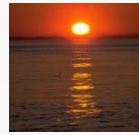
Reunión de relaciones – Ejemplos

- *proveedores* **inner join** *envios* **on** *proveedores.snum = envios.snum*

  Muestra todas las túplas donde existen las coincidencias, esto es similar a un producto cartesiano completo

- *proveedores* **left inner join** *envios* **on** *proveedores.snum = envios.snum*

  Muestra todas las túplas donde existen las coincidencias y además muestra todas las apariciones de proveedores y pone en null los valores para el envío.



Reunión de relaciones – Ejemplos

- *proveedores* **right outer join** *envios*

  Muestra todas las túplas donde existen las coincidencias y además muestra todas las apariciones de envíos y pone en null los valores para los proveedores.

- *proveedores* **full outer join** *envios*

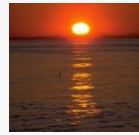
  Muestra todas las túplas donde existen las coincidencias y además muestra todas las apariciones de envíos y todas las apariciones de proveedores y pone en null los valores que no se encuentran.

Reunión de relaciones – Ejemplos

- *proveedores* ***inner join*** *envios* ***natural***
- *proveedores* **full outer join** *envios* **using** *(snum)*

Obtener todos los proveedores y envíos que no tienen sus consecuentes en la otra tabla.

```
select    *
from      proveedores1 full outer join envios1 on
          proveedores1.snum = envios1.snum
where     proveedores1.snum is null or
          envios1.snum is null
```



Vistas

- En algunos casos, no es deseable para todos los usuarios ver el modelo lógico completo (es decir, todas las relaciones actuales almacenadas en la base de datos).
- Considere una persona que necesita conocer sólo los proveedores que son de Londres y los envíos que han realizado. Esta persona debería ver una relación descrita en SQL como

  (**select** *snum, snombre, pnum, cant*
  from *proveedores, envios*
  where *proveedores.snum = envios.snum and*
  proveedores.ciudad = “Londres”)

- Una **vista** proporciona un mecanismo para ocultar ciertos datos de la vista de ciertos usuarios. Se simplifica la percepción del usuario.
- Cualquier relación que no es del modelo conceptual pero se hace visible para el usuario como una “relación virtual” se denomina una **view**.
- Se cuenta con seguridad automática para datos ocultos.

Definición de vista

- Una vista se define utilizando la instrucción **create view** que tiene la forma

  create view *v* **as** <expresión de consulta>

  donde <expresión de consulta> es cualquier expresión de consulta legal de SQL. El nombre de la vista se representa por *v.*
- Una vez definida la vista, su nombre puede utilizarse para referirse a la relación virtual que la vista genera.
- La definición de vista no es lo mismo que la creación de una nueva relación mediante la evaluación de la expresión de consulta.
  - Una definición de vista permite el ahorro de una expresión para ser sustituida por consultas que utilizan esa vista.

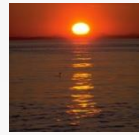
Consultas de ejemplo

- Una vista de los proveedores de Londres.

```
create view proveedores_londres as
    (select snum, snombre, situacion
     from proveedores
     where proveedores.ciudad = "Londres" )
```

- Listar los envios realizados por los proveedores de Londres

```
select *
        from proveedores_londres as pl, envios
        where pl.snum = envios.snum
```



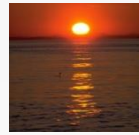
Vistas definidas en función de otras

- Una vista puede utilizarse en la expresión que define a otra vista
- Se dice que una relación de vistas v_1 *depende directamente de una relación de vistas* v_2, si v_2 se utiliza en la expresión que define a v_1
- Se dice que una relación de vistas v_1 *depende* de la relación de vistas v_2 tanto si v_1 depende directamente de v_2 como si hay un camino de dependencias de v_1 a v_2
- Se dice que una relación de vistas es *recursiva* si depende de sí misma

Expansión de vistas

- Una forma de definir el significado de las vistas definidas en términos de otras vistas.
- Sea la vista v_1 definida por una expresión e_1 que puede contener a su vez usos de relaciones de vistas.
- La expansion de vistas de una expresión repite la siguiente etapa de sustitución:

  repeat
  Averiguar todas las relaciones de vistas v_i en e_1
  Sustituir la relación de vistas v_i por la expresión que define v_i
  until no queden más relaciones de vistas en e_1
- Mientras las definiciones de vistas no sean recursivas, este bucle concluirá



CREATE VIEW

CREATE VIEW	Crea una vista de una tabla base
	CREATE VIEW vista [(columna [, columna [...]])] AS consulta
EJEMPLO	CREATE VIEW buenos_proveedores AS SELECT snum, situacion, ciudad FROM proveedores WHERE situacion > 15
	CREATE VIEW total_envios (pnum, cantotal) AS SELECT pnum, sum(cant) FROM envios GROUP BY pnum

Operaciones de DML en VISTAS

"No todas las vistas se pueden actualizar"

CREATE VIEW prov1 AS SELECT snum, ciudad FROM proveedores	CREATE VIEW prov2 AS SELECT situacion, ciudad FROM proveedores

- Puedo insertar un registro en la nueva vista.
- Puedo eliminar un registro.
- Puedo modificar un registro.

Analizar cada ejemplo

Otros casos:

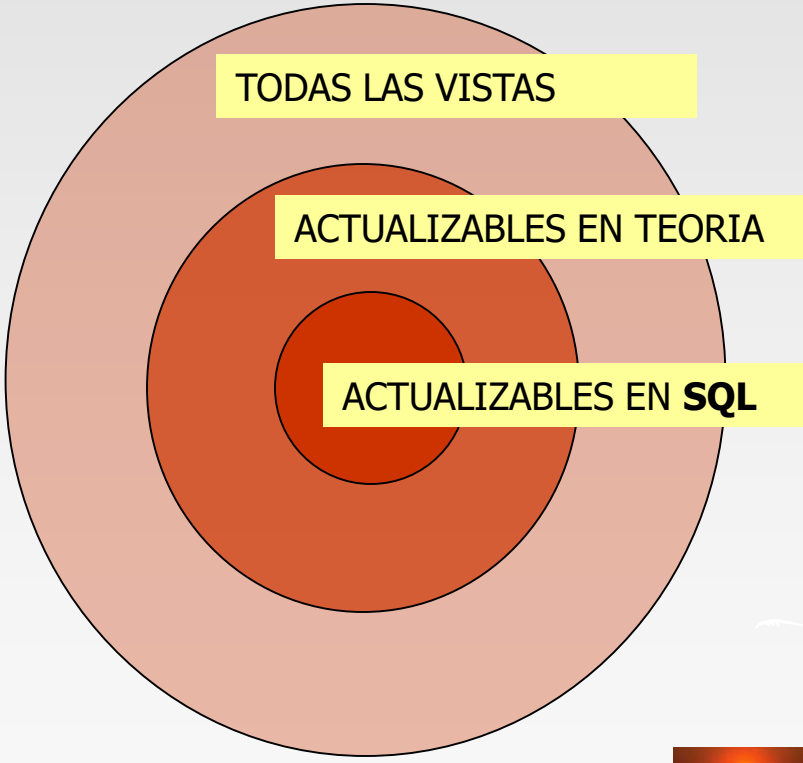
```
CREATE VIEW prov_londres
    AS  SELECT snum, snombre, situacion, ciudad
    FROM proveedores
    WHERE ciudad = “LONDRES”
```

```
CREATE VIEW cosituados (snum, snombre, situacion, prociudad,
                        pnum, pnombre, color, peso, parciudad)
    AS SELECT snum, snombre, situacion, pro.ciudad,
              pnum, pnombre, color, peso, par.ciudad
       FROM proveedores pro, partes par
       WHERE pro.ciudad = par.ciudad
```

```
CREATE VIEW total_envios ( pnum, cantotal )
     AS SELECT pnum, sum( cant )
        FROM envios
        GROUP BY pnum
```

Conclusión

- Según su naturaleza algunas vistas pueden ponerse al día pero otras no. (Tema pendiente de investigación)
- Cada producto avanza en esto de diferente manera. (Aún ninguno puede actualizar todas las vistas sin la ayuda del usuario)
- La mayoría permiten actualizar vistas que son subconjunto de filas y columnas (o combinacion) extraidos de una misma tabla.



SQL DDL – DROP VIEW

DROP VIEW	Elimina una vista de la base de datos
	DROP VIEW vista
EJEMPLO	DROP VIEW buenos_proveedores
	DROP VIEW total_envios

Fin del capítulo 3

©Silberschatz, Korth y Sudarshan
Consulte www.db-book.com sobre condiciones de uso