UNIVERSIDAD DE LA EMPRESA

CAPITULO 4
LENGUAJES DE CONSULTA

En la era de los avances tecnológicos, la obtención de los datos en forma rápida y eficiente es un tema que se investiga constantemente, estas investigaciones hacen surgir a los lenguajes de consulta.

4.1 INTRODUCCION

Los lenguajes de consulta nos permiten solicitar determinada información sobre nuestros datos. Este tipo de lenguajes son de alto nivel y le dan un mayor poder expresivo a nuestras consultas. Son una herramienta muy importante dentro del tema Base de Datos, ya que si no podemos realizar estas solicitudes sobre nuestra información, de una manera muy similar a nuestro lenguaje, caeríamos en la misma idea de la obtención de los datos en los sistemas de archivos. Estos lenguajes que estudiaremos los dividiremos en dos: lenguajes de tipo teórico y lenguajes comerciales. Los lenguajes de tipo teórico nos permiten representar la consultas de forma concisa, además en ellos se basan de alguna manera los lenguajes comerciales y son en determinada forma quienes definen el rumbo de como se deben formular las consultas. Los lenguajes de tipo comercial son aquellos que están implementados y se utilizan en distintos paquetes de software.

4.2 ALGEBRA RELACIONAL

El algebra relacional dispone de varios operadores los cuales se clasifican en dos clase: operadores fundamentales y operadores adicionales. Con los operadores fundamentales se pueden realizar todas las consultas que uno desea y las operaciones adicionales se pueden escribir en función de las fundamentales y la utilidad de ellas es que nos proporcionan mayor expresividad al algebra relacional.

Los operadores fundamentales son: *Proyección, selección, producto cartesiano, unión y diferencia.*

Los operadores adicionales son: *Join natural, theta join, intersección, cociente y renombrar atributos. En este curso solo daremos el theta join, para simplificar simplemente lo llamaremos join.*

Vamos a tomar como notación de instancia de relación a un conjunto I=(RS,T) siendo RS un esquema de relación primitivo ({A_1,...,A_n},DOM,dom} y T un conjunto de tuplas {t_1,...,t_m}.

Todas las operaciones del algebra se aplican sobre una expresión del algebra relacional y el resultado es otra expresión del algebra relacional.

4.2.1 Operadores fundamentales

Definición: *[Proyección]* Sea I=(RS,T) una instancia de relación, se definirá proyección de I sobre B como otra instancia $I_{PROY}=(RS_{PROY},T_{PROY})$ donde $RS_{PROY}=(\{B_1,...,B_r\},DOM_{PROY},dom_{PROY})$ con B incluido o igual a A y $T_{PROY}=\{t_1,...,t_m\}$ con $t_i=t(B)$. La notación que se utilizará es la siguiente:

$$\Pi(I, B_1,\ldots, B_r)$$

■

Definición: *[Selección]* Sea I=(RS,T) una instancia de relación, se definirá selección de I donde se cumple la condición C, como otra instancia $I_{SELEC}=(RS_{SELEC},T_{SELEC})$ donde $RS_{SELEC}=RS$ y T_{SELEC} incluido o igual a T, t_i e T_{SELEC} si y solo si satisface la condición C. La notación que se utilizará es la siguiente:

$$\sigma(I, Condición\ C)$$

■

Definición: *[Producto Cartesiano]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación, se definirá producto cartesiano entre I_1 e I_2 a otra instancia I_{PROD} donde $I_{PROD}=(RS_{PROD},T_{PROD})$, siendo $RS_{PROD}=\{RS_1,RS_2\}$ y T_{PROD} igual al producto cartesiano de conjuntos entre T_1 y T_2 . La notación que se utilizará es la
siguiente:

$$I_1 x\, I_2$$

Definición: *[Unión]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación, con la precondición de que RS_1 y RS_2 sean compatibles, se definirá unión entre I_1 e I_2 a otra instancia I_{UNION} donde $I_{UNION}=(RS_{UNION},T_{UNION})$, siendo $RS_{UNION}=RS_1=RS_2$ y T_{UNION} igual a la unión de conjuntos entre T_1 y T_2 . La notación que se utilizará es la siguiente:

$$I_1 \cup I_2$$

■

Definición: *[Diferencia]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación, con la precondición de que RS_1 y RS_2 sean compatibles, se definirá diferencia entre I_1 e I_2 a otra instancia I_{DIF} donde $I_{DIF}=(RS_{DIF},T_{DIF})$, siendo $RS_{DIF}=RS_1=RS_2$ y T_{DIF} igual a la diferencia de conjuntos entre T_1 y T_2
La notación que se utilizará es la siguiente:

$$I_1 - I_2$$

■

A continuación se darán como dos instancias del ejemplo del hotel de Guillermo Tell visto en el capítulo de Modelo Relacional de Base de Datos, la instancia H del esquema relación HABITACIONES y la instancia M del esquema relación MUCAMAS.

Instancia H

Número-habit	Número-Camas	Baño	Piso	Precio
201	3	True	2	250
202	2	True	2	150
203	2	True	2	150
301	3	True	3	250
302	2	True	3	150
303	2	True	3	150
304	1	False	3	100
305	1	False	3	100

Instancia M

Número-mucama	Número-habit
M1	201
M1	202
M2	203
M1	301
M1	302
M2	303
M2	304
M2	305

Ejemplo 1:

Necesitamos saber la cantidad de camas que pose cada cuarto, y deseamos visualizar los datos concernientes sólo a ellos. Esta información se puede representar con la siguiente tabla.

Número-habit	Número-camas
201	3
202	2
203	2
301	3
302	2
303	2
304	1
305	1

Se puede resolver esta consulta utilizando el operador proyección. Y diremos que hemos realizado una proyección de H sobre {Número-habit,Número-camas}

$$\prod(H, Numero - habit, Numero - camas)$$

■

Ejemplo 2:

Ahora supondremos que necesitamos saber los departamentos cuyo costo es inferior a 160 U$S. La respuesta a esta consulta es la siguiente tabla.

Número-habit	Número-Camas	Baño	Piso	Precio
202	2	True	2	150
203	2	True	2	150
302	2	True	3	150
303	2	True	3	150
304	1	False	3	100
305	1	False	3	100

Resolveremos esta consulta utilizando el operador *selección*. Y diremos que hemos realizado una selección de las HABITACIONES donde el Precio<160.

$$\sigma(HABIT, Precio < 160)$$

La selección de una instancia para alguna condición solo contiene aquellas tuplas que cumplen la condición. En el ejemplo hemos visto una posible condición, pero se pueden utilizar distintas expresiones lógicas en las que participan los atributos del esquema en cuestión.

■

Ejemplo 3:

Veremos dos operaciones utilizando otra instancia de HABITACIONES que denominaremos H1, que sería la siguiente:

Instancia H1

Número-habit	Número-Camas	Baño	Piso	Precio
205	3	True	2	300
207	2	True	2	250

Número-habit	Número-Camas	Baño	Piso	Precio
301	3	True	3	250
302	2	True	3	150

Y las siguientes son las operaciones antes mencionadas entre H y H1.

H U H1

Número-habit	Número-Camas	Baño	Piso	Precio
201	3	True	2	250
202	2	True	2	150
203	2	True	2	150
205	3	True	2	300
207	2	True	3	250
301	3	True	3	250
302	2	True	3	150
303	2	True	3	150
304	1	False	3	100
305	1	False	3	100

H - H1

Número-habit	Número-Camas	Baño	Piso	Precio
201	3	True	2	250
202	2	True	2	150
203	2	True	2	150
303	2	True	3	150
304	1	False	3	100
305	1	False	3	100

■

Ejemplo 4:

Otra operación de conjuntos que utilizaremos es el *producto cartesiano*

R

A	B
1	a
1	b

S

A	C
1	c
1	d
2	c

R x S

A	B	A	C
1	a	1	c
1	a	1	d
1	a	2	c
1	b	1	c
1	b	1	d
1	b	2	c

■

4.2.2 Operadores adicionales

Definición: *[Theta Join]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación y C una condición lógica, se definirá join entre I_1 e I_2 a otra instancia I_{JOIN} donde $I_{JOIN}=(RS_{JOIN},T_{JOIN})$, siendo $RS_{JOIN}=\{RS_1,RS_2\}$ y T_{JOIN} incluido o igual al producto cartesiano de conjuntos entre T1 y T2, t_i e T_{JOIN} si y solo si satisface a la condición θ . La notación que se utilizará es la siguiente:

$$(I_1 \bowtie I_2, \text{ condición } \theta)$$

Definición: *[Join Natural]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación, se definirá join natural entre I_1 e I_2 a otra instancia I_{JOIN} donde $I_{JOIN}=(RS_{JOIN},T_{JOIN})$, siendo $RS_{JOIN}=\{RS_1,RS_2\}$ y T_{JOIN} incluido o igual al producto cartesiano de conjuntos entre T1 y T2, t_i e T_{JOIN} si y solo si satisface a la condición de igualdad entre todos los atributos comunes, y cada atributo común aparece solo una vez. La notación que se utilizará es la siguiente:

$$I_1 * I_2$$

Definición: *[Intersección]* Sean $I_1=(RS_1,T_1)$ e $I_2=(RS_2,T_2)$ dos instancias de relación, con la precondición de que RS_1 y RS_2 sean compatibles, se definirá intersección entre I_1 e I_2 a otra instancia I_{INTER} donde $I_{INTER}=(RS_{INTER},T_{INTER})$, siendo $RS_{INTER}=RS_1=RS_2$ y T_{INTER} igual a la intersección de conjuntos entre T_1 y T_2 . La notación que se utilizará es la siguiente:

$$I_1 \cap I_2$$

■

Definición:*[Cociente]*. Tenemos dos esquemas R y S, con ATR_s incluido en ATR_r. El cociente r ÷ s, con r y s instancias de R y S respectivamente, es representado por una tabla en la cual están los atributos de R que no están en S. Si llamamos X al conjunto de atributos que no están en S e Y al resto de ellos, el cociente sería el conjunto de valores X tal que la tupla XY aparece en r para todos los valores Y que aparecen en S.

r

A	B	C	D
a	a	1	1
b	b	1	2
a	a	2	1
a	b	1	1
a	b	1	2
b	a	1	2
b	a	1	1

s

A	B
b	a
a	a
a	b

r ÷ s

C	D
1	1

■

Ejemplo 5:

Se desea saber toda la información correspondiente a las habitaciones e inclusive que mucama es la que trabaja en esa habitación. Esta consulta se representa con la tabla que sigue a continuación:

H.Número-habit	Número-Camas	Baño	Piso	Precio	Número-mucama	M.Número-habit
201	3	True	2	250	M1	201
202	2	True	2	150	M1	202
203	2	True	2	150	M2	203
301	3	True	3	250	M1	301
302	2	True	3	150	M1	302
303	2	True	3	150	M2	303
304	1	False	3	100	M2	304
305	1	False	3	100	M2	305

Esta consulta se resuelve utilizando el operador join de esta manera:

$$(H * M, H.Numero\text{-}habit = M.Numero\text{-}habit)$$

■

Para este caso, el join, se puede escribir en base a las operaciones fundamentales como:

$$H * M = \sigma(HxM, H.Numero\text{-}habit = M.Numero\text{-}habit)$$

■

Ejemplo 6:

El otro operador sobre conjuntos es la *intersección*, que vemos como se aplica utilizando el mismo método que utilizamos para diferencia y unión.

$$H \cap H1$$

Número-habit	Número-Camas	Baño	Piso	Precio
301	3	True	3	250
302	2	True	3	150

Para este caso, la intersección, se puede escribir en base a las operaciones fundamentales como:

$$H \cap H1 = H - (H - H1)$$

■

Definición: *[Renombrar]*. Sea I=(RS,T) una instancia de relación, se definirá como renombrar a otra instancia $I_{REN}=(RS_{REN},T_{REN})$ donde T_{REN} = T y $ATR_{REN}=f(ATR)$ siendo f una función biyectiva que asocia a un elemento de ATR con uno y solo un elemento de ATR_{REN}. La notación que se utilizará es la siguiente:

$$\mu(I, f)$$

■

Ejemplo 6:

Deseamos renombrar los atributos de M por NM y NH.

NM	NH
M1	201
M1	202
M2	203
M1	301
M1	302
M2	303
M2	304
M2	305

La consulta se resuelve con el operador *renombrar* de la siguiente manera:

$$\mu(M, f(Numero - mucama) \rightarrow NM\, f(Numero - habit) \rightarrow NH)$$

■

4.2.3 Sintaxis del Algebra Relacional

$ExpAlg = IdEsquema \mid$
$IdInstancia \mid$
$\Pi\ (ExpAlg; ConjAtrib) \mid$
$\sigma\ (ExpAlg; ValorLógico) \mid$
$\mu\ (ExpAlg; IdFunción) \mid$
$\mu\ (ExpAlg; NotFunción) \mid$
$ExpAlg \cup ExpAlg \mid$
$ExpAlg \cap ExpAlg \mid$
$ExpAlg - ExpAlg \mid$
$ExpAlg \div ExpAlg \mid$

$ConjAtrib = IdAtrib \mid$
$IdAtrib, ConjAtrib$

$NotFunción = IdAtrib \rightarrow IdAtrib \mid$
$IdAtrib \rightarrow IdAtrib, NotFunción$

$ValorLógico = ExpLog\ oplog\ ExpLog \mid$
$IdFunción(ConjAtrib)$

$ExpLog = Valor\ op\ Valor$

$Valor = Constante \mid IdAtrib$

$op = < \mid > \mid \leq \mid \geq \mid = \mid \neq$ _

$oplog = \vee \mid \wedge$ _ _

Los símbolos terminales de ésta gramática son: ConjAtrib que es un conjunto de atributos incluido a igual a ATR e IdAtrib son los identificadores de los atributos.

4.3 S.Q.L.

4.3.1 INTRODUCCION

SQL significa "Structured Query Language", el lenguaje SQL consiste en un conjunto de facilidades para definir, manipular y controlar los datos en una base de datos relacional.

Utilizando el mismo mecanismo que en algebra y en calculo describiremos como se utiliza el lenguaje SQL para luego profundizar en una explicación formal de este lenguaje. Nos referiremos siempre a la versión estándar del mismo haciendo en algunos casos referencia a otros dialectos no estandarizados.

En SQL estándar se utiliza la noción de ambiente (environment). Cada ambiente se constituye por una única base de datos (database). Una base de datos es el conjunto de todos los datos definidos por todos los esquemas en dicho ambiente. Cada esquema es un conjunto de definiciones de tablas, un conjunto de definiciones de vistas, y un conjunto de definiciones de privilegios.

Los datos en la base de datos son vistos por el usuario como una colección de tablas con nombre (named tables), cada una de las cuales tiene un nombre único en la base de datos. Cada tabla con nombre puede ser una tabla base (base table) o una vista (view). Una tabla base es una tabla "real" autónoma (es decir, no definida en términos de otras tablas). Una vista es una tabla "virtual": es una tabla con nombre, derivada de otras tablas. Las tablas base y las vistas se crean con los comandos CREATE TABLE y CREATE VIEW, respectivamente. El estándar no especifica formas de destruir tablas En otros dialectos SQL se dispone de comandos DROP TABLE y DROP VIEW para destruir tablas y vistas respectivamente.

Una tabla consiste en una fila de encabezados de columna, junto a cero o mas renglones de valores de datos. Para una tabla dada, la columna de encabezado especifica una o mas columnas con nombre con un tipo de datos asociado a dicha columna. En una misma tabla no pueden haber dos columnas con el mismo nombre. Cada renglón de datos contiene exactamente un valor por cada columna especificada en el renglón de encabezado. Todos los valores en una misma columna son de un mismo tipo, que es el tipo asociado a la

columna en la columna de encabezado. La cantidad de renglones de datos de una tabla se denomina cardinalidad. El numero de columnas de una tabla se denomina grado .

Los valores escalares son strings y numeros En otros dialectos, se dispone de otros tipos, por ejemplo "fecha". Un string es una secuencia de cero o mas caracteres.

El valor nulo (null value) es un valor especial que representa "valor desconocido".

4.3.2 SENTENCIAS DE MANIPULACION DE DATOS

Al comienzo del curso se comento que era el Lenguaje de Manipulación de Datos (Data Manipulation Language, DML), en este capítulo veremos las cuatro sentencias del lenguaje SQL, que componen el DML. Esas sentencias son: Select, Insert, Delete, Update.

4.3.2.1 Sentencia Select

En principio daremos la sintaxis de la sentencia **SELECT,** y luego con ejemplos iremos explicando de una forma adecuada su significado haciendo una correlación entre SQL y Algebra Relacional. La sintaxis de esta sentencia es la siguiente:

SELECT A_i op A_l,...,f(t.A_i),...,r_i.A_k
FROM ...,r_i,...,r_j t,...
[**WHERE** C_a]
[**GROUP BY** A_s,...,t.A_r,...,r_i.A_t]
[**HAVING** C_b]
[**ORDER BY** A_n,...,t.A_m,...,r_i.A_o]

Donde los $\mathbf{A_i}$ corresponden a nombre de atributos, los $\mathbf{r_i}$ a nombres de instancias (tablas), los $\mathbf{C_i}$ condiciones, **op** operadores aritméticos y **f** funciones que denotaremos *funciones agregadas*.

La sentencia **SELECT** se utiliza para consultar a la base de datos. Hay varias cláusulas que pueden emplearse en esta sentencia. Hay dos que son requeridas (obligatorias) **SELECT** y **FROM**. Si se utilizan las otras cláusulas, deben aparecer en el orden en que se muestran en la sintaxis.

A continuación iremos viendo ejemplos para ver como se utiliza la sentencia **SELECT**. Se utiliza la base de datos ACMEDB.

Ejemplo:

Listar nombre, apellido y teléfono de cada cliente.

```
SELECT ALL Nombre, Apellido, Telefono
FROM Clientes
;
```

■

Observaciones:

Para listar todas las columnas de una tabla, usar * junto al **SELECT**. Por ejemplo:

```
SELECT *
FROM Stock;
```

o también:

```
SELECT Stock.*
FROM Stock;
```

La palabra ALL se asume por defecto, por lo que en la práctica suele no usarse.

■

Ejemplo:

Listar los números de clientes que tienen órdenes pedidas. Los números que se repiten no deben aparecer en el resultado.

```
SELECT UNIQUE NrClient
FROM Ordenes
;
```

o también:

```
SELECT DISTINCT NrClient
FROM Ordenes
```

;

■

4.3.2.2 Cláusula ORDER BY

Se usa para que los resultados aparezcan ordenados. El orden (por defecto) es ASCendente. Para ordenar DESCendentemente se usa la palabra DESC a continuación de la columna correspondiente. Un valor NULL es tratado como menor que un valor no nulo. Las columnas mencionadas en el ORDER BY deben aparecer en la cláusula **SELECT**.

Ejemplo:

Listar los artículos del stock, ordenados por descripción y precio unitario, en forma ascendente.

```
SELECT *
FROM Stock
ORDER BY Descrip, PrUnidad
;
```

■

Ejemplo:

Listar los artículos en stock, ordenando en forma descendente por precio unitario.

```
SELECT *
FROM Stock
ORDER BY PrUnidad DESC
;
```

■

4.3.2.3 Cláusula WHERE

Se usa para especificar un criterio de búsqueda de datos y condiciones para vincular datos de diferentes tablas (join). Los *criterios de búsqueda* pueden ser:

- Exclusión de valores: **!= <>**
- Rango de valores: **BETWEEN**
- Subconjunto de valores: **IN**
- Valor nulo: **IS [NOT] NULL**
- Expresiones regulares en strings
- **WHERE** con exclusión de valores

Ejemplo:

Listar números de clientes, compañía y estado, de los clientes que no son del estado CA (California).

```
SELECT NrClient, Compania, Estado
FROM Clientes
WHERE Estado <> "CA"
;
```

o también:

```
SELECT NrClient, Compania, Estado
FROM Clientes
WHERE Estado != "CA"
;
```

■

■ **WHERE con búsqueda de un rango de valores**

Ejemplo:

Listar los números de stock, códigos de fabricantes, descripciones y precios unitarios, de los artículos en stock cuyos precios unitarios están entre $10.00 y $30.00 inclusive.

```
SELECT NrStock, CodFabr, Descrip, PrUnidad
FROM   Stock
WHERE  PrUnidad >= 10.00 AND
       PrUnidad <= 30.00
;
```

o también:

```
SELECT NrStock, CodFabr, Descrip, PrUnidad
FROM   Stock
WHERE  PrUnidad BETWEEN 10.00 AND 30.00
;
```

■

■ **WHERE con subconjunto de valores**

Ejemplo:

Listar los apellidos, ciudades, estados y teléfonos, de los clientes del estado CA (California) o del estado NV (Nevada). El resultado se desea ordenado por apellidos.

```
SELECT Apellido, Ciudad, Estado, Telefono
FROM Clientes
WHERE Estado = "CA" OR Estado = "NV"
ORDER BY Apellido
;
```

o también:

```
SELECT Apellido, Ciudad, Estado, Telefono
FROM Clientes
WHERE Estado IN ("CA", "NV")
ORDER BY Apellido
;
```

■

■ **WHERE con valor nulo (NULL)**

Ejemplo:

Listar los apellidos, ciudades y teléfonos, de los clientes del estado CA (California) que tienen valor desconocido en su segunda dirección. El resultado se desea ordenado por apellidos.

```
SELECT Apellido, Ciudad, Telefono
FROM Clientes
WHERE Estado = "CA" AND Direcc2 IS NULL
ORDER BY Apellido
;
```

■

Observaciones:

El valor NULL se considera "valor desconocido". El valor 0 y el valor "" no se consideran valores nulos y son distintos del valor NULL. El valor NULL es insertado cuando al agregar un renglón a una tabla, se dejan columnas sin datos.

■

■ **WHERE con expresiones regulares**

Una *expresión regular* es un patrón de búsqueda de strings, similar al uso de los wildcards que se utilizan en los sistemas operativos *DOS* (cláusula **LIKE**) y *UNIX* (cláusula **MATCHES**). Las expresiones regulares se especifican mediante las cláusulas **LIKE** y **MATCHES**.

[NOT] LIKE:

%	representa una secuencia de caracteres
_	representa un único caracter

[NOT] MATCHES:

*	representa una secuencia de caracteres
?	representa un único caracter
[c-c]	representa rangos de caracteres
[cc]	representa conjuntos de caracteres

Ejemplo:

Listar artículos en stock cuya descripción empiece con los caracteres "BOLSO".

```
SELECT *
FROM Stock
WHERE Descrip LIKE "BOLSO%"
;
```

o también:

```
SELECT *
FROM Stock
WHERE Descrip MATCHES "BOLSO*"
;
```

■

Ejemplo:

Listar datos de los fabricantes cuyos códigos tengan una letra "R" en la segunda letra del código.

```
SELECT *
FROM Fabricnt
WHERE CodFabr LIKE "_R_"
;
```

o también:

```
SELECT *
FROM Fabricnt
WHERE CodFabr MATCHES "?R?"
;
```

■

Ejemplo:

Listar datos de los fabricantes cuyos nombres empiecen con letras desde la "A" hasta la "N".

```
SELECT *
FROM Fabricnt
WHERE NomFabr MATCHES "[A-N]*"
;
```

■

Ejemplo:

Listar datos de los fabricantes cuyos nombres empiecen con la letra "A" o con la letra "N".

```
SELECT *
FROM Fabricnt
WHERE NomFabr MATCHES "[AN]*"
;
```

■

Observaciones: Estos tipos de expresiones regulares no pueden realizarse con la cláusula **LIKE**.

■

4.3.2.4 Expresiones aritméticas

Una *expresión aritmética* puede usarse en la cláusula **SELECT** o en la cláusula **WHERE**.

Los *operadores aritméticos* son:

*	multiplicación
/	división
+	adición
-	sustracción

Ejemplo:

Listar los datos de los artículos en stock, junto con el precio unitario multiplicado por 1.05.

```
SELECT Stock.*, PrUnidad*1.05
FROM Stock
;
```

■

Ejemplo:

```
SELECT    NrOrden, Artics.NrStock, Artics.CodFabr, Cantidad,
          PrUnidad, Cantidad*PrUnidad, Precio

FROM Artics, Stock
WHERE     Artics.NrStock = Stock.NrStock AND
          Artics.CodFabr = Stock.CodFabr
ORDER BY  NrOrden
;
```

■

4.3.2.5 Funciones agregadas

Las *funciones agregadas* se usan para realizar conteos, sumas, promedios, hallar máximos y mínimos, en columnas con datos numéricos.

Las funciones disponibles son:

- **COUNT(*)**
- **COUNT(DISTINCT columna)**

- **SUM(x)**
- **SUM(DISTINCT columna)**

- **AVG(x)**
- **AVG(DISTINCT columna)**

- **MAX(x)**
- **MIN(x)**

donde *x* puede ser una columna o una expresión aritmética.

Si el conjunto que está en la cláusula **SELECT** incluye alguna función agregada, no se pueden incluír otras columnas a menos que se use la cláusula **GROUP BY**.

- **Claúsula COUNT**

COUNT cuenta todos los renglones que satisfacen la condición de la cláusula **WHERE**, aún cuando los renglones contengan valores **NULL** o no. Si se desean contar valores diferentes de alguna columna se usa la cláusula **COUNT(DISTINCT columna)**. Si todos los valores de la columna son **NULL**, da 0. Si hay valores **NULL** y otros no lo son, no se consideran a los valores **NULL**. **COUNT(*)** cuenta todos los renglones, aún cuando el valor de cada columna en el renglón sea **NULL**.

Ejemplo:

Contar todas las órdenes fechadas en abril y mayo de 1986.

```
SELECT COUNT(*)
FROM Ordenes
WHERE FechaOrd BETWEEN "01/04/86" AND "31/05/86"
;
```

■

Ejemplo:

Contar los números de stock diferentes que hay en la tabla de stock.

```
SELECT COUNT(*)
FROM Stock
;
```

■

■ **Cláusulas MAX y MIN**

Se usan para hallar valores máximos y mínimos en columnas aritméticas de los renglones seleccionados con la cláusula **WHERE**. Si una columna contiene valores NULL y valores no NULL, MAX y MIN ignoran los valores NULL. Una consulta retornará el valor apropiado basado en el resto de los renglones. Si todos los valores son NULL, MIN y MAX retornarán NULL.

Ejemplo:

Hallar el máximo y el mínimo precio unitario de los artículos en stock cuyo código de fabricante es HSK.

```
SELECT MAX(PrUnidad), MIN(PrUnidad)
FROM Stock
WHERE CodFabr = "HSK"
;
```

■

4.3.2.6 Cláusula GROUP BY

Un *grupo* es un conjunto de renglones que tienen el mismo valor en una determinada columna. **GROUP BY** se usa para producir un único renglón de resultados, por cada grupo.

Ejemplo:

Agrupar los artículos de la tabla Artics. Por cada grupo listar el número de orden, la suma de los precios de los artículos del grupo y su valor promedio.

```
SELECT NrOrden, SUM(Precio), AVG(Precio)
FROM Artics
GROUP BY NrOrden
;
```

■

Observaciones:

Cuando se agrupan datos, solamente las columnas incluídas en la cláusula **GROUP BY** o en funciones agregadas, pueden aparecer en el conjunto de la cláusula **SELECT**. Sin embargo, una columna en la cláusula **GROUP BY** no necesita estar en el conjunto de la cláusula **SELECT**.

■

Ejemplo:

```
SELECT NrOrden, SUM(PrUnidad)
FROM Artics
GROUP BY NrOrden, CodFabr
;
```

■

4.3.2.7 Cláusula HAVING

La cláusula **HAVING** solamente puede especificarse cuando hay una cláusula **GROUP BY**. Permite aplicar una condición para seleccionar los grupos que van a aparecer en el resultado.

Ejemplo:

Listar en forma agrupada los números de órdenes que aparecen en Artics, indicando la suma de los precios de cada orden. Sólamente listar las órdenes cuys suma supere los $ 150.

```
SELECT NrOrden, SUM(Precio)
FROM Artics
GROUP BY NrOrden
HAVING SUM(Precio) > 150
;
```

■

4.3.3 Relación entre SELECT y Algebra Relacional

Se verá la relación existente entre la sentencia **SELECT** y el algebra relacional, lo cual nos permitirá profundisar un poco más en esta sentencia.

4.3.3.1 Selección

En SQL se implementa mediante la cláusula **WHERE**. La tabla resultante tiene las mismas columnas que la tabla original.

Ejemplo:

Obtener los datos de los clientes que viven en la ciudad Palo Alto.

```
SELECT *
FROM Clientes
WHERE Ciudad = "PALO ALTO"
;
```

■

4.3.3.2 Proyección

En SQL se implementa colocando en el **SELECT** el conjunto de atributos sobre el cual se desea proyectar.

Ejemplo:

Obtener los datos de ciudad, estado y código postal de todos los clientes.

```
SELECT UNIQUE Ciudad, Estado, CodPost
FROM Clientes
;
```

o también:

```
SELECT DISTINCT Ciudad, Estado, CodPost
FROM Clientes
;
```

■

4.3.3.3 Selección y Proyección

Las consultas típicamente involucran a las operaciones de selección y de proyección en una única sentencia.

Ejemplo:

Obtener los datos de ciudad, estado y código postal de los clientes del estado identificado como CA.

```
SELECT UNIQUE Ciudad, Estado, CodPost
FROM Clientes
WHERE Estado = "CA"
;
```

■

4.3.3.4 Producto cartesiano

En SQL se realiza solamente con colocar más de una tabla en el **FROM** y no tener **WHERE**.

Ejemplo:

```
SELECT *
FROM Fabricnt, Stock
;
```

■

Observaciones:

Analice detalladamente el resultado de la consulta anterior. Han aparecido datos incorrectos debido a que no todas las combinaciones posibles entre los datos de los fabricantes y los datos de stock, son combinaciones válidas.

■

4.3.3.5 Join

Se usa cuando dos tablas tienen una o más columnas en común y pueden vincularse para crear una nueva tabla de resultados, la tabla resultado se compone de todas las columnas y renglones de las tablas que se "joinean". El vínculo entre las tablas se menciona en la cláusula WHERE. Un JOIN produce una tabla resultado que contiene columnas idénticas. Se obtiene cuando la tabla resultado muestra a todas las columnas de las dos tablas que fueron joineadas. Se usan los nombres de las tablas y alias para distinguir columnas de una tabla y otra, cuando las columnas tienen mismo nombre.

Ejemplo:

Obtener los datos de los fabricantes y los artículos en stock que fabrican.

```
SELECT *
FROM Fabricnt, Stock
WHERE Fabricnt.CodFabr = Stock.CodFabr
;
```

o también:

```
SELECT *
FROM Fabricnt F, Stock S
WHERE F.CodFabr = S.CodFabr
;
```

■

4.3.4 SubConsultas

Las sentencias **SELECT** pueden anidarse en una cláusula **WHERE** para formar una subconsulta. A la sentencia **SELECT** más exterior en la consulta se la llama "SELECT externa". A las interiores se las llama "anidadadas".

Ejemplo:

Obtener los datos de clientes que viven en la misma ciudad en la que vive el cliente cuyo apellido es Higgins.

```
SELECT *
FROM Clientes
WHERE Ciudad =
        (SELECT Ciudad
        FROM Clientes
        WHERE Apellido = "HIGGINS")
;
```

■

Una SELECT anidada puede usarse para:

Comparar una expresión con el resultado de otra sentencia **SELECT**, determinar si una expresión está incluída en los resultados de otra sentencia **SELECT** y determinar si hay renglones en el resultado de otra sentencia **SELECT**.

¿Cómo usar subconsultas?

Una subconsulta puede dar un único valor, ningún valor, o un conjunto de valores. El conjunto de la cláusula **SELECT** de las subconsultas que usan las palabras **IN, ALL** o **ANY** deben contener solamente una columna o expresión. Si se utiliza la palabra **EXISTS** deben contener un renglón entero. La subconsulta no debe tener cláusula **ORDER BY**.

4.3.4.1 Condiciones de subconsultas

■ expr oprel **{ ALL|ANY|SOME }** (sentencia **SELECT**)

■ expr **[NOT] IN** (sentencia **SELECT**)

■ **[NOT] EXISTS** (sentencia **SELECT**)

oprel: **= < > <= >= != <>**

■ **ALL**

Ejemplo:

Hallar las órdenes que se refieren solamente a equipamiento suministrado por el fabricante de código ANZ.

```
SELECT Artics.*
FROM Artics A
WHERE "ANZ" = ALL
        (SELECT CodFabr
        FROM Artics B
        WHERE B.NrOrden = A.NrOrden)
;
```

■

■ **ANY**

Ejemplo:

Hallar las órdenes donde alguno de los equipamientos es suministrado por el fabricante de código ANZ.

```
SELECT Artics.*
FROM Artics A
WHERE "ANZ" = ANY
```

```
        (SELECT CodFabr
        FROM Artics B
        WHERE B.NrOrden = A.NrOrden)
;
```

■

■ **IN**

Ejemplo:

Obtener los nombres de los fabricantes que suministran equipamiento de tenis.

```
SELECT  NomFabr
FROM Fabricnt
WHERE CodFabr IN
        (SELECT UNIQUE CodFabr
        FROM Stock
        WHERE Descrip MATCHES "*TENIS*")
;
```

■

■ **EXISTS**

La expresión: **EXISTS (SELECT * FROM ...)**

es verdadera si y sólo si el resultado del **SELECT** tiene al menos un renglón.

Ejemplo:

Hallar los nombres de los fabricantes que suministran equipamiento de tenis.

```
SELECT NomFabr
FROM Fabricnt
WHERE EXISTS
        (SELECT *
```

```
        FROM Stock
        WHERE Fabricnt.CodFabr = Stock.CodFabr AND
        Descrip MATCHES "*TENIS*")
;
```

■

4.3.5 Operaciones de conjuntos

Las operaciones **UNION, INTERSECCION** y **DIFERENCIA** del Algebra Relacional pueden realizarse en SQL. La operación **UNION** está implementada. La **INTERSECCION** y **DIFERENCIA** no están implementadas pero pueden realizarse usando **SELECT** con la cláusula **IN**.

4.3.5.1 Unión

La unión de dos tablas produce una tabla que contiene a los renglones de ambas. Se realiza usando la cláusula **UNION** entre dos **SELECT**s. Los renglones duplicados son excluídos del resultado. Para mantenerlos se usa **UNION ALL**. Las tablas deben ser compatibles, es decir:

- Deben tener la misma cantidad de columnas.
- Los tipos de datos deben ser compatibles para las columnas correspondientes entre sí. Las columnas de tipo CHAR deben tener el mismo ancho.

Ejemplo:

Listar las órdenes activas y las órdenes cerradas del cliente número 101.

```
SELECT *
FROM Ordenes
WHERE NrClient = 101
UNION
SELECT *
FROM OrdenesC
WHERE NrClient = 101
;
```

■

4.3.5.2 Intersección

La intersección de dos tablas produce una tabla con los renglones que son comunes a ambas tablas. La intersección se realiza usando **SELECT** con **IN**.

Ejemplo:

Listar números de clientes y números de orden, de aquellos clientes que tienen órdenes tales que las mismas están activas y a la vez, cerradas (una inconsistencia en los datos!).

```
SELECT NrClient, NrOrden
FROM Ordenes
WHERE NrOrden IN
        (SELECT NrOrden
        FROM OrdenesC)
;
```

■

4.3.5.3 Diferencia

La diferencia de dos tablas A y B (A - B) produce una tabla con los renglones de A que no están en B. La diferencia se realiza con **SELECT** y **NOT IN**.

Ejemplo:

Hacer la diferencia entre las tablas de órdenes y de órdenes cerradas.

```
SELECT NrClient, NrOrden
FROM Ordenes
WHERE NrOrden NOT IN
        (SELECT NrOrden
        FROM OrdenesC)
;
```

■

Bibliografía.

[Korth 1992]. Korth, H.F., Silberschatz, A. 1992. *Fundamentos de Bases de Datos,* McGraw-Hill.

[Paredaens 1989]. Paredaens, J., De Bra, P., Gyssens, M., Van Gucht, D. 1989. *The Structure of the Relational Data Model,* Springer-Verlag.

[Ullman 1988]. Ullman, J.D. 1988. *Principles of Database and Knowledge-Base Systems,* Computer Science Press, Inc.

[Date 1989]. Date, C. J., 1989. *A Guide to the SQL Standard,* Addison-Wesley Publishing Company.

■