SQL

1. INTRODUCCION.

El modelo relacional fue propuesto por E.F.Codd en un artículo ya famoso ("A relational model of data for large shared data banks" - Comn.ACM 13,6 – June 1970) . Desde ese momento se instituyeron varios proyectos de investigación con el propósito de construir sistemas de gestión de bases de datos relacionales.
Entre estos proyectos podemos mencionar:

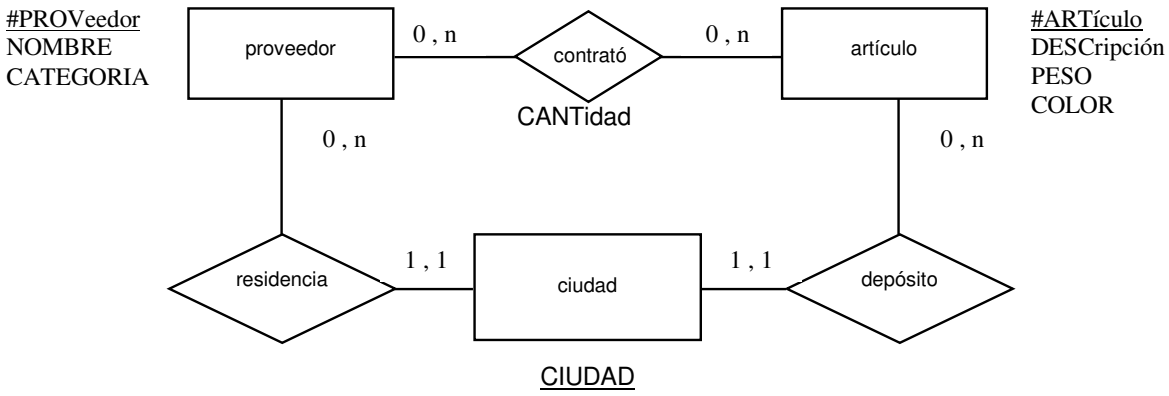
- Sistema R del IBM San José Research Laboratory
- Ingres de la Universidad de California en Berkeley
- Query-by-example del IBM T.J.Watson Research Center
- PRTV (Peterlee Relational Test Vehicle) del IBM Scientific Center en Peterlee, Inglaterra.

El lenguaje SQL se introdujo como lenguaje de consulta del Sistema R. Posteriormente, varios sistemas comerciales lo adoptaron como lenguaje para sus bases de datos. Actualmente existe una propuesta, bajo los auspicios del American National Standards Institute (ANSI) para crear un lenguaje SQL estándar.
El nombre SQL está formado por las iniciales de Structured Query Language (Lenguaje de consultas estructuradas).

Los lenguajes formales que vimos hasta ahora (álgebra relacional, cálculo relacional de tuplas y cálculo relacional de dominios) permiten representar las consultas en forma concisa. Sin embargo los sistemas comerciales de bases de datos requieren un lenguaje de consulta "más amable con el usuario". Aunque se les da el nombre de lenguajes de consulta, los lenguajes como el SQL realizan muchas otras funciones además de consultar bases de datos. Por ejemplo, tienen capacidad para definir estructuras de datos, para modificar los datos en la base de datos, para especificar las autorizaciones de acceso, para especificar las restricciones de integridad, etc.

Nuestro objetivo no es dar una guía completa del lenguaje SQL, más bien queremos presentar sus conceptos y construcciones fundamentales. Cada implementación de este lenguaje puede diferir en sus detalles, contar únicamente con un subconjunto del lenguaje completo, y/o contar con funciones adicionales propias de esa versión.
Nosotros trabajaremos con la sintaxis SQL detallada en el punto 9.

Todos los ejemplos de este apunte están basados en el siguiente modelo E-R:



A partir de este MER se han definido las siguientes relaciones:

proveedores	todos los proveedores de la empresa. Por cada proveedor nos interesa su número identificatorio (#PROV), su nombre (NOMBRE), su categoría (CATEGORIA) y la ciudad donde reside (CIUDAD).
artículos	todos los artículos con que la empresa trabaja. Por cada artículo nos interesa su número identificatorio (#ART), su descripción (DESC), su peso (PESO), su color (COLOR) y la ciudad donde está depositado (CIUDAD).
contratos	cantidad (CANT) que un proveedor (#PROV) suministra de un artículo (#ART).

2. TABLAS.

EL SQL se basa en el modelo relacional: implementa las relaciones como tablas, los atributos como columnas y las tuplas como filas de las tablas.
Para definir una tabla en SQL se usa la sentencia CREATE TABLE:

CREATE TABLE tabla (columna tipo [NOT NULL] , ...) ;

Por ejemplo, para crear una tabla que implemente la relación proveedores podemos definir:

```
CREATE TABLE proveedores (#PROV      INTEGER NOT NULL,
                          NOMBRE     CHAR (10),
                          CATEGORIA  INTEGER,
                          CIUDAD     CHAR (15) NOT NULL);
```

NOT NULL indica que, al insertar una fila en la tabla, el valor de esa columna no puede ser omitido.
En este caso, cada proveedor debe tener un número identificatorio y una ciudad de residencia.

Nótese que el SQL no implementa el concepto de dominio del MODELO RELACIONAL, tan solo permite definir el tipo (alfanumérico, entero, etc.) y la longitud de las columnas.

```
CREATE TABLE artículos (#ART      CHAR (2) NOT NULL,
                        DESC      CHAR (25),
                        PESO      INTEGER,
                        COLOR     CHAR (8),
                        CIUDAD    CHAR (15) NOT NULL);
```

```
CREATE TABLE contratos (#PROV     INTEGER NOT NULL,
                        #ART      CHAR (2) NOT NULL,
                        CANT      INTEGER NOT NULL);
```

Cuando se desea eliminar una tabla se usa la sentencia DROP TABLE.
Para modificar la estructura de una tabla, ya sea modificando la especificación de tipo y/o la longitud de una columna o agregando una columna, se usa la sentencia ALTER TABLE.

3. MANIPULACION DE DATOS.

Los cuatro operadores básicos son:

SELECT (consulta las filas de una o más tablas)
UPDATE (modificación de filas de una tabla)
INSERT (agregado de filas a una tabla)
DELETE (eliminación de filas de una tabla)

La sintaxis completa de estas sentencias se especifica en el punto 9.

SELECT

La estructura básica de la sentencia de consulta SELECT es:

SELECT A1 , A2 , A3 , ……., An
FROM r1 , r2 , r3 , ……., rm
[WHERE p]

donde los A1 , A2 , ... , An son atributos (columnas) , las r1 , r2 , ..., rm son relaciones (tablas) y p es un predicado.

Una consulta SELECT ..... es equivalente a la expresión en álgebra relacional:

$$\pi_{A1 , A2 , A3 ,....., An} \ (\sigma_p \ (r1 \times r2 \times r3 \times .....\times rm))$$

La cláusula SELECT corresponde a la operación de proyección del álgebra relacional. Sirve para listar todos los atributos (columnas) que se desean en el resultado de una consulta.

La cláusula FROM es una lista de relaciones (tablas) que se van a examinar durante la ejecución de la expresión.

La cláusula WHERE (que es opcional) corresponde al predicado de la operación de selección del álgebra relacional. Incluye atributos (columnas) de las relaciones (tablas) que aparecen en la cláusula FROM.

Para seguir los ejemplos consideraremos las siguientes tablas:

proveedores

#PROV	NOMBRE	CATEGORIA	CIUDAD
1	Jaime	20	Córdoba
6	Celia	20	Córdoba
3	ACME	30	Rosario
4	González	10	Rosario
5	XXX	10	Trelew

artículos

#ART	DESC	PESO	COLOR	CIUDAD
10	A	5	rojo	Jujuy
20	F	7	rojo	Rosario
30	A	5	azul	Córdoba
40	Z	3	verde	Córdoba
50	C	1	verde	Rosario
60	W	9	rojo	Córdoba

contratos

#PROV	#ART	CANT
1	10	300
6	20	100
1	60	200
1	50	400
3	60	200
1	40	100
4	10	100
4	60	300
4	30	400
1	30	200
3	20	200
4	20	300
3	10	150

1) Listar el número de todos los artículos.

1.a) SELECT #ART FROM artículos;

#ART
10
20
30
40
50
60

1.b) SELECT #ART FROM contratos; (Ver aclaración ejercicio 15.b)

#ART
10
20
60
50
60
40
10
60
30
30
20
20
10

1.c) SELECT DISTINCT #ART FROM contratos; (Ver aclaración ejercicio 15.b)

#ART
10
20
60
50
40
30

2) Listar el número, nombre, categoría y ciudad de todos los proveedores.

2.a) SELECT #PROV , NOMBRE , CATEGORIA , CIUDAD FROM proveedores;

2.b) SELECT * FROM proveedores;

#PROV	NOMBRE	CATEGORIA	CIUDAD
1	Jaime	20	Córdoba
6	Celia	20	Córdoba
3	ACME	30	Rosario
4	González	10	Rosario
5	XXX	10	Trelew

3) Obtener el número y la categoría de los proveedores residentes en Rosario.

```
SELECT  #PROV , CATEGORIA  FROM proveedores
WHERE CIUDAD = “Rosario”;
```

#PROV	CATEGORIA
3	30
4	10

4) Obtener el número y la categoría de los proveedores residentes en Rosario, ordenados ascendentemente por categoría.

4.a)
```
SELECT  #PROV , CATEGORIA  FROM proveedores
WHERE CIUDAD = “Rosario”
ORDER BY CATEGORIA ASC;
```

4.b)
```
SELECT  #PROV , CATEGORIA  FROM proveedores
WHERE CIUDAD = “Rosario”
ORDER BY 2 ASC;
```

#PROV	CATEGORIA
4	10
3	30

Los atributos por los que se quiere ordenar la salida deben aparecer en la tabla resultado, es decir, esos atributos deben ser especificados en la lista del SELECT.

¡¡¡¡ OJO !!!! NO ES VALIDO

```
SELECT  #PROV  FROM proveedores
WHERE CIUDAD = “Rosario”
ORDER BY CATEGORIA ASC;
```

5) Listar el número de proveedor de aquellos proveedores de Rosario cuya categoría sea mayor a 20.

```
SELECT  #PROV  FROM proveedores
WHERE CIUDAD = “Rosario” AND CATEGORIA > 20;
```

#PROV
3

6) Listar el número de los proveedores cuya categoría no sea 20.

```
SELECT  #PROV  FROM proveedores
WHERE CATEGORIA <> 20;
```

#PROV
3
4
5

7) Listar las cantidades (expresadas en decenas) del artículo 20 provistas por los distintos proveedores.

```
SELECT  #PROV , CANT / 10  FROM contratos
WHERE #ART =  “20”;
```

#PROV	
6	10
3	20
4	30

8) Listar los datos de los proveedores que suministran el artículo 20.

```
SELECT  *  FROM proveedores
WHERE #PROV  IN  ( SELECT  #PROV  FROM contratos WHERE #ART = “ 20” );
```

Podemos suponer que esta sentencia se resuelve así: primero el select interno obtendría una tabla con los proveedores que suministran el artículo 20.

#PROV
6
3
4

Luego se recorrería la tabla proveedores y por cada #PROV se verificaría si pertenece a la tabla recién calculada.

#PROV	NOMBRE	CATEGORIA	CIUDAD
6	Celia	20	Córdoba
3	ACME	30	Rosario
4	González	10	Rosario

9) Listar el número de los proveedores que no suministran el artículo 20.

¡¡¡¡ OJO !!!!

```
SELECT  DISTINCT #PROV  FROM contratos
WHERE #ART <> “20”;
```

no da el resultado buscado.

#PROV
1
3
4

Por ejemplo, el proveedor 3 aparece en el resultado y sin embargo suministra el artículo 20. ¿Por qué? Porque el SELECT codificado selecciona aquellas tuplas de la tabla contratos donde #ART sea distinto de 20.

9.a)
```
SELECT  #PROV  FROM proveedores
WHERE  #PROV NOT IN ( SELECT  #PROV  FROM contratos WHERE #ART = “20” );
```

Podemos suponer que se resuelve así:
Primero se resuelve el select interior para obtener una relación con los proveedores que suministran el articulo 20.

#PROV
6
3
4

Luego se recorre la tabla proveedores y por cada #PROV se verifica que no pertenezca a la tabla recién calculada.

#PROV
1
5

9.b)
```
SELECT  #PROV  FROM proveedores
MINUS
SELECT  #PROV  FROM contratos WHERE #ART = “20”;
```

Hacemos la diferencia entre una tabla que contiene el #PROV de todos los proveedores y otra tabla con el #PROV de los proveedores que suministran el artículo 20.

10) Listar el número de los proveedores con la misma categoría que el proveedor 6.

10.a)
```
SELECT  #PROV  FROM proveedores
WHERE CATEGORIA = ( SELECT  CATEGORIA  FROM proveedores
                            WHERE #PROV = 6 )
        AND #PROV <> 6;
```

#PROV
1

Nótese el resultado de este SELECT :

```
SELECT  #PROV  FROM proveedores
WHERE CATEGORIA = ( SELECT  CATEGORIA  FROM proveedores
                            WHERE #PROV = 6 );
```

#PROV
1
6

10.b) La sintaxis del WHERE es : WHERE columna θ subselect

donde θ puede ser > < >= <= = <>

Para obtener el número de los proveedores con mayor categoría que el proveedor 6.

```
SELECT  #PROV  FROM proveedores
WHERE CATEGORIA > ( SELECT  CATEGORIA  FROM proveedores
                            WHERE #PROV = 6 );
```

#PROV
3

11) Listar el número de proveedor , número de artículo y ciudad para aquellos proveedores y artículos que residen en la misma ciudad.

En Álgebra relacional haríamos el join natural de las relaciones proveedores y artículos. El SQL no provee una implementación directa de los operadores JOIN ni JOIN NATURAL. Aplicando la definición del join natural en función de los operadores básicos del álgebra,

$$\pi_{\#PROV\ ,\ \#ART\ ,\ proveedores.CIUDAD}\ (\ \sigma_p\ (\ proveedores\ x\ artículos\ )\ )$$

donde p : proveedores.CIUDAD = artículos.CIUDAD

podemos expresar esta consulta en SQL como :

```
SELECT  #PROV , #ART , artículos.CIUDAD  FROM proveedores , artículos
WHERE proveedores.CIUDAD = artículos.CIUDAD;
```

Podemos usar alias del nombre de las tablas, para facilitarnos la escritura de la consulta :

```
SELECT  #PROV , #ART , a.CIUDAD  FROM proveedores p , artículos a
WHERE p.CIUDAD = a.CIUDAD;
```

Veamos como se resolvería esta consulta :

	#PROV	NOMBRE	CATEGORIA	p.CIUDAD	#ART	DESC	PESO	COLOR	a.CIUDAD
	1	Jaime	20	Córdoba	10	A	5	rojo	Jujuy
	6	Celia	20	Córdoba	10	A	5	rojo	Jujuy
	3	ACME	30	Rosario	10	A	5	rojo	Jujuy
	4	González	10	Rosario	10	A	5	rojo	Jujuy
	5	XXX	10	Trelew	10	A	5	rojo	Jujuy
	1	Jaime	20	Córdoba	20	F	7	rojo	Rosario
	6	Celia	20	Córdoba	20	F	7	rojo	Rosario
*	3	ACME	30	Rosario	20	F	7	rojo	Rosario
*	4	González	10	Rosario	20	F	7	rojo	Rosario
	5	XXX	10	Trelew	20	F	7	rojo	Rosario
*	1	Jaime	20	Córdoba	30	A	5	azul	Córdoba
*	6	Celia	20	Córdoba	30	A	5	azul	Córdoba
	3	ACME	30	Rosario	30	A	5	azul	Córdoba
	4	González	10	Rosario	30	A	5	azul	Córdoba
	5	XXX	10	Trelew	30	A	5	azul	Córdoba
*	1	Jaime	20	Córdoba	40	Z	3	verde	Córdoba
*	6	Celia	20	Córdoba	40	Z	3	verde	Córdoba
	3	ACME	30	Rosario	40	Z	3	verde	Córdoba
	4	González	10	Rosario	40	Z	3	verde	Córdoba
	5	XXX	10	Trelew	40	Z	3	verde	Córdoba
	1	Jaime	20	Córdoba	50	C	1	verde	Rosario
	6	Celia	20	Córdoba	50	C	1	verde	Rosario
*	3	ACME	30	Rosario	50	C	1	verde	Rosario
*	4	González	10	Rosario	50	C	1	verde	Rosario
	5	XXX	10	Trelew	50	C	1	verde	Rosario
*	1	Jaime	20	Córdoba	60	W	9	rojo	Córdoba
*	6	Celia	20	Córdoba	60	W	9	rojo	Córdoba
	3	ACME	30	Rosario	60	W	9	rojo	Córdoba
	4	González	10	Rosario	60	W	9	rojo	Córdoba
	5	XXX	10	Trelew	60	W	9	rojo	Córdoba

Del producto cartesiano entre las tablas proveedores y artículos, el sistema seleccionaría aquellas filas que satisfacen la condición p, es decir, que tienen el mismo valor en los atributos CIUDAD. (Filas marcadas con un asterisco)

#PROV	NOMBRE	CATEGORIA	p.CIUDAD	#ART	DESC	PESO	COLOR	a.CIUDAD
3	ACME	30	Rosario	20	F	7	rojo	Rosario
4	González	10	Rosario	20	F	7	rojo	Rosario
1	Jaime	20	Córdoba	30	A	5	azul	Córdoba
6	Celia	20	Córdoba	30	A	5	azul	Córdoba
1	Jaime	20	Córdoba	40	Z	3	verde	Córdoba
6	Celia	20	Córdoba	40	Z	3	verde	Córdoba
3	ACME	30	Rosario	50	C	1	verde	Rosario
4	González	10	Rosario	50	C	1	verde	Rosario
1	Jaime	20	Córdoba	60	W	9	rojo	Córdoba
6	Celia	20	Córdoba	60	W	9	rojo	Córdoba

A continuación, el sistema proyectaría sobre las columnas especificadas en la cláusula SELECT.

#PROV	#ART	a.CIUDAD
3	20	Rosario
4	20	Rosario
1	30	Córdoba
6	30	Córdoba
1	40	Córdoba
6	40	Córdoba
3	50	Rosario
4	50	Rosario
1	60	Córdoba
6	60	Córdoba

12) Se desea una lista con nro. de proveedor, número de artículo y nombre de la ciudad de aquellos proveedores que suministran artículos que se depositan en la misma ciudad donde ellos residen.

En este caso se requiere realizar un join natural entre tres relaciones:

SELECT p.#PROV , a.#ART , a.CIUDAD FROM proveedores p , artículos a , contratos c
WHERE p.#PROV = c.#PROV AND a.#ART = c.#ART AND p.CIUDAD = a.CIUDAD;

p.#PROV	a.#ART	a.CIUDAD
1	60	Córdoba
1	40	Córdoba
1	30	Córdoba
3	20	Rosario
4	20	Rosario

13) Listar el número de aquellos artículos suministrados por proveedores residentes en Córdoba.

13.a) SELECT DISTINCT #ART FROM contratos c , proveedores p
WHERE c.#PROV = p.#PROV AND CIUDAD = “Córdoba”;

13.b) SELECT DISTINCT #ART FROM contratos
WHERE #PROV IN (SELECT #PROV FROM proveedores
WHERE CIUDAD = “Córdoba”);

#ART
10
20
60
50
40
30

14) Listar la descripción de los artículos depositados en Córdoba que son suministrados por proveedores residentes en Córdoba.

SELECT DESC FROM artículos
WHERE CIUDAD = “Córdoba”
AND < #ART , CIUDAD > IN
(SELECT #ART , CIUDAD FROM proveedores p , contratos c
WHERE c.#PROV = p.#PROV);

DESC
A
Z
W

15) Obtener el número de aquellos artículos cuyo peso esté entre 4 y 8, o sean provistos por el proveedor 3.

15.a) SELECT #ART FROM artículos WHERE PESO BETWEEN (4 AND 8)
UNION
SELECT #ART FROM contratos WHERE #PROV = 3;

Nótese que la operación UNION elimina duplicados.

15.b) SELECT DISTINCT c.#ART FROM artículos a , contratos c
WHERE a.#ART = c.#ART AND (#PROV = 3 OR (PESO BETWEEN (4 AND 8)));

Nótese que esta solución sólo es válida si todos los artículos cuyo peso esté entre 4 y 8 (en la tabla artículos) son provistos en algún contrato (es decir, figuran en la tabla contratos).

15.c)
```
SELECT  #ART  FROM artículos
    WHERE ( PESO >= 4 AND PESO <= 8 )
            OR #ART IN ( SELECT  #ART  FROM contratos WHERE #PROV = 3 );
```

#ART
10
20
30
60

16) Obtener la descripción de los artículos que se depositan en la misma ciudad donde está el depósito del artículo 50.

16.a)
```
SELECT  a2.DESC  FROM artículos a1 , artículos a2
    WHERE a1.#ART = “50” AND a1.CIUDAD = a2.CIUDAD AND a2.#ART <> “50”;
```

16.b)
```
SELECT  DESC  FROM artículos
    WHERE CIUDAD = ( SELECT  CIUDAD  FROM  artículos
                        WHERE #ART = “50”  )
          AND #ART <> “50”;
```

DESC
F

17) Obtener los detalles de los proveedores que suministran los artículos 10 o 60.

```
SELECT  *  FROM proveedores
WHERE #PROV IN ( SELECT  #PROV  FROM contratos
                    WHERE #ART IN ( “10” , “60” ) );
```

#PROV	NOMBRE	CATEGORIA	CIUDAD
1	Jaime	20	Córdoba
3	ACME	30	Rosario
4	González	10	Rosario

18) Obtener los detalles de los proveedores que suministran los artículos 10 y 60.

```
SELECT  *  FROM proveedores
WHERE #PROV IN ( ( SELECT  #PROV  FROM contratos
                     WHERE #ART = “10” )
                   INTERSECT
                   ( SELECT  #PROV  FROM contratos
                     WHERE #ART = “60” ) );
```

19) Obtener el nombre de aquellos proveedores que suministran el artículo 10.

19.a)
```
SELECT  NOMBRE  FROM proveedores
WHERE  #PROV IN ( SELECT  #PROV  FROM contratos
                        WHERE #ART = “10” );
```

19.b)
```
SELECT  NOMBRE  FROM proveedores p , contratos c
WHERE  p.#PROV = c.#PROV AND #ART = “10”;
```

19.c)
```
SELECT  NOMBRE  FROM proveedores p
WHERE  EXISTS ( SELECT  *  FROM contratos c
                    WHERE p.#PROV = c.#PROV AND #ART = “10” );
```

El predicado EXISTS toma el valor verdadero si el resultado del SELECT interior no es vacío, es decir, si existe al menos una fila de la tabla contratos que satisface la condición WHERE. En este caso, si al menos una fila de la tabla contratos se corresponde con el valor de proveedores.#PROV y con el valor 10 en #ART.

NOMBRE
Jaime
González
ACME

20) Obtener el nombre de aquellos proveedores que no suministran el artículo 10.

Esta consulta puede pensarse como “obtener el nombre de aquellos proveedores tal que no exista una fila en contratos que los relacione con el artículo 10”.

20.a)
```
SELECT  NOMBRE  FROM proveedores p
WHERE  NOT EXISTS ( SELECT  *  FROM contratos c
                        WHERE p.#PROV = c.#PROV AND #ART = “10” );
```

20.b)
```
SELECT  NOMBRE  FROM proveedores
WHERE  #PROV NOT IN ( SELECT  #PROV  FROM contratos
                        WHERE #ART = “10” );
```

21) Obtener los detalles de los proveedores que suministran todos los artículos.

Queremos obtener los proveedores para los que no exista un artículo que ellos no provean.

```
SELECT  *  FROM proveedores p
WHERE NOT EXISTS ( SELECT  *  FROM artículos a
                        WHERE NOT EXISTS ( SELECT  *  FROM contratos c
                                                WHERE p.#PROV = c.#PROV
                                                        AND
                                                a.#ART    = c.#ART ) );
```

La sentencia SELECT permite especificar las funciones MAX , MIN , COUNT , SUM y AVG para calcular el máximo, mínimo, cantidad de ocurrencias, suma y promedio de una columna.

22) Calcular la cantidad total de artículos.

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

6

23) Obtener la cantidad total de artículos actualmente suministrados.

```
SELECT  COUNT ( DISTINCT #ART )  FROM contratos;
```

6

¡¡¡¡ OJO !!!!

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

13

24) Calcular cuantos artículos suministra el proveedor 3.

```
SELECT  COUNT (*)  FROM contratos
WHERE #PROV = 3;
```

3

25) Obtener el nombre del proveedor con la mínima categoría.

```
SELECT  NOMBRE , "CATEGORIA MINIMA:" , CATEGORIA  FROM proveedores
WHERE CATEGORIA = ( SELECT MIN ( CATEGORIA )  FROM proveedores );
```

NOMBRE		CATEGORIA
González	CATEGORIA MINIMA:	10
XXX	CATEGORIA MINIMA:	10

26) Calcular la cantidad total, la cantidad promedio y la cantidad mínima suministrada del artículo 20.

```
SELECT  SUM ( CANT ) , AVG ( CANT ) , MIN ( CANT )  FROM contratos
WHERE #ART = "20";
```

600	200	100

27) Obtener el número de los proveedores con categoría menor que el valor promedio actual de categoría.

```
SELECT  #PROV  FROM proveedores
WHERE CATEGORIA < ( SELECT  AVG ( CATEGORIA )  FROM proveedores );
```

#PROV
4
5

28) Obtener la cantidad total suministrada de cada artículo.

```
SELECT  #ART , SUM ( CANT )  FROM contratos
GROUP BY #ART;
```

Esto se resuelve como si el operador GROUP BY agrupara las filas de la tabla referenciada en el FROM en particiones o grupos, tal que en cada grupo todas las filas tengan el mismo valor del atributo GROUP BY. La tabla resultado tendrá una fila por cada grupo.

#PROV	#ART	CANT
1	10	300
4	10	100
3	10	150
6	20	100
3	20	200
4	20	300
1	60	200
3	60	200
4	60	300
1	50	400
1	40	100
4	30	400
1	30	200

#ART	
10	550
20	600
60	700
50	400
40	100
30	600

29) Agrego a la consulta anterior la condición de que sólo me interesan aquellos artículos que son suministrados en una cantidad total mayor a 500.

```
SELECT  #ART , SUM ( CANT )  FROM contratos
GROUP BY #ART
HAVING SUM ( CANT ) > 500;
```

#ART	
10	550
20	600
60	700
30	600

La cláusula HAVING restringe la respuesta a aquellos grupos que satisfacen la condición especificada. La expresión en la cláusula HAVING debe tener un único valor por grupo. La cláusula HAVING es usada para eliminar grupos tal como la cláusula WHERE es usada para eliminar filas.

30) ¿ Cuál es la mayor cantidad suministrada ?

```
SELECT  MAX ( CANT )  FROM contratos;
```

400

31) ¿ Cuántos proveedores residen en Córdoba ?

31.a) SELECT COUNT (*) FROM proveedores
WHERE CIUDAD = “Córdoba”;

31.b) SELECT COUNT (#PROV) FROM proveedores
WHERE CIUDAD = “Córdoba”;

2

32) ¿ Cuántos artículos de color rojo están depositados en Rosario ?

SELECT COUNT (*) FROM artículos
WHERE COLOR = “rojo” AND CIUDAD = “Rosario”;

1

33) Listar los artículos suministrados por proveedores residentes en Córdoba y la cantidad total suministrada de cada uno.

33.a) SELECT #ART , SUM (CANT) FROM proveedores p , contratos c
WHERE p.#PROV = c.#PROV AND CIUDAD = “Córdoba”
GROUP BY #ART;

33.b) SELECT #ART , SUM (CANT) FROM contratos
WHERE #PROV IN (SELECT #PROV FROM proveedores
WHERE CIUDAD = “Córdoba”)
GROUP BY #ART;

#ART	
10	300
20	100
60	200
50	400
40	100
30	200

34) Contar la cantidad de artículos depositados en Rosario que son suministrados por proveedores en Rosario y la cantidad total suministrada.

34.a) SELECT COUNT (DISTINCT #ART) , SUM (CANT) FROM contratos
WHERE #PROV IN (SELECT #PROV FROM proveedores WHERE CIUDAD = “Rosario”)
AND
#ART IN (SELECT #ART FROM artículos WHERE CIUDAD = “Rosario”);

34.b) SELECT COUNT (DISTINCT c.#ART) , SUM (CANT)
FROM contratos c , proveedores p , artículos a
WHERE c.#PROV = p.#PROV AND c.#ART = a.#ART AND
p.CIUDAD = “Rosario” AND a.CIUDAD = “Rosario”;

1	500

35) ¿ Cuál es la cantidad promedio suministrada de cada artículo por los proveedores de cada ciudad ?

SELECT #ART , CUIDAD , AVG (CANT) FROM contratos c , proveedores p
WHERE p.#PROV = c.#PROV
GROUP BY #ART , CIUDAD;

El sistema realizaría el join natural (producto cartesiano , selección y proyección) entre proveedores y contratos:

c.#PROV	#ART	CANT	p.#PROV	NOMBRE	CATEGORIA	CIUDAD
1	10	300	1	Jaime	20	Córdoba
6	20	100	6	Celia	20	Córdoba
1	60	200	1	Jaime	20	Córdoba
1	50	400	1	Jaime	20	Córdoba
3	60	200	3	ACME	30	Rosario
1	40	100	1	Jaime	20	Córdoba
4	10	100	4	González	10	Rosario
4	60	300	4	González	10	Rosario
4	30	400	4	González	10	Rosario
1	30	200	1	Jaime	20	Córdoba
3	20	200	3	ACME	30	Rosario
4	20	300	4	González	10	Rosario
3	10	150	3	ACME	30	Rosario

lo agruparía por #ART, y dentro de #ART por CIUDAD de su proveedor :

c.#PROV	#ART	CANT	p.#PROV	NOMBRE	CATEGORIA	CIUDAD
1	10	300	1	Jaime	20	Córdoba
4	10	100	4	González	10	Rosario
3	10	150	3	ACME	30	Rosario
6	20	100	6	Celia	20	Córdoba
3	20	200	3	ACME	30	Rosario
4	20	300	4	González	10	Rosario
1	60	200	1	Jaime	20	Córdoba
3	60	200	3	ACME	30	Rosario
4	60	300	4	González	10	Rosario
1	50	400	1	Jaime	20	Córdoba
1	40	100	1	Jaime	20	Córdoba
4	30	400	4	González	10	Rosario
1	30	200	1	Jaime	20	Córdoba

obtendremos :

#ART	CIUDAD	
10	Córdoba	300
10	Rosario	125
20	Córdoba	100
20	Rosario	250
60	Córdoba	200
60	Rosario	250
50	Córdoba	400
40	Córdoba	100
30	Rosario	400
30	Córdoba	200

36) Averiguar la ciudad donde se almacenan más artículos.

36.a)
```
CREATE TABLE  temp1  (  C          INTEGER  NOT NULL  );
INSERT INTO temp1
SELECT  COUNT (*)  FROM artículos
GROUP BY CIUDAD;
```

C
1
2
3

```
SELECT  CIUDAD  FROM artículos
GROUP BY CIUDAD
HAVING COUNT (*) = ( SELECT  MAX ( C )  FROM temp1 );
```

CIUDAD
Córdoba

```
DROP TABLE  temp1;
```

36.b) CREATE TABLE temp2 (CIUDAD CHAR (15), C INTEGER);
INSERT INTO temp2
SELECT CIUDAD , COUNT (*) FROM artículos
GROUP BY CIUDAD;

CIUDAD	C
Jujuy	1
Rosario	2
Córdoba	3

SELECT CIUDAD FROM temp2
WHERE C = (SELECT MAX (C) FROM temp2);

CIUDAD
Córdoba

DROP TABLE temp2;

UPDATE

UPDATE tabla
SET columna = expresión , ……
[WHERE condición] ;

Todas las filas de la tabla que satisfacen la condición son modificadas de acuerdo a las asignaciones de la cláusula SET.

U1) Cambiar el color del artículo 50 a amarillo, aumentar su peso en 2 y fijar su ciudad como "desconocida" (NULL).

UPDATE artículos
SET COLOR = "amarillo" , PESO = PESO + 2 , CIUDAD = NULL
WHERE #ART = "50";

U2) Duplicar la categoría de todos los proveedores de Córdoba.

UPDATE proveedores
SET CATEGORIA = CATEGORIA * 2
WHERE CIUDAD = "Córdoba";

U3) Poner en cero la cantidad suministrada por los proveedores de Trelew.

U3.a)
```
UPDATE contratos
    SET CANT = 0
    WHERE #PROV IN ( SELECT  #PROV  FROM proveedores
                        WHERE CIUDAD = “Trelew” );
```

U3.b)
```
UPDATE contratos c
    SET CANT = 0
    WHERE “Trelew” = ( SELECT  CIUDAD  FROM proveedores p
                        WHERE p.#PROV = c.#PROV );
```

DELETE

```
DELETE  FROM  tabla
[ WHERE condición ] ;
```

Todas las filas de la tabla que satisfacen la condición son borradas.

D1) Eliminar al proveedor 1.

```
DELETE  FROM  proveedores
WHERE #PROV = 1;
```

¿ Qué pasa si la tabla contratos contiene suministros del proveedor 1 ?

D2) Eliminar todos los proveedores de Rosario.

```
DELETE  FROM  proveedores
WHERE CIUDAD = “Rosario”;
```

D3) Eliminar todos los contratos de suministro.

```
DELETE  FROM  contratos;
```

¿ Desaparece la tabla contratos ?

D4) Eliminar todos los suministros de proveedores cordobeses.

```
DELETE  FROM  contratos
WHERE #PROV IN ( SELECT  #PROV  FROM proveedores
                        WHERE CIUDAD = "Córdoba" );
```

¿ Existe otra forma de escribir esto ?

INSERT

La sentencia INSERT tiene dos formatos.

Para insertar una fila en una tabla:

```
INSERT  INTO  tabla  [ ( columna , ..... )  ]
VALUES  ( constante , ..... );
```

Para insertar n filas en una tabla:

```
INSERT  INTO  tabla  [ ( columna , ..... )  ]
subselect;
```

I1) Dar de alta el artículo 70, depositado en Córdoba, que pesa 2 kilos, la descripción y el color no están determinados por ahora.

I1.a)
```
INSERT  INTO  artículos ( #ART , CIUDAD , PESO )
    VALUES ( "70" , "Córdoba" , 2 );
```

I1.b)
```
INSERT  INTO  artículos
    VALUES ( "70" , NULL , 2 , NULL , "Córdoba"  );
```

I2) Insertar un contrato del proveedor 4 para suministrar 100 unidades del artículo 50.

```
INSERT  INTO  contratos
VALUES ( 4 , "50" , 100 );
```

¿ Podría darse el caso de que el artículo 50 no existiese en la tabla artículos ?

I3) Por cada artículo suministrado, obtener el nro. de artículo y la cantidad total suministrada de ese artículo, y salvar el resultado en una tabla temporaria.

```
CREATE TABLE  temp  ( #ART     CHAR ( 2 ) NOT NULL  ,  CANTOTAL      INTEGER );
INSERT  INTO  temp
SELECT  #ART , SUM ( CANT )  FROM contratos
GROUP BY #ART;
```

4. VISTAS.

Las vistas (o views) son el mecanismo que nos provee el lenguaje SQL para implementar el concepto de esquema externo. Una vista es una tabla virtual, es decir, que al crearse una vista no se está creando una copia de los datos, tan sólo se crea la definición de una visión particular de los datos.
Las vistas brindan los siguientes beneficios: simplificar la interacción con la base de datos, restringir los datos a los que tiene acceso el usuario de la vista, y facilitar la reestructuración de la base (independencia lógica de datos).

La sentencia LDD para crear una vista es:

```
CREATE VIEW  vista  AS  subselect;
```

Para eliminar una vista se utiliza la sentencia LDD:

```
DROP VIEW  vista;
```

Una vista se utiliza con las sentencias LMD igual que cualquier otra tabla.
Veamos algunos ejemplos:

V1) Crear una vista PROCOR con los datos de los proveedores cordobeses.

```
CREATE VIEW  procor  AS
SELECT  NOMBRE , CATEGORIA  FROM proveedores
WHERE CIUDAD = “Córdoba”;
```

V2) Usando la vista (tabla virtual) PROCOR, obtener el nombre del primer (en orden alfabético) proveedor cordobés.

```
SELECT  MIN ( NOMBRE )  FROM procor;
```

Para resolver este SELECT el sistema “traduce” la consulta sobre la vista PROCOR a una consulta sobre la (s) tabla (s) base de acuerdo a la definición de la vista:

```
SELECT  MIN ( NOMBRE )  FROM proveedores
WHERE CIUDAD = “Córdoba”;
```

V3) Crear una vista con los datos completos de los contratos.

```
CREATE VIEW  v1  ( #ART ,  DESC , PESO , COLOR , DEPOSITO , #PROV , NOMBRE ,
                        CAT , RESIDENCIA , CANTIDAD ) AS
SELECT  a.* , p.* , CANT  FROM proveedores p , artículos a , contratos c
WHERE p.#PROV = c.#PROV AND a.#ART = c.#ART;
```

Ejercicio : resolver los ejemplos 33, 34 y 35 anteriores usando esta vista v1.

Este ejemplo muestra que una vista puede estar definida sobre más de una tabla base. Si bien las vistas simplifican las consultas, pueden ocurrir inconvenientes cuando se quieren realizar actualizaciones.

Como ejemplo de los posibles inconvenientes que pueden surgir en la actualización de una vista, vamos a crear una vista con el número de artículo, descripción y la cantidad total suministrada de cada artículo.

```
CREATE VIEW  vvv  AS
SELECT  a.#ART , DESC , SUM ( CANT )  FROM artículos a , contratos c
WHERE a.#ART = c.#ART
GROUP BY a.#ART , DESC;

SELECT  * FROM vvv;
```

¿ Qué pasaría si quisiéramos dar de alta una fila en la vista vvv, del artículo 30, con descripción J y cantidad 250 ?

Recordemos que los datos de una vista están en realidad almacenados en las tablas base sobre la que está definida la vista. En este caso, los datos están almacenados en las tablas artículos y contratos.

Analice que pasaría en estas tablas al ejecutarse la sentencia:

```
INSERT INTO vvv
VALUES ( “30” , “J” , 250 );
```

Para evitar estos problemas, muchos sistemas han adoptado la siguiente restricción: solamente se permite la actualización de las vistas definidas sobre una sola tabla base y que además:

1) cada fila de la vista se corresponde con una fila diferente y distinguible de la tabla base.
2) cada columna de la vista se corresponde con una columna diferente y distinguible de la tabla base.

Al definir una vista no se puede especificar la cláusula ORDER BY.

¿ por qué ?

5. INDICES.

Un índice es un mecanismo para mejorar el rendimiento de las operaciones. Los índices son transparentes para el acceso a los datos, es decir, que la existencia o no de índices no afecta al LMD SQL. Nótese que hasta ahora hemos explicado todas las sentencias del LMD SQL (SELECT , INSERT , UPDATE , DELETE) sin hablar de los índices.

En general, un índice se implementa como una lista invertida sobre el campo (o campos) que se quiere indexar. Existirá una entrada en el índice por cada valor distinto de ese campo. Por cada entrada en el índice, figurarán apuntadores (pointers) a las filas (registros) que contengan ese valor.

Para crear un índice se usa la sentencia:

CREATE [UNIQUE] INDEX índice ON tabla (columna [{ ASC / DESC }] , ...);

La opción UNIQUE asegura que no existirá más de un apuntador por cada entrada en el índice.

Para eliminar un índice se usa: DROP INDEX índice [ON tabla];

Ejemplos:

X1) Crear un índice sobre el número de artículo para la tabla artículos.

```
CREATE UNIQUE INDEX i1 ON artículos ( #ART );
```

X2) Crear un índice sobre el color para la tabla artículos.

```
CREATE INDEX xcolo ON artículos ( COLOR DESC );
```

X3) Crear un índice para la clave primaria de la tabla contratos.

```
CREATE UNIQUE INDEX i55 ON contratos ( #PROV DESC , #ART );
```

En los casos en que el SGBD no incluye en su LDD, cláusulas para la implementación de claves primarias y/o claves foráneas, es conveniente aplicar la siguiente regla práctica para cada tabla:

- crear un índice UNIQUE para la clave primaria.
- crear un índice (solamente UNIQUE cuando los relacionamientos son funcionales en ambos sentidos, o sea, cardinalidad 1 : 1) para cada clave foránea.

Por último, un interrogante: no está permitida la definición de índices sobre vistas. ¿ Por qué ?

6. AUTORIZACIONES DE ACCESO.

Un aspecto importante en la administración de una base de datos es el tema de la seguridad de los datos. Si bien no toda la información es confidencial, la mayoría de los programas de aplicación y usuarios no necesitan tener acceso a la totalidad de la base de datos. SQL provee sentencias para otorgar y retirar privilegios de acceso a las tablas (bases y virtuales).
El creador de una tabla tiene todos los privilegios sobre esa tabla (lectura, inserción, borrado, actualización, etc) y puede delegar algunos o todos de esos privilegios a otros usuarios.

```
GRANT  {  privilegio , ..... / ALL  }  ON  { tabla  /  vista  }
TO  {  usuario , ..... / PUBLIC  }
[ WITH GRANT OPTION ];

REVOKE  {  privilegio , ..... / ALL  }  ON  { tabla  /  vista  }
FROM  {  usuario , ..... / PUBLIC  } ;

privilegio  :=  {  SELECT  /  INSERT  /  UPDATE  /  DELETE  /  ALTER  /  INDEX  }
```

donde:

SELECT	: consultar filas
INSERT	: insertar filas
UPDATE	: modificar filas
DELETE	: borrar filas
ALTER	: alterar estructura de tabla
INDEX	: crear índices

Veamos algunos ejemplos:

A1) Se ha decidido otorgar a todos los usuarios el privilegio de lectura sobre la tabla base de artículos.

```
GRANT  SELECT  ON  artículos  TO  PUBLIC;
```

A2) El privilegio de actualizar la columna COLOR de esa misma tabla se otorga a los usuarios SW45 y HH99 con la posibilidad de que ellos otorguen ese privilegio a otros usuarios.

```
GRANT  UPDATE ( COLOR )  ON  artículos  TO  SW45 , HH99
WITH  GRANT  OPTION;
```

A3) Retirarle al usuario X007 todos los privilegios que tenía sobre la vista vvv.

```
REVOKE  ALL  ON  vvv  FROM  X007;
```

7. IMPLEMENTACION DE LA INTEGRIDAD REFERENCIAL.

La definición de las claves primarias, como así también de las claves foráneas, constituyen un aspecto conceptual del diseño que, inevitablemente, tiene consecuencia en la performance de la solución y en la calidad de los datos.
En términos de Modelos de Datos, constituyen **restricciones explícitas**, que contribuyen a describir más acabadamente las reglas de gestión de la realidad que se representa. Tal vez por esta razón es que algunos DBMS las incluyen dentro de lo que denominan integrity constraints (restricciones de integridad).
Asimismo, algunos DBMS permiten identificar mediante un nombre a cada restricción de integridad, y ello, a su vez, habilita la posibilidad de activar (enable) o desactivar (disable) la restricción en un momento dado, a través de los comandos respectivos.

Por cada clave es necesario considerar tres aspectos:

1. ¿ Puede esa clave foránea aceptar valores nulos ?
   En otras palabras, ¿ es posible que exista una instancia de esa entidad para la cual no se conozca el valor de la clave foránea de referencia ?
   La respuesta NO depende del capricho del diseñador, sino del mundo real que se está modelizando o reglas de negocio.

2. ¿ Qué debe pasar si se intenta borrar una entidad cuya clave primaria es clave foránea en otras ?
   Por ejemplo : se intenta borrar un proveedor del cual existen suministros.
   Existen 3 posibilidades: CASCADA , RESTRINGIDO y NULIFICACION.

   CASCADA : La operación de borrado se propaga en cascada para borrar también los suministros asociados.
   RESTRINGIDO : La operación de borrado se restringe al caso en que no existan suministros asociados (caso contrario se rechaza).
   NULIFICACION : La clave foránea toma valor nulo en todos los suministros asociados y el proveedor es borrado (nótese que este caso no es aplicable si la clave foránea no acepta valores nulos).

3. ¿ Qué debería pasar si se intenta actualizar la clave primaria de una entidad que es clave foránea en otras ?
   Por ejemplo : se intenta actualizar el #PROV de un proveedor para el cual existe al menos un suministro asociado.
   Existen 3 posibilidades: CASCADA , RESTRINGIDO y NULIFICACION.

CASCADA : La operación de actualización se propaga en cascada para actualizar también la clave foránea en los suministros asociados (es la posibilidad más probable en la práctica).

RESTRINGIDO : La operación de actualización se restringe al caso en que no existan suministros asociados (caso contrario se rechaza).

NULIFICACION : La clave foránea toma valor nulo en todos los suministros asociados y el proveedor es entonces actualizado (nótese que este caso no es aplicable si la clave foránea no acepta valores nulos).

Nota:

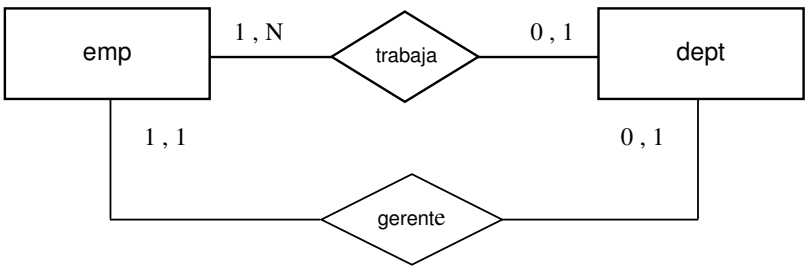
Los aspectos planteados deben considerarse desde un punto de vista conceptual, dado que no todos los DBMS permiten estas tres variantes.

Por esta misma razón, en los ejemplos que se presentan a continuación, utilizamos un pseudo-lenguaje SQL, de manera que la sintaxis correcta dependerá de cada dialecto particular.

Ejemplo 1 :

```
CREATE TABLE  pay   (#P            INTEGER NOT NULL,
                     #A            INTEGER NOT NULL,
                     #Y            INTEGER NOT NULL,
                     CANT          INTEGER,
                    PRIMARY KEY    ( #P , #A , #Y ),
                    FOREIGN KEY    ( #P ) REFERENCES proveedores ( #P )
                                   NULLS NOT ALLOWED
                                   DELETE CASCADES
                                   UPDATE CASCADES,.....);
```

Ejemplo 2 :



```
CREATE TABLE emp  (#EMP          INTEGER NOT NULL,
                   #DEPT         INTEGER         ,
                   ……            ……
                   PRIMARY KEY   ( #EMP ),
                   FOREIGN KEY   ( #DEPT ) REFERENCES dept ( #DEPT )
                                 NULLS  ALLOWED ( Un empleado puede no
                                 pertenecer a algún departamento )
                                 DELETE NULLIFIES ( Al borrarse un departamento,
                                 sus empleados asociados quedan desasignados )
                                 UPDATE CASCADES ( Al modificarse un #DEPT,
                                 se modifican también los empleados asociados )
                                 ,.....);

CREATE TABLE dept (#DEPT         INTEGER NOT NULL,
                   GERENTE       INTEGER NOT NULL,
                   ……            ……
                   PRIMARY KEY   ( #DEPT ),
                   FOREIGN KEY   ( GERENTE ) REFERENCES emp ( #EMP )
                                 NULLS NOT ALLOWED ( Todo departamento debe
                                 tener un gerente )
                                 DELETE RESTRICTED ( Los gerentes no pueden
                                 ser despedidos, antes deben ser removidos )
                                 UPDATE CASCADES ( Al modificarse el #EMP de
                                 un gerente, se realiza la misma modificación en el
                                 departamento asociado )
                                 ,.....);
```

Ejemplo 3 :

Aquellos tipos de entidades que dependen existencialmente de otras DEBEN especificar :

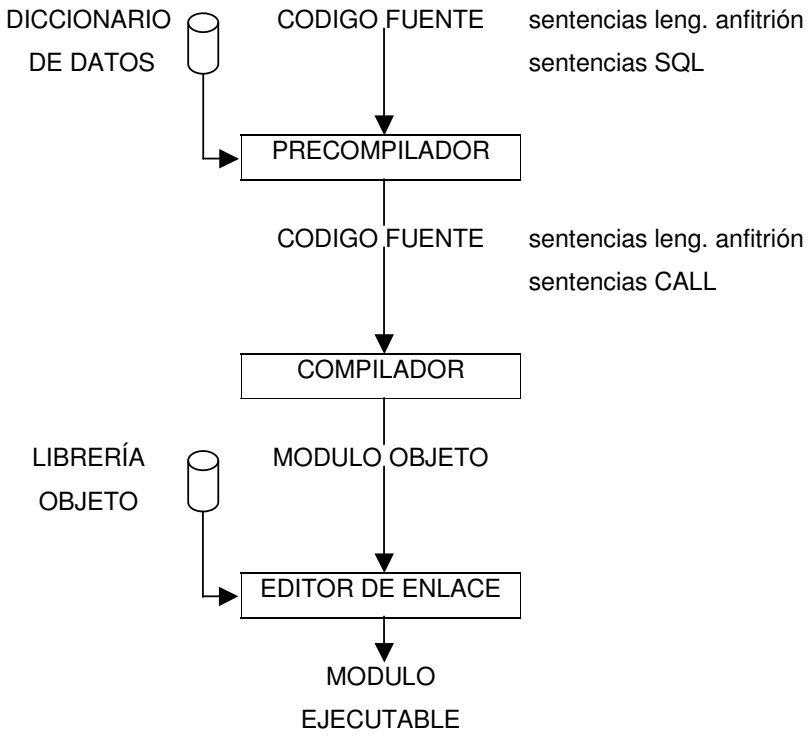
NULLS NOT ALLOWED
DELETE CASCADES
UPDATE CASCADES

para la clave foránea que referencia a la entidad de la que dependen.

```
CREATE   TABLE   familiar  (#FAMILIAR      INTEGER NOT NULL,
                            #EMP           INTEGER NOT NULL,
                            ……             ……
                           PRIMARY KEY     ( #FAMILIAR ),
                           FOREIGN KEY     ( #EMP ) REFERENCES empleado ( #EMP )
                                           NULLS NOT ALLOWED
                                           DELETE CASCADES
                                           UPDATE CASCADES,.....);
```

8. COMANDOS SQL INCLUIDOS EN UN LENGUAJE ANFITRION.

En los sistemas SYSTEM R, SQL/DS y DB2, las sentencias SQL se pueden emitir interactivamente o incluirse en un lenguaje de programación tal como COBOL o PL/1. El SGBD primero procesa las sentencias SQL incluidas en un programa, para traducirlas a proposiciones CALL que llaman a rutinas adecuadas del SGBD. Como lo muestra la figura siguiente una aplicación con sentencias SQL, pasa primero por un preprocesador y luego a través del compilador normal del lenguaje anfitrión.



En SQL/DS y DB2, una sentencia SQL incluida en un programa COBOL lleva el prefijo EXEC SQL. La sentencia termina con END-EXEC. El siguiente ejemplo muestra una sentencia SQL incluida en un programa COBOL para recuperar el nombre y peso del artículo nro. I3. Dentro de una sentencia SQL, las variables locales son referidas con un prefijo ":".

```
EXEC SQL      SELECT  DESC , PESO
              INTO :ARTI , :PESO
              FROM  artículos
              WHERE  #ART = "I3"
END-EXEC.
```

El siguiente es un ejemplo de sentencias SQL incluidas en un programa COBOL a ser procesado en un sistema SQL/DS o DB2. La sentencia INCLUDE se emite para incluir una estructura llamada SQLCA (SQL Communication Area), la cual es utilizada por el SGBD para informar sobre la ejecución de la sentencia SQL.Uno de los campos de la SQLCA es el SQLCODE. Un valor 0 en el SQLCODE significa que la ejecución fue exitosa. Un valor negativo indica una condición de error mientras que valores positivos indican condición normal. Por ejemplo, el valor 100 de SQLCODE significa que se llegó al final de los datos.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL  BEGIN DECLARE SECTION  END-EXEC.
77  ARTI         PIC    X (02).
77  PESO         PIC    9 (06).
77  NRO-ART      PIC    X (02).
    EXEC SQL  END DECLARE SECTION  END-EXEC.
    EXEC SQL  INCLUDE SQLCA  END-EXEC.
PROCEDURE DIVISION.
    ACCEPT NRO-ART.
    EXEC SQL
       SELECT  DESC , PESO
       INTO :ARTI , :PESO
       FROM artículos
       WHERE #ART = :NRO-ART
    END-EXEC.
    IF SQLCODE = 0 THEN DISPLAY ARTI , PESO.
    STOP RUN.
```

Uno de los problemas de incluir sentencias SQL en un programa de aplicación, es que el lenguaje anfitrión está preparado para manejar sentencias de entrada/salida solamente para un registro a la vez. Sin embargo, la sentencia SQL SELECT puede recuperar varios registros a la vez.
Para resolver tales conflictos en el modo de operación de E/S entre SQL y el lenguaje de programación convencional, algunos sistemas relacionales proporcionan un mecanismo cursor para traer un registro a la vez de una tabla recuperada mediante una sentencia SQL, como se mostrará a continuación.

Se puede asignar un cursor a una tabla y el indicador de cursor recorrerá la tabla a medida que cada fila de la tabla se mueva del buffer al área de datos del programa. En SQL/DS y DB2 se declara un cursor con la sentencia DECLARE para asociarlo con una expresión SQL como sigue:

```
EXEC SQL
DECLARE  nombre-cursor CURSOR  FOR
SELECT  NOMBRE  FROM proveedores
WHERE #PROV IN ( SELECT  #PROV  FROM contratos WHERE #ART = :NRO-ART )
END-EXEC.
```

Al cursor se le asigna un nombre y en un programa se pueden definir varios cursores. En el momento de declarar un cursor no se recupera ningún dato.
Existen tres operadores cursor: OPEN , FETCH y CLOSE. Estos operadores se usan para activar, avanzar y desactivar al cursor respectivamente.

```
EXEC SQL  OPEN  nombre-cursor  END-EXEC.
```

```
EXEC SQL CLOSE nombre-cursor  END-EXEC.
```

```
EXEC SQL  FETCH  nombre-cursor  INTO  :NOMBRE  END-EXEC.
```

La sentencia FETCH causa que el cursor avance al siguiente renglón de la tabla y mueva el registro indicado por el cursor del buffer del sistema hacia los campos de datos especificados en la cláusula INTO. La sentencia FETCH está normalmente codificada dentro de un ciclo para que los registros en la tabla se puedan procesar uno por uno.

El siguiente es un ejemplo donde se incluyen operadores cursor en un programa COBOL. El programa efectúa una consulta SQL y muestra los datos del nombre de aquellos proveedores que suministran el artículo cuyo número es igual a un valor ingresado desde una terminal.
La proposición DECLARE en la PROCEDURE DIVISION asigna un cursor curs1 a la expresión SQL que sigue. El cursor se activa para el procesamiento mediante la proposición OPEN. Las proposiciones FETCH recuperan el primer registro y los subsecuentes en la tabla respectivamente.
Cuando se procesaron todos los registros de la tabla, el cursor se desactiva con el comando CLOSE.

```
IDENTIFICATION DIVISION.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
    EXEC SQL  BEGIN DECLARE SECTION  END-EXEC.
77  NOMBRE    PIC    X (10).
77  NRO-ART   PIC    X (02).
    EXEC SQL  END DECLARE SECTION  END-EXEC.
    EXEC SQL  INCLUDE SQLCA  END-EXEC.
```

```
PROCEDURE DIVISION.
PRINCIPAL.
   EXEC SQL
      DECLARE  curs1  CURSOR  FOR
      SELECT  NOMBRE  FROM proveedores
      WHERE #PROV IN ( SELECT  #PROV  FROM contratos
                              WHERE #ART = :NRO-ART )
   END-EXEC.
   ACCEPT NRO-ART.
   EXEC SQL  OPEN  curs1  END-EXEC.
   EXEC SQL  FECTH  curs1  INTO  :NOMBRE  END-EXEC.
   PERFORM  PROCESAR-TABLA  UNTIL  SQLCODE  NOT = 0.
   EXEC SQL  CLOSE  curs1  END-EXEC.
   STOP RUN.
PROCESAR-TABLA.
   DISPLAY NOMBRE.
   EXEC SQL  FECTH  curs1  INTO  :NOMBRE  END-EXEC.
```

9. SINTAXIS SQL.

La sintaxis de presentación de las sentencias SQL es la siguiente:

[a]	significa que 'a' es opcional
{ a / b }	significa que se debe elegir una de las opciones 'a' ó 'b' (o excluyente)
a , ...	significa que 'a' puede repetirse 1 a n veces

Las palabras claves van en mayúsculas.
Las palabras clave default van subrayadas.
Los paréntesis y comillas deben ser escritos tal como aparecen.

SQL DATA DEFINITION COMMANDS

ALTER TABLE tabla **ADD** (columna tipo [**NOT NULL**] , ...);

ALTER TABLE tabla **MODIFY** (columna [tipo] [**NOT NULL**] , ...);

CREATE [**UNIQUE**] **INDEX** índice **ON** tabla (columna [{ **ASC** / **DESC** }] , ...);

CREATE TABLE tabla (columna tipo [**NOT NULL**] , ...);

tipo := { **CHAR** (tamaño) / **DATE** / **DECIMAL** / **FLOAT** / **INTEGER** / **SMALLINT** / **VARCHAR** / **LONG VARCHAR**}

CREATE VIEW vista [(columna , ...)] **AS** subselect;

DROP { **INDEX** índice [**ON** tabla] / **TABLE** tabla / **VIEW** vista };

SQL ACCESS CONTROL COMMANDS

GRANT { privilegio , ... / **ALL** } **ON** { tabla / vista }
TO { usuario , ... / **PUBLIC** }
[**WITH GRANT OPTION**];

REVOKE { privilegio , ... / **ALL** } **ON** { tabla / vista }
FROM { usuario , ... / **PUBLIC** };

privilegio := { **SELECT** / **INSERT** / **UPDATE** / **DELETE** / **ALTER** / **INDEX** }

SQL DATA MANIPULATION COMMANDS

UPDATE tabla [alias]
SET columna = expresión , ...
[**WHERE** condición];

DELETE FROM tabla
[**WHERE** condición];

INSERT INTO tabla [(columna , ...)]
{ **VALUES** (constante , ...) / subselect };

select := { subselect [**ORDER BY** { expresión / posición } [{ **ASC** / **DESC** }] …..] /
subselect { **UNION** / **INTERSECT** / **MINUS** } (select) };

subselect := **SELECT** [**DISTINCT**] { [tabla.] * / expresión , ……..}
FROM tabla [alias] , ……
[**WHERE** condición]
[**GROUP BY** expresión , ……. [**HAVING** condición]

condición := { [**NOT**] predicado / [**NOT**] predicado { **AND** / **OR** } condición }

predicado := {

expresión	θ	{ expresión / (subselect) }
/ expresión	[**NOT**] **BETWEEN**	(expresión **AND** expresión)
/ columna	**IS** [**NOT**] **NULL**	
/	**EXISTS**	(subselect)
/ expresión	[**NOT**] **IN**	{ (constante , ...) / (subselect) }

}

expresión := {

[{ + / - }]	{ función / constante / columna }
/ [{ + / - }]	{ función / constante / columna } { + / - / * / % } expresión

}

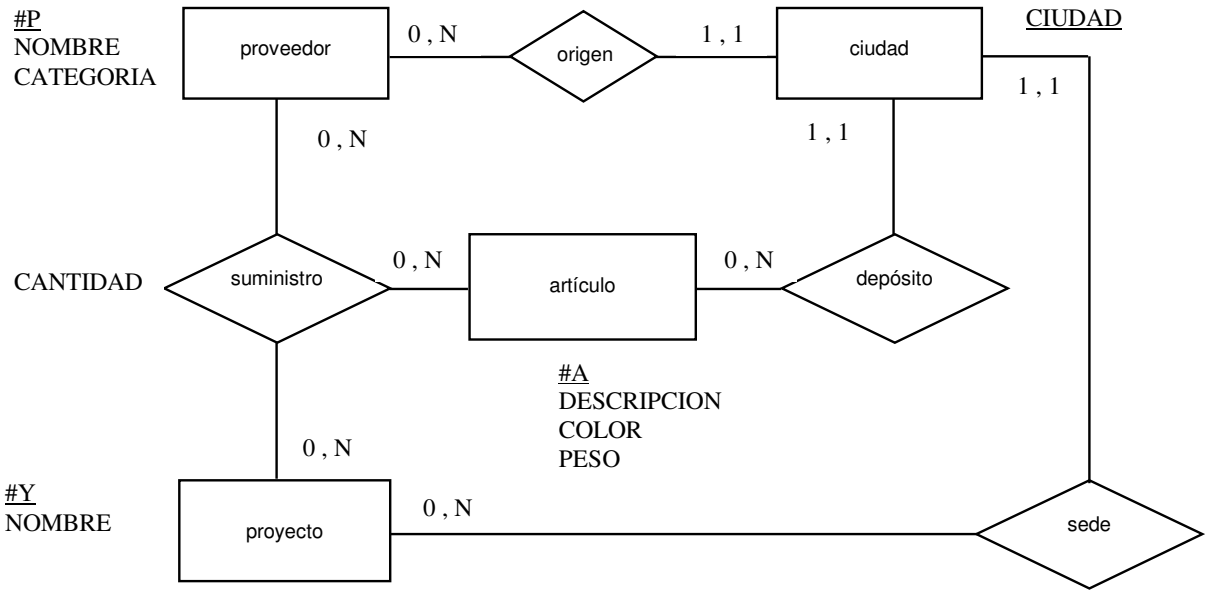
función := {{ **MAX** / **MIN** / **SUM** / **AVG** } (columna) / **COUNT** ([**DISTINCT**] { * / columna }) }

θ := { > / < / = / >= / <= / <> }

Observación: No se admite como < condición > de la cláusula WHERE < expresiones > que involucren < funciones >.

10. EJERCITACION PROPUESTA SQL.

1) Transformar el siguiente diagrama E-R a un modelo Relacional.



2) Definir las tablas correspondientes en SQL. Usar esas tablas para los demás ejercicios.

INSERT , DELETE , UPDATE

3) Cambiar el color de todos los artículos rojos a violeta.
4) Eliminar todos los proyectos que no reciben suministros.
5) Eliminar todos los proyectos con sede en San Luis y sus correspondientes suministros.
6) Dar de alta un nuevo proveedor : su nombre es ACME, su número es 9 y reside en Avellaneda. Todavía no se le ha asignado una categoría.
7) Construir una tabla que contenga los números de los artículos provistos por algún proveedor cordobés o suministrados a algún proyecto cordobés.
8) Aumentar un 10 % el peso de todos los artículos sin suministros.

SELECT

9) Obtener todos los datos de todos los proyectos con sede en Quilmes.
10) Listar los suministros cuya cantidad esté entre 300 y 600 inclusive.
11) Obtener los números de aquellos artículos suministrados por proveedores rosarinos.
12) Listar todos los pares de ciudades tales que un proveedor de la primera ciudad provea a un proyecto de la segunda ciudad.
13) Listar los números de aquellos artículos suministrados a algún proyecto por un proveedor que resida en la misma ciudad que el proyecto.
14) Listar los números de todos los proyectos abastecidos por lo menos por un proveedor que no resida en la misma ciudad que el proyecto.
15) Obtener todos los pares de números de artículos tales que algún proveedor suministre ambos productos.
16) Listar la cantidad total de proyectos suministrados por el proveedor 8.
17) Por cada artículo suministrado a un proyecto, obtener el número de artículo, el número de proyecto y la cantidad total suministrada ordenada por número de proyecto (ascendente) y cantidad (descendente).
18) Listar el número de artículo de aquellos artículos suministrados a algún proyecto en una cantidad promedio mayor a 345.
19) Listar todos los proveedores que no tengan categoría asignada.
20) Obtener los números de aquellos proveedores cuya categoría sea menor a la del proveedor 66.
21) Listar los números de aquellos proyectos cuya ciudad es la primera en una lista alfabética de ciudades.
22) Listar los números de aquellos proveedores que abastecen algún proyecto con el artículo 10 en una cantidad mayor que la cantidad promedio en que el artículo 10 es suministrado a ese proyecto.
23) Listar los números de aquellos proyectos que son abastecidos exclusivamente por el proveedor 3.
24) Listar los números de aquellos artículos que son suministrados a todos los proyectos con sede en Lanús.
25) Listar los números de aquellos proveedores que suministran, al menos, el mismo artículo a todos los proyectos.
26) Listar los números de aquellos proyectos abastecidos, al menos, con todos los artículos suministrados por el proveedor 8.

VISTAS

27) Definir una vista CORDOBA con los números de los artículos provistos por algún proveedor cordobés o suministrados a algún proyecto cordobés.
28) Crear una vista CATEGORIA-PROVISION que brinde, para cada categoría de proveedor, la cantidad total suministrada de cada artículo.

AUTORIZACIONES DE ACCESO

29) Autorizar al usuario U888 a consultar las vistas de los ejercicios anteriores, y a otorgar este privilegio a otros usuarios si lo desea.
30) Autorizar a los usuarios K111 y C321 a insertar tuplas en la tabla proyectos.
31) Retirarle al usuario U777 todos los privilegios que tenía sobre las tablas proveedores y proyectos.
32) Autorizar a todos los usuarios a modificar el color en la tabla de artículos.

INDICES

33) Definir los índices necesarios para implementar las claves foráneas de las tablas creadas en el ejercicio 2.

INTEGRIDAD REFERENCIAL

34) Definir las claves primarias y foráneas de cada tabla.

SQL INCLUIDO

35) Escribir un programa (en un lenguaje anfitrión) con sentencias SQL inmersas, para listar los datos de los proveedores ordenados por número de proveedor. Por cada proveedor listar los números de proyecto que ese proveedor suministra, ordenados por número de proyecto.
36) Revisar el ejercicio anterior para que no aparezcan listados aquellos proveedores que no suministran a proyecto alguno.

11. RESOLUCION ILUSTRATIVA EJERCICIO 35.

1) Lenguaje anfitrión : COBOL

```
       IDENTIFICATION DIVISION.
       PROGRAM-ID.   EJERC-35.
       etc.....................................
       .....................................
       DATA    DIVISION.
       FILE    SECTION
      *
       FD      IMPRESORA
               LABEL RECORD OMITTED.
       01      REG-IMPRE                   PIC X(132).
      *
      * ---------  ----------------------------------------  ------------------------------------------------------------------------*
       WORKING-STORAGE SECTION.
               EXEC SQL BEGIN DECLARE   SECTION  END-EXEC.
       77      WS-NRO-PROV                 PIC X(03).
       77      WS-NOMBRE-PROV              PIC X(30).
       77      WS-CITY-PROV                PIC X(20).
       77      WS-CATEG-PROV               PIC 9(04).
       77      WS-NRO-PROY                 PIC X(03).
               EXEC SQL  END  DECLARE   SECTION  END-EXEC.
               EXEC SQL INCLUDE SQLCA   END-EXEC.
       77      CONT-LINE                   PIC 9(03) VALUE 71.
       77      COD_ESTADO                  PIC ----9 VALUE +0.
      *
       01      LINEA-TITULOS.
               03 FILLER.....
       etc.....  .....................................................
      *
```

```
PROCEDURE DIVISION.
MAIN.
        EXEC SQL WHENEVER       SQLERROR GOTO RUT-ERROR END-EXEC.
        EXEC SQL                DECLARE CS-PROV CURSOR FOR
                                SELECT  *  FROM   PROVEEDORES
                                ORDER BY 1        END-EXEC.
        EXEC SQL                DECLARE CS-PROY CURSOR FOR
                                SELECT DISTINCT NRO-Y FROM SUMINISTROS
                                WHERE NRO-P = :WS-NRO-PROV
                                ORDER BY 1        END-EXEC.
        OPEN IMPRESORA.
        PERFORM PROCESO-T-PROV.
        CLOSE IMPRESORA.
        STOP RUN.
PROCESO-T-PROV.
        EXEC SQL OPEN   CS-PROV     END-EXEC.
        EXEC SQL FETCH CS-PROV
                INTO                :WS-NRO-PROV , :WS-NOMBRE-PROV ,
                                    :WS-CATEG-PROV , :WS-CITY-PROV
                                    END-EXEC.
        PERFORM PROCESO-1-PROV      UNTIL SQLCODE = 100.
        EXEC SQL CLOSE CS-PROV      END-EXEC.
        EXEC SQL COMMIT             END-EXEC.
PROCESO-1-PROV.
        PERFORM IMPRIMIR-PROV.
        EXEC SQL OPEN   CS-PROY     END-EXEC.
        EXEC SQL FETCH CS-PROY
                INTO                :WS-NRO-PROY
                                    END-EXEC.
        PERFORM PROCESO-1-PROY      UNTIL SQLCODE = 100.
        EXEC SQL CLOSE CS-PROY      END-EXEC.
        EXEC SQL FETCH CS-PROV
                INTO                :WS-NRO-PROV , :WS-NOMBRE-PROV ,
                                    :WS-CATEG-PROV , :WS-CITY-PROV
                                    END-EXEC.
PROCESO-1-PROY.
        PERFORM IMPRIMIR-PROY.
        EXEC SQL FETCH CS-PROY
                INTO                :WS-NRO-PROY
                                    END-EXEC.
```

```
   RUT-ERROR.
            MOVE SQLCODE TO COD-ESTADO.
            EXEC SQL WHENEVER SQLERROR          CONTINUE END-EXEC.
            DISPLAY 'ERROR, EL CODIGO DE STATUS  ES:', COD-ESTADO.
            EXEC SQL  ROLLBACK  END-EXEC.
            STOP RUN.
   etc……
   ……
```

2) Lenguaje anfitrión : C

```
#include  <stdio.h>
/**********  VARIABLES DE INTERFASE CON LA BASE DE DATOS  **********/
EXEC SQL BEGIN DECLARE SECTION;
/*  Username                */  varchar                 userid [ 40 ];
/*  Clave                   */  long                    nro_prov;
/*  Clave                   */  long                    nro_proy;
/*  Descrip                 */  varchar                 nombre_prov [ 20 ];
/*  Dato                    */  long                    cat_prov;
/*  Dato                    */  long                    city_prov;
EXEC SQL END DECLARE SECTION;
/**********  CONTROL DE INTERFASE ENTRE RDBMS Y PROGRAMA  **********/
EXEC SQL INCLUDE SQLCA;
/**********  VARIABLES DE USO INTERNO DEL PROGRAMA  **********/
FILE  *fdw;
int  exitcode=0;……
/**********  DEFINICION DE FUNCIONES  **********/
void proceso_todo_proveedor ( );
void proceso_un_proveedor ( long , varchar , long , long );
void proceso_todo_proyecto ( );
void proceso_un_proyecto ( long , long );
void imprimo_proveedor ( FILE* , long , varchar , long , long );
void imprimo_proyecto ( );
void sqlerror ( );
/**********  COMIENZO PROGRAMA PRINCIPAL  **********/
int main ( int argc , char  *argv [ ] )
{
/**********  CONEXIÓN A LA BASE DE DATOS SEGÚN PARAMETRO EXTERNO  **********/
  strcpy ( userid , argv [ 1 ] );
  EXEC SQL WHENEVER SQLERROR "DO" sqlerror;  /*  CONTINUE , GOTO , EXIT  */
  EXEC SQL CONNECT :USERID;
```

```
/**********  DECLARACION DE LOS CURSORES  **********/
  EXEC SQL
    DECLARE cs_prov CURSOR FOR
    SELECT * FROM proveedores
    ORDER BY 1;
  EXEC SQL
    DECLARE cs_proy CURSOR FOR
    SELECT DISTINCT nro_y FROM suministros
    WHERE nro_p = :nro_prov
    ORDER BY nro_y;
fdw = fopen ( argv [ 2 ] );          /*** ABRO ARCHIVO DE IMPRESION SEGUN PARAMETRO  ***/
proceso_todo_proveedor ( );      /*** PROCESO PRINCIPAL                                ***/
fclose ( fdw );                      /*** CIERRO ARCHIVO DE IMPRESION                ***/
exit ( exitcode );
/**********  SALIDA ANORMAL DEL PROGRAMA  **********/
void sqlerror ( )
    printf ( "\n\%.170s\n\n" , sqlca.sqlerrm.sqlerrmc );
    exitcode = sqlca.sqlcode;
    EXEC SQL WHENEVER SQLERROR CONTINUE;
    EXEC SQL ROLLBACK WORK RELEASE;
    exit ( exitcode );
}
/*************************************************************************************************************/
void proceso_todo_proveedor ( )
{
  EXEC SQL OPEN cs_prov;
  for ( ;; )
  {
    EXEC SQL
      FETCH cs_prov
      INTO :nro_prov , :nombre_prov , :cat_prov , :city_prov;
    proceso_un_proveedor (nro_prov , nombre_prov , cat_prov , city_prov );
    if  ( sqlca.sqlcode == 100 )
      break;
  }
  EXEC SQL CLOSE cs_prov;
  EXEC SQL COMMIT WORK RELEASE;
}
```

```
void proceso_un_proveedor ( long nro_prov , varchar nombre_prov , long cat_prov ,
                                    long city_prov );
{
 imprimo_proveedor ( fdw , nro_prov , nombre_prov , cat_prov , city_prov);
 .....
 EXEC SQL OPEN cs_proy;
 for ( ;; )
 {
  EXEC SQL
    FETCH cs_proy
    INTO :nro_proy;
  proceso_un_proyecto (nro_prov , nro_proy );
  if  ( sqlca.sqlcode == 100 )
    break;
 }
 EXEC SQL CLOSE cs_proy;
}
void proceso_un_proyecto ( long , long );
{
 imprimo_proyecto (  );
 .....
}
void imprimo_proveedor ( FILE* fdw , long nro_prov , varchar nombre_prov , long cat_prov ,
                                  long city_prov );
{
 fprintf  ( fdw , " %d  %s  %d  %d  \n" , nro_prov , nombre_prov , cat_prov , city_prov );
}
void imprimo_proyecto (  );
{
 fprintf  ( ……..
}
```