élogos

ACCESO A DATOS EN JAVA

élogos

Fundamentos del lenguaje SQL

LMD (Lenguaje de manipulación de datos)

SELECT

```
SELECT [DISTINCT] <lista_columnas> | *
        FROM <lista_tablas>
        [WHERE <predicado>]
```

EJ: Visualizar todos los vuelos que tengan como origen o destino Cáceres.

```
SELECT *
        FROM VUELOS
        WHERE ORIGEN='CACERES'
        OR DESTINO='CACERES'
```

EJ: Visualizar todos los vuelos que tengan como origen Madrid o Londres y como destino Londres o Madrid.

```
SELECT *
        FROM VUELOS
        WHERE (ORIGEN='MADRID'
                AND DESTINO='LONDRES')
        OR (ORIGEN='LONDRES'
                AND DESTINO='MADRID')
```

Claúsula IN

Expresa la pertenencia del valor de una columna a un determinado conjunto de valores.

EJ: Seleccionar aquellos vuelos que tengan como origen Madrid, Barcelona o Sevilla.

```
SELECT *
        FROM VUELOS
        WHERE ORIGEN IN ('MADRID','BARCELONA','SEVILLA')
```

O también

```
SELECT *
        FROM VUELOS
        WHERE ORIGEN='MADRID' OR ORIGEN='BARCELONA' OR ORIGEN='SEVILLA'
```

EJ: Visualizar todos los vuelos existentes excepto aquellos que llegan a Londres o a Copenhague.

```
SELECT *
        FROM VUELOS
        WHERE DESTINO NOT IN ('LONDRES','COPENHAGUE')
```

Claúsula BETWEEN

Sirve para establecer o expresar un rango de valores. Obedece a la siguiente sintaxis:

<nombre_columna> BETWEEN valor1 AND valor2

El rango será [valor1, valor2], extremos incluidos.

EJ: Recuperar todos los vuelos que salgan entre las 6 y las 12 de la mañana.

```
SELECT *
        FROM VUELOS
        WHERE HORA_SALIDA BETWEEN '06.00.00'
                                AND '12.00.00'
```

ó también

```
SELECT *
        FROM VUELOS
        WHERE HORA_SALIDA >= '06.00.00'
        AND HORA_SALIDA <= '12.00.00'
```

EJ: En la columna NUM_VUELO representaré los vuelos con 6 caracteres. Los dos primeros caracteres indicarán la compañía a la que pertenece cada vuelo (IB®Iberia, BA®British Airways), los cuatro caracteres siguientes corresponderán al número de vuelo. Bajo estas condiciones recupérense todos los vuelos que no pertenecen a IBERIA.

```
SELECT *
        FROM VUELOS
        WHERE NUM_VUELO NOT BETWEEN 'IB0000'
                                AND 'IB9999'
```

Claúsula LIKE

Sirve para especificar, con la ayuda de metasímbolos, cadenas de caracteres que comparten ciertos caracteres en común. Los metasímbolos que serán utilizados son:

% Equivale a una cadena de caracteres de longitud comprendida entre 0 y n.
'AB%' AB, ABCDE, AB 497

_ Equivale a un único carácter
'A_B' A B, A4B, AJB

EJ: Recuperar todos los vuelos pertenecientes a la compañía IBERIA.

```
SELECT *
        FROM VUELOS
        WHERE NUM_VUELOS LIKE 'IB%'
```

O también

```
SELECT *
        FROM VUELOS
        WHERE NUM_VUELOS LIKE 'IB_ _ _ _'
```

Expresiones aritméticas

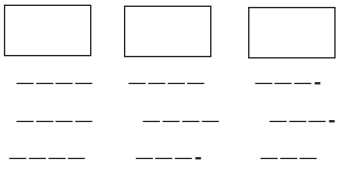
+, -, *, /

Pueden ser utilizadas tanto después de SELECT como después de WHERE. En el primer caso trabajarían sobre columnas y en el segundo sobre filas.

EJ: Visualizar la longitud y la envergadura de todos los aviones, expresando las magnitudes en pies (en la base de datos está almacenado en metros, para pasar 1 metro a pies se ha de multiplicar por 3.28), y la velocidad de crucero en mph (está en Km/h, habrá que dividir por 1.6).

```
SELECT LONGITUD*3.28, ENVERGADURA*3.28, VELO_CRUC/1.6
        FROM AVIONES
```

Etiquetas ®



En DB/2 de IBM las etiquetas toman los nombres de las columnas (col1, col2, col3)
En SQL-SERVER las etiquetas quedarían así (LONGITUD*3.28, ENVERGADURA*3.28, VELO_CRUC/1.6)

EJ: Relación entre la longitud y la envergadura de todos los aviones.

```
SELECT LONGITUD/ENVERGADURA
        FROM AVIONES
```

EJ: Seleccionar aquellos aviones cuya longitud supere a su envergadura en más de un 10%.

```
SELECT *
        FROM AVIONES
        WHERE LONGITUD > ENVERGADURA*1.10
```

Funciones de columna

Son funciones que operan con todas las filas que cumplen la condición expuesta en la claúsula WHERE. Su resultado es un único valor. Sintaxis:

1º) <f_columna> ([DISTINCT] <nombre_columna>)
2º) <f_columna> (<expresión>), donde <expresión> es una expresión aritmética en la cual debe participar, al menos, una columna.
3º) COUNT(*)

Funciones

<f_columna>:

MIN: Calcula el valor mínimo de una columna.
MAX: Calcula el valor máximo de una columna.
AVG: Calcula la media aritmética de una columna.
SUM: Calcula la suma de todos los campos de una columna.
COUNT: Cuenta el nº de filas de una columna.

A	B
3	5
2	8
3	7
4	3

COUNT(A)=COUNT(B)
COUNT(A)=4, COUNT(B)=4
El COUNT de dos columnas de una misma tabla es igual. COUNT(*) sirve para obtener el nº de filas.

EJ: Seleccionar los valores mínimo y máximo de la columna que almacena las velocidades de crucero.

```
SELECT MIN(VELO_CRUC), MAX(VELO_CRUC)
        FROM AVIONES
```

EJ: Averiguar a qué hora parte el primer vuelo hacia Madrid.

```
SELECT MIN (HORA_SALIDA)
        FROM VUELOS
        WHERE DESTINO='MADRID'
```

Regla que cumplen las funciones de columna

La función de columna sólo podrá especificarse detrás de la particula SELECT o en la claúsula HAVING, pero nunca dentro de la claúsula WHERE.

EJ: Se desea saber cuál es el vuelo que tiene la mínima hora de salida.

```
SELECT *
        FROM VUELOS
        WHERE HORA_SALIDA=(SELECT MIN(HORA_SALIDA)  FROM VUELOS)
```

Claúsula GROUP BY-HAVING

Sirve para dividir una tabla en grupos de filas que comparten características comunes. La sintaxis es:

```
SELECT <lista_columnas>, <funciones_de_columna>
        FROM <lista_tablas>
        [WHERE <predicado>]
        [GROUP BY <lista_columnas>]
        [HAVING <predicado>]
```

EJ: Efectúese una SELECT que visualice el mínimo valor de hora de salida para cada uno de los diferentes destinos.

```
SELECT DISTINCT DESTINO
        FROM VUELOS

SELECT MIN(HORA_SALIDA)
        FROM VUELOS
        WHERE DESTINO LIKE '%'
```

A continuación se muestra un ejemplo de lo que no se debe hacer:

```
SELECT MIN(HORA_SALIDA)
        FROM VUELOS
        WHERE DESTINO IN (SELECT DISTINCT DESTINO
                                FROM VUELOS)
```

Sentencia GROUP BY:

```
SELECT DESTINO, MIN(HORA_SALIDA)
        FROM VUELOS
        GROUP BY DESTINO
```

Tabla VUELOS ® Tabla auxiliar ® Tabla x 'MADRID'
WHERE GROUP BY DESTINO Tabla y 'BARCELONA'
Tabla z 'SEVILLA'
<f_columna>

GROUP BY crea una serie de subtablas compuestas por filas con el mismo valor para la columna de agrupamiento (en este ejemplo la columna DESTINO). Se aplicarán a continuación funciones de columna sobre cada subtabla de forma independiente.

MADRID, x
BARCELONA, y
SEVILLA, z

No se puede poner en GROUP BY un campo que no se haya incluido en la sentencia SELECT.

EJ: Obtener el origen del vuelo para cada uno de los vuelos que tienen la mínima hora de salida para cada uno de los destinos.

EJ: Obtener el número de vuelos que existen para cada uno de los orígenes.

```
SELECT ORIGEN, COUNT(*)
        FROM VUELOS
        GROUP BY ORIGEN
```

Claúsula HAVING

Permite elegir aquellos grupos que se quieren visualizar.

EJ: Visualizar los grupos que tienen para cada uno de los orígenes la mínima hora de salida siendo anterior a las 12 horas.

```
SELECT ORIGEN, MIN(HORA_SALIDA)
        FROM VUELOS
        GROUP BY ORIGEN
        HAVING MIN(HORA_SALIDA) < '12.00'
```

HAVING no interferirá en la agrupación por filas de GROUP BY.

EJ: Se desea seleccionar la hora de salida más temprana para cada origen y destino.

```
SELECT ORIGEN, DESTINO, MIN(HORA_SALIDA)
        FROM VUELOS
        GROUP BY ORIGEN, DESTINO
```

EJ: Visualizar los orígenes que tengan más de dos vuelos.

```
SELECT ORIGEN
        FROM VUELOS
        GROUP BY ORIGEN
        HAVING COUNT(*) > 2
```

EJ: Visualizar los vuelos de IBERIA que tengan más de 150 plazas libres.

```
SELECT NUM_VUELO, SUM(PLAZAS_LIBRES)
        FROM RESERVAS
        GROUP BY NUM_VUELO
        HAVING NUM_VUELO LIKE 'IB%'
        AND SUM(PLAZAS_LIBRES)>150
```

O también

```
SELECT NUM_VUELO, SUM(PLAZAS_LIBRES)
        FROM RESREVAS
        WHERE NUM_VUELO LIKE 'IB%'
        GROUP BY NUM_VUELO
        HAVING NUM_VUELO 'IB%'
        AND SUM(PLAZAS_LIBRES)>150
```

TRATAMIENTO DE NULOS

Operaciones aritméticas

Cualquier operación aritmética sobre un campo nulo nos devolverá como resultado un valor nulo.
Tomemos como ejemplo la siguiente tabla:

NULOS

COL_A	COL_B
15	10
35	35
140	NULL
NULL	100
NULL	NULL
7	110
33	60
NULL	NULL
NULL	NULL

```
SELECT COL_A+COL_B
        FROM NULOS
```

```
COL_A+COL_B
25
70
NULL
NULL
NULL
117
93
NULL
NULL
```

Funciones de columna

Ignoran los campos NULL, exceptuando la función COUNT.

```
SELECT AVG(COL_A)
SELECT SUM(COL_A)/COUNT(*)
```

```
AVG(COL_A)=46
SUM(COL_A)/COUNT(*)=25.5
```

Comparaciones

Dos valores nulos no son iguales ni son distintos, sino indeterminados.

```
SELECT *
        FROM NULOS
        WHERE COL_A=COL_B
```

```
COL_A COL_B
35      35
```

```
SELECT *
        FROM NULOS
        WHERE COL_A<>COL_B
```

```
COL_A COL_B
15      10
140     NULL
NULL    100
7       110
33      60
```

```
SELECT *
        FROM NULOS
        WHERE COL_A IS NULL
```

Esta orden visualiza todas las filas en las que el campo perteneciente a la columna COL_A es nulo.

Ordenación

Dependiendo del sistema gestor en uso los valores nulos serán los de mayor o los de menor peso.

DB/2 de IBM:

NULL ® Mayor peso. En ordenación ascendente serán los últimos.

```
SELECT COL_A
        FROM NULOS
        ORDER BY COL_A
```

```
COL_A
7
15
33
35
140
NULL
NULL
NULL
NULL
```

SQL-SERVER:

NULL ® Menor peso. En ordenación ascendente serán los primeros.

```
SELECT COL_A
        FROM NULOS
        ORDER BY COL_A
```

```
COL_A
NULL
NULL
NULL
NULL
7
15
33
33
140
```

DISTINCT

No elimina los valores nulos repetidos.

```
SELECT DISTINCT COL_A
        FROM NULOS
```

```
COL_A
15
35
140
NULL
NULL
7
33
NULL
NULL
```

Índices únicos

Sobre una columna de índice único sólo está permitida la existencia de un valor nulo.

```
CREATE UNIQUE INDEX IXNULOS
        ON NULOS
        (COL_A)
```

Devolvería un error, ya que existe más de un campo con NULL.
Para este caso los nulos se interpretan como valores iguales.

GROUP BY

Todos los nulos quedarán agrupados en el mismo grupo.

```
SELECT COL_A, COUNT(*)
        FROM NULOS
        GROUP BY COL_A
```

```
COL_A   COUNT(*)
15              1
35              1
140             1
NULL            4
7               1
33              1
```

Todos los valores NULL se agrupan y COUNT devuelve el número de filas que tenían NULL en COL_A.

SUBSELECT

Responde a la siguiente sintaxis:

```
SELECT <lista_columnas>
        FROM <lista_tablas>
        WHERE <nombre_columna> <CONCATENADOR> (SELECT <nombre_columna>
                                                FROM <lista_tablas>
                                                WHERE <Predicado>)
<CONCATENADOR>
```

Puede ser un operador de comparación o la claúsula IN
Operadores de comparación: >,<,>=,<=,=,<>

Restricciones: ha de exigirse que el resultado de la Subselect sea un único valor al usar como concatenador un operador de comparación. Si usamos IN puede devolver más de un valor.

Cada Select se ejecuta una única vez, desde la más interna, hasta la más externa.

EJ: Se desea recuperar las plazas libres que hay en cada vuelo MADRID-LONDRES del día 20/02/92.

{Las plazas libres es un campo de la tabla de reservas. En la tabla de vuelos tenemos el origen y el destino de cada vuelo.}

```
SELECT *
        FROM RESERVAS
        WHERE FECHA_SALIDA='20.02.1992'
        AND NUM_VUELO IN(SELECT NUM_VUELO
                                FROM VUELOS
                                WHERE ORIGEN='MADRID'
                                AND DESTINO='LONDRES')
```

ANY, ALL

Se usan para poder utilizar operadores de comparación con subselects que nos devuelvan más de un valor único como resultado.

```
SELECT <lista_columna>
        FROM <lista_tablas>
        WHERE <nombre_columna> <CONCATENADOR> {ANY/ALL} (<Subselect>)
```

Una expresión ANY es cierta si lo es para algún valor de los que devuelve la Subselect.
Una expresión ALL es cierta si lo es para todos los valores que devuelve la Subselect.

3>ANY(2,5,7) ® Cierto
3=ANY(2,5,7) ® Falso
3>ALL(2,5,7) ® Falso
3<ALL(9,10,11) ® Cierto

EJ: Se quieren recuperar los aviones cuya longitud sea mayor que la envergadura de todos ellos.

```
SELECT *
        FROM AVIONES
        WHERE LONGITUD > ALL (SELECT ENVERGADURA
                                FROM AVIONES)
```

O también

```
SELECT *
        FROM AVIONES
        WHERE LONGITUD > (SELECT MAX(ENVERGADURA)
                                FROM AVIONES)
```

=ANY e IN tienen la misma función.

3=ANY(2,3,5) y 3 IN (2,3,5) devuelven ambos Cierto.

Subselects correlacionadas

Son un tipo especial de subselect. La sintaxis es similar:

```
SELECT <lista_columnas>
        FROM <nombre_tabla_externa>
        WHERE <nombre_columna> <CONCATENADOR> (SELECT <nombre_columna>
                                                FROM <nombre_tabla_interna>
                                                WHERE <Predicado>)
```

En <Predicado> habrá una sentencia del tipo

<nombre_columna_tabla_interna> <operador> <nombre_columna_tabla_externa>

Las formas de ejecutar una subselect ordinaria y unas correlacionadas son diferentes. Las subselects correlacionadas obedecen al siguiente algoritmo:

ALGORITMO Subselect_Correlacionada

1. Seleccionar fila de tabla externa
2. Ejecutar SELECT interno
3. Evaluar la condición del WHERE externo
   - Cierto: la fila seleccionada en 1 será una fila de salida
4. Si existe alguna fila más en la tabla externa ir al paso 1

EJ: Se desea recuperar las reservas cuyo número de plazas libres sea mayor que la media para ese mismo vuelo.

```
SELECT *
        FROM RESERVAS, A
        WHERE PLAZAS_LIBRES > (SELECT AVG(PLAZAS_LIBRES)
                                FROM RESERVAS
                                WHERE NUM_VUELO=A.NUM_VUELO)
```

RESERVAS

NUM_VUELO	FECHA_SALIDA	PLAZAS_LIBRES
IB740	20.02.92	5
IB740	25.02.92	15
IB740	03.03.92	10

AVG(PLAZAS_LIBRES)=10

Alias

Es un sobrenombre que se le da a una tabla y que debe ser único para toda la consulta. Se escribe dejando un blanco detrás del nombre de la tabla a la cual se va a calificar.

EJ: Se quiere recuperar los aviones que tienen menos de 1 hora y cuarto de recorrido como término medio.

VUELOS

NUM_VUELO	ORIGEN	DESTINO	DISTANCIA
B747	MADRID	LONDRES	10000
B747	MADRID	PARIS	4000

AVIONES

NUM_VUELO	VELO_CRUC
.....	.....

v=e/t, t>e/v, 1.25>e/v, v*1.25 > AVG(DISTANCIA)

SELECT AVIONES SELECT VUELOS

```
SELECT *
        FROM AVIONES
        WHERE 1.25*VELO_CRUC > (SELECT AVG(DISTANCIA)
                                FROM VUELOS
                                WHERE NUM_VUELO=AVIONES.NUM_VUELO)
```

EXISTS-NOT EXISTS

Se define para comprobar la existencia o ausencia del valor devuelto por una Subselect. Una expresión con EXIST devuelve Cierto si la Subselect nos devuelve al menos un valor.

WHERE EXISTS (<Subselect>) ¬ Cierto

EJ: Seleccionar toda la información de vuelos para aquellos que tengan origen Madrid y en los que queden plazas libres.

```
SELECT *
        FROM VUELOS
        WHERE ORIGEN='MADRID'
        AND EXISTS (SELECT *
                        FROM RESERVAS
                        WHERE PLAZAS_LIBRES > 0
                        AND NUM_VUELO=VUELOS.NUM_VUELO)
```

EJ: Obtener los tipos de avión y capacidades para aquellos en los que queden menos de 30 plazas libres (JOIN).

ORDER BY

Se define para ordenar la salida de una consulta por los campos que se especifiquen a continuación.

Sintaxis:

```
        SELECT
                FROM
                WHERE
                GROUP BY
                HAVING
                ORDER BY
```

ORDER BY <especificación de columna><orden>{,<especificación de columna><orden>}

<especificación de columna>=<nombre de columna>|<posición de columna>
<orden>=ASC|DESC

Ej: Obtener el número de plazas libres que quedan para cada vuelo y ordenar el resultado de más a menos plazas libres. Para igual número de plazas ordénese por número de vuelo.

```
SELECT NUM_VUELO, SUM(PLAZAS-LIBRES)
        FROM RESERVAS
        GROUP BY NUM_VUELO
        ORDER BY 2 DESC, NUM_VUELO
```

UNION-UNION ALL

Se define para recuperar, usando una única consulta, información que se obtiene a partir de más de una consulta.

Sintaxis:

```
<SELECT>
        UNION [ALL]
        <SELECT>
                {UNION[ALL]
                <SELECT>}
```

Características:

Cada SELECT devuelve un conjunto de filas. La unión será la tabla resultado.

Condiciones de cada estructura SELECT:

- Todas deben ser iguales o compatibles una a una. Esto supone que por cada columna tengamos un único tipo de dato.

- Pueden ser completas (WHERE, GROUP BY,...), exceptuando la cláusula ORDER BY, que se ubicará al final de la última SELECT.

UNION sin ALL proporciona un resultado sin filas duplicadas.

Ej: Sacar una lista de todas aquellas ciudades para las que haya vuelo, ordenadas alfabéticamente.

```
SELECT ORIGEN
        FROM VUELOS
        UNION
                SELECT DESTINO
                        FROM VUELOS
                        ORDER BY 1
```

Catálogo del sistema o diccionario de datos:

Es el alma de un sistema gestor. Se define como un conjunto de tablas que forman una base de datos, y son definidas y mantenidas automáticamente por el sistema gestor. Sirven para almacenar información sobre los objetos definidos por los usuarios.

```
SELECT *
        FROM VUELOS
```

1- El sistema busca en el catálogo si existe la tabla VUELOS.
2- Verifica si el usuario tiene acceso a esa información.
3- Se pregunta cuáles y cuántas columnas tiene la tabla VUELOS.

DB2 IBM:

SYSTABLES: una fila por cada tabla definida en la instalación.
SYSCOLUMNS: una fila por cada columna definida.
SYSINDEXES: una fila por cada índice definido.
SYSVIEW: una fila por cada vista.
SYSTABAUTH: una fila por cada autorización definida.

Todas las tablas son directamente consultadas por usuarios autorizados.
ADM (Administrador): es la persona que concede autorizaciones a los usuarios.
Un usuario autorizado puede efectuar operaciones del tipo:

```
SELECT *
        FROM SYSTABLES
        WHERE NAME='RESERVAS'
```

NAME	DBNAME	CARD
RESERVAS	AEROPUERTO	.....

DBNAME: nombre de la BdD a la que pertenece la tabla.
CARD: nº de filas de la tabla.
OWNER: usuario creador de la tabla.

```
SELECT *
        FROM SYSCOLUMNS
        WHERE DBNAME='RESERVAS'
```

Nos da la información sobre todas las columnas que pertenecen a la tabla reservas.

NAME	TBNAME	COL_NO	COL_TYPE	LENGTH	NULLS
NUM_VUELO	RESERVAS	1	CHAR	6	N
FECHA_SALIDA	RESERVAS	2	DATE	8	N

TBNAME: nombre de la tabla.
COL_NO: posición de la columna en la tabla.
COL_TYPE: tipo de dato
LENGTH: longitud del dato de la columna.
NULLS: indica si se permite valor nulo.

Ej: Obténgase la última hora de salida para cada destino de los vuelos realizados por aviones capaces de almacenar más combustible que un tercio de la media que pueden almacenar los demás aviones.

```
SELECT DESTINO, MAX(HORA_SALIDA)
        FROM VUELOS
        WHERE TIPO_AVION IN (SELECT TIPO
                                    FROM A
                                    WHERE COMBUSTIBLE>1/3*(SELECT AVG(COMBUSTIBLE)
                                                                FROM AVIONES
                                                                WHERE TIPO<>A.TIPO)
```

Ej: Crear una vista sobre la tabla vuelos con las columnas ORIGEN y DESTINO para aquellos vuelos que no sean de IBERIA. Visualizar el contenido de la lista para los vuelos que no partan de Madrid. Borrar la vista.

```
CREATE VIEW V_VUELOS
(V_ORIGEN, V_DESTINO)
AS SELECT ORIGEN, DESTINO
        FROM VUELOS
        WHERE NUM_VUELO NOT LIKE 'IB%'

SELECT *
        FROM V_VUELOS
        WHERE V_ORIGEN<>'MADRID'

DROP VIEW V_VUELOS
```

Ej: Visualice los tipos de avión, el doble de su longitud y la mitad de su envergadura, para aquellos aviones con envergadura mayor que la media y que realizan vuelos desde o hacia Barcelona, ordenándolos de mayor a menor longitud.

```
SELECT TIPO_AVION, 2*LONGITUD, .5*ENVERGADURA
        FROM AVIONES, VUELOS
        WHERE ENVERGADURA>(SELECT AVG(ENVERGADURA)
                                    FROM AVIONES)
        AND
                AVIONES.TIPO_AVION=VUELOS.TIPO_AVION
        AND
                (ORIGEN='BARCELONA' OR DESTINO='BARCELONA)
        ORDER BY 2 DESC
```

Ej: Visualice las tres primeras letras de los orígenes y destinos de los vuelos realizados por aviones con longitud mayor con longitud mayor que la media y envergadura menor que 2/3 de la máxima envergadura, ordenados alfabéticamente por destino.

SUBSTRING (SQL)
(SUBSTRNG), (DB2)

SUBSTRING (string, posición, nºcaracteres)

nom_col / cadena con comillas (“)

```
SELECT SUBSTRING (ORIGEN, 1, 3), SUBSTRING (DESTINO, 1, 3)
        FROM VUELOS
        WHERE TIPO_AVION IN (SELECT TIPO
                                    FROM AVIONES
                                    WHERE LONGITUD > (SELECT AVG(LONGITUD)
                                                        FROM AVIONES)
                                    AND ENVERGADURA*3/2 < (SELECT MAX(ENVERGADURA)
                                                        FROM AVIONES)
                                    ORDER BY 2
```

Ej: Visualice el total de plazas libres por número de vuelo para aquellos realizados desde Madrid a Barcelona o Sevilla y que recorran una distancia mayor que la media de todos los vuelos que salen de Madrid, ordenándolos de menor a mayor.

```
SELECT SUM(PLAZAS_LIBRES), NUM_VUELO
        FROM RESERVAS, VUELOS
        WHERE RESERVAS, NUM_VUELO=VUELOS.NUM_VUELO
        AND ORIGEN='MADRID'
        AND DESTINO IN ('BARCELONA', 'SEVILLA')
        AND DISTANCIA > (SELECT AVG(DISTANCIA)
                                FROM VUELOS
                                WHERE ORIGEN='MADRID')
        ORDER BY 1
```

Ej: Obtener para cada número de vuelo el total de plazas libres de los vuelos que recorran distancias menores que 2/3 de la media de las distancias recorridas por vuelos de otras compañías.

```
SELECT NUM_VUELOS, SUM(PLAZAS_LIBRES)
        FROM RESERVAS, VUELOS ¬ V
        WHERE RESERVAS.NUM_VUELO=VUELOS.NUM_VUELOS
        AND DISTANCIA*3/2 < (SELECT AVG(DISTANCIA)
                                        FROM VUELOS
                                        WHERE SUBSTRING (NUM_VUELO, 1, 2) <>
                                                SUBSTRING (VUELOS.NUM_VUELO, 1, 2)
```

TEORIA DE LA NORMALIZACION

Introducción:

Nos basaremos en la siguiente tabla:

AUTORES-LIBROS				
NOMBRE	NACION	COD_LIB	TITULO	EDITOR
Date	USA	999	IBD	AW
Ad.Mig.	ESP	888	CyD	RM
Ma.Piat.	ITA	888	CyD	RM
Date	USA	777	BdD	AW

Bibliografía: Diseño y Gestión de Bases de Datos. Angle Lucas.

Se plantean una serie de problemas:

Redundancia: cuando un autor tiene varios libros, se repite la nacionalidad.

Anomalías de modificación: si Ad.Mig. y Ma.Piat. desean cambiar de editor, se modifica en los 2 lugares. A priori no podemos saber cuántos autores tiene un libro. Los errores son frecuentes al olvidar la modificación de un autor. Se pretende modificar en un sólo sitio.

Anomalías de inserción: se desea dar de alta un autor sin libros, en un principio. NOMBRE y COD_LIB son campos clave, una clave no puede tomar valores nulos.

Teoría de la normalización: la teoría de la normalización ofrece una serie de reglas para efectuar una modelización óptima.

La tabla anterior debería dividirse en 3 tablas:

AUTORES (NOMBRE, NACION)
LIBROS (COD_LIB, TITULO, EDITOR)
ESCRIBE (NOMBRE, COD_LIB)

En los años 70 Codd creó las bases de la teoría de la normalización. A cada regla de la teoría la denominó forma normal. Codd creó las formas normales 1ª, 2ª y 3ª. La 3ª forma normal originó problemas. Boyce ayudo a solventarlos con la f.n. de Boyce-Codd (FNBC). A finales de los 70 Fagin creó las formas normales 4ª y 5ª.

Las formas normales se basan en el concepto de dependencia, que comprende las restricciones definidas sobre los atributos de una relación. Existen diferentes tipos de dependencia:

- Dependencias funcionales (Formas normales 1ª, 2ª y 3ª y FNBC)
- Dependencias multivaluadas (4ª forma normal)
- Dependencia de JOIN (5ª forma normal)

Formas normales

1ª forma normal: es una restricción inherente del modelo relacional. Se dice que una tabla está en 1ª forma normal si no existen en ella grupos repetitivos.

Una tabla no puede tener en un campo más de un valor.

TITULO	AUTOR
CyD	Ad.Mig.
	Ma.Piat.

Hay un grupo repetitivo. De este modo la tabla no es plana y no está en 1ª forma normal. Para convertirla a 1ª forma normal:

TITULO	AUTOR
CyD	Ad.Mig.
CyD	Ma.Piat.

2ª forma normal: partimos de la idea de dependencia funcional: un atributo o conjunto de atributos B depende funcionalmente de A sólo si a cada valor de A le corresponde un único valor de B:

A ® B <=> a cada valor de A le corresponde un único valor de B

A	B
x1	y1
x2	y2
x3	y3

Ej: DNI depende funcionalmente de NOMBRE y NOMBRE de DNI

DNI ® NOMBRE
NOMBRE ® DNI

NOMBRE ¬® DNI

Ej: DIRECCION depende funcionalmente de DNI, pero DNI no depende funcionalmente de DIRECCION

DNI ® DIRECCION

DIRECCION -/® DNI

Ej: TITULO, LIBRO no dependen funcionalmente de DNI, AUTOR, porque un autor puede escribir varios libros

DNI, AUTOR -/® TITULO, LIBRO

Ej: Se tiene una base de datos de pluriempleados:

Atributos: DNI, EMPRESA, SUELDO

DNI -/® EMPRESA
DNI -/® SUELDO

Se puede concatenar atributos, obteniendo:

DNI, EMPRESA ® SUELDO

Sueldo es el atributo implicado que depende de DNI y EMPRESA juntos, que son atributos implicantes. También:

DNI ® NOMBRE, DIRECCION

Las 3 primeras formas normales más la forma normal de Boyce-Codd se basan en dependencias funcionales obedeciendo al siguiente teorema:

Dada una relación R con un conjunto de atributos A que cumple R(A), x®y, es posible una descomposición en dos tablas de la siguiente manera:

R(A), x®y

R(x,y) es una relación compuesta por los atributos que forman la dependencia funcional
R(A-y) es una relación compuesta por los atributos de R excluyendo el atributo implicado

R(A)=R(x, y) R(A-y) el JOIN de ambas forma la relación original.

Las relaciones a partir de ahora se definirán como un conjunto de atributos con dependencias funcionales R(A, DF).

Para normalizar la tabla habrá que conocer todas las dependencias funcionales, pero en la relación que nos den sólo tendremos algunas, a partir de las cuales podremos hallar el resto.

Aplicaremos las propiedades de las dependencias funcionales para obtener todo el conjunto de posibles dependencias funcionales que puedan existir en la relación.

Al conjunto inicial de dependencias funcionales lo llamaremos F, conjunto a partir del cual obtendremos el resto de dependencias funcionales. A cada nueva dependencia funcional obtenida a partir de F la llamaré f. Al nuevo conjunto que contenga todas las dependencias funcionales que obtenga le llamaré F+. Una vez hallado F+ podré aplicar las formas normales de la teoría de la normalización.

Dependencia funcional derivada

Dado un conjunto F de dependencias funcionales se dice que f deriva de F (F f) si f se obtiene a partir de F.

Cierre de un conjunto de dependencias funcionales

Se define Cierre (F+) como el conjunto de todas las dependencias funcionales implicadas por F o halladas a partir de F.

Propiedades de las dependencias funcionales

a) Axiomas
b) Propiedades propiamente dichas

a) Axiomas

a.1) Axioma reflexivo

Si Y está incluido en X entonces X ® Y (Si Y c X => X ® Y)

Ej: CODPROV c CODPOSTAL

CODPOSTAL ® CODPROV

A un código postal le corresponde un único código de provincia.

a.2) Aumentatividad

Si X ® Y y Z c W => XW ® YZ

z Se demuestra del siguiente modo: Z c W equivale a W ® Z. Si tenemos X ® Y y W ® Z podemos afirmar que XW ® YZ

a.3) Transitividad

Si X ® Y y Y ® Z => X ® Z

b) Propiedades propiamente dichas

b.1) Unión

X ® Y y X ® Z => X ® YZ

Demostración:

Si X ® Y (aumentatividad con X) => X ® XY
Si X ® Z (aumentatividad con Y) => XY ® XZ
Si X ® XY y XY ® YZ (transitividad) => X ® YZ

b.2) Pseudotransitividad

X ® Y y WY ® Z => WX ® Z

Demostración:

Si X ® Y (aumentatividad con W) => WX ® WY
Si WX ® WY y WY ® Z (transitividad) => WX ® Z

b.3) Descomposición

X ® Y y Z c Y => X ® Z

Demostración:

Si Z c Y (axioma reflexivo) => Y ® Z
Si X ® Y y Y ® Z (transitividad) => X ® Z

Dependencia funcional total

El conjunto de atributos Y tiene dependencia funcional total con X si Y tiene dependencia funcional con X (X ® Y) y además no existe ningún subconjunto Z de X (Z c X) con el cual Y tenga dependencia funcional (Z -/® Y).

Diagramas de dependencias funcionales

Son una herramienta que sirve para tener una visión general de los datos y de las dependencias funcionales entre ellos. S e representa en forma de grafo con los implicantes de las dependencias funcionales en un rectángulo, de los que salen flechas hacia los implicados.

Ej: Dado: ABC ® MNS
M ® N
BC ® OPR
O ® P
C ® Q , obtener el diagrama de dependencias funcionales.

A C AB ® C

B B ® C

Es una dependencia funcional total.

Ej: Hallar si las siguientes dependencias funcionales son totales:

a) DNI, EMPRESA ® SUELDO

b) DNI, EMPRESA ® NOMBRE

SUPERCLAVE Y CLAVE

Superclave (SK): es el atributo o conjunto de atributos tales que en una relación R(A, DF) se cumple que SK ® A. SK es el implicante capaz de implicar a la tabla completa.

Ej: En una tabla compuesta por X, Y, Z y W, si W es superclave se cumple que:

W ® X
W ® Y
W ® Z

Clave (K): es el atributo o conjunto de atributos tales que en una relación R(A, DF) es superclave y además no existe ningún subconjunto K' c K tal que K' ® A

K, R(A, DF), K _ SK /\ no existe K' c K / K' ® A

Debe tener una dependencia funcional total con los atributos de la tabla. Si tenemos:

WZ ® A
WZ ® B
WZ ® C

pero

Z -/® A
Z -/® B
Z -/® C

La clave es la mínima superclave, no descomponible en claves menores.

2ª forma normal: se dice que una relación está en 2ª forma normal si cumple las siguientes condiciones:

1) Está en 1ª forma normal
2) Cada atributo no principal o secundario (no forma parte de la clave), tiene una dependencia funcional total con la clave.

Ej: AB ® C
B ® D

A C

B D

D depende funcionalmente de B sólo y debería depender de AB para estar en 2ª forma normal. Para convertirlo en 2ª forma normal se descompone en 2 tablas:

Tabla 1: clave con dependencias totales

Tabla 2: parte de la clave implicante con dependencias parciales

A
C B D
B

Esto cumple el teorema 1º enunciado en el capítulo (Descomposición por JOIN)

Ej: Pasar a 2ª forma normal la siguiente tabla:

DNI NOMBRE

EMPLEADO SUELDO

DNI

SUELDO DNI NOMBRE EMPLEADO

Dependencia funcional transitiva: se cumple si:

A ® B
B -/® A => A ® C
B ® C

Gráficamente:

3ª forma normal: se dice que una tabla está en 3ª forma normal si está en 2ª forma normal y además cumple que ningún atributo no principal depende transitivamente de la clave.

Ej: Pasar a 3ª forma normal:

A ® B	B ® A	C ® D
A ® C	B ® C	C ® E
A ® D	B ® D	
A ® E	B ® E	

A y B son claves candidatas a principales. Elegimos una de las 2, por ejemplo A.

A → B, C, D, E

A → B, C → D, B → E

No está en 3ª forma normal porque existen atributos no principales que dependen transitivamente de la clave de la relación.

Para pasarlo a 3ª forma normal lo descompongo en 2 tablas:

1ª tabla: clave con dependencias no transitivas.

B
A
C

2ª tabla: clave con dependencias transitivas.

D
C
E

Ej: Tenemos las siguientes dependencias de la tabla ALUMNOS:

NºMATRICULA ® AULA, GRUPO
GRUPO ® AULA

Pasarlo a 3ª forma normal.

Descomponemos en 2 tablas:

1ª tabla: clave con dependencias no transitivas.

B
A
C

2ª tabla: dependencias transitivas.

D
C
E

Ej: Pasar a 3ª forma normal las siguientes dependencias de la tabla alumnos:

NºMATRICULA ® AULA, GRUPO
GRUPO ® AULA

Forma normal de Boyce-Codd: Trata de resolver los problemas que origina la 3ª forma normal. Se dice que una relación R está en FNBC sólo si todo determinante o todo implicante (conjunto de atributos a la izquierda de la relación) es clave.

Ej:

A
C
B

Está en 3ª forma normal, pero no en FNBC.

AB ® C
C ® B
AB ® B

Para pasar a FNBC una relación R en la cual existe una dependencia del tipo X ® Y siendo X un atributo no principal y siendo Y un atributo principal, descomponemos R en 2 proyecciones:

R1 formada por los atributos X e Y ® R1=(X, Y)
R2 formada por todos los atributos de R exceptuando Y ® R2=(A - Y)

Obtenemos:

C B

A

C

Ej: Tenemos una tabla de un callejero:

CALLEJERO (DIRECCION, CIUDAD, C_POSTAL)

C_POSTAL ® CIUDAD
DIRECCION, CIUDAD ® C_POSTAL

C_POSTAL CIUDAD
DIRECCION
C_POSTAL
CIUDAD DIRECCION

C_POSTAL

Dependencia multivaluada:

Sean A y B dos subconjuntos distintos de atributos de una tabla T se dice que A tiene una dependencia multivaluada con B ó que A multidetermina a B ó que B depende multivaluadamente de A (A ®® B) así para cada valor de A tenemos un conjunto, bien sea de valores de B que son independientes de los demás atributos, o la relación.

1. A <= B
2. Independientemente del resto de atributos de A

Ej:

Los profesores de una facultad imparten varias asignaturas y una asignatura es impartida por varios profesores.

Una asignatura tiene varios textos y un texto puede utilizarse en varias asignaturas, independientemente del profesor que las imparte.

PROFESOR	ASIGNATURA	TEXTO
ANA	DIGITALES	T1
ANA	DIGITALES	T2
LUIS	DIGITALES	T1
LUIS	DIGITALES	T2
LUIS	COMUNICACIONES	T2
LUIS	COMUNICACIONES	T3

Asignatura ®® Profesor

a) Cada asignatura tiene definidos varios profesores
b) Se cumple

Asignatura ®® Texto

a) Cada asignatura tiene asignado más de un texto
b) Se cumple

Siempre que se dé una dependencia X ®® Y tiene que darse una dependencia X ®® A - (X U Y).
Para que se dé debe tener más de 2 atributos.

Profesor ®/® Texto

a) Para cada profesor hay definidos más de un texto
b) Depende de Asignatura

Profesor ®® Asignatura

a) Un profesor tiene asignadas varias asignaturas
b) No se puede dar por una serie de teoremas matemáticos

4ª Forma Normal: una tabla está en 4ªFN si está en 3ªFN y se cumple que las únicas dependencias multivaluadas existentes son las existentes con los atributos secundarios.

Cuando no existen dependencias multivaluadas y la tabla está en 3ªFN para pasar a 4ªFN tendremos en cuenta el teorema de FAGIN: "Una tabla T con los atributos A, B, C se puede descomponer sin pérdida de información en 2 proyecciones: T1 con los atributos A y B y T2 con los atributos A y C, sólo si A multidetermina a B y C (A ®® B / C).

Para pasar a 4ªFN si existe una dependencia multivaluada X ®® Y la dividimos en 2 tablas.

1. R1 (x, y)
2. R2 (A - y)

R: x ®® y

R1 (x, y)

R2 (A - y)

Ej:

Asignatura ®® Texto
Asignatura ®® Profesor

(asignatura, texto) (asignatura, profesor)

R (Asignatura, Texto, Profesor)

1. Asignatura ®® Texto, x
2. Asignatura ®® Profesor, y

R1 (Asignatura, Texto, x)
R2 (Asignatura, Profesor, y)

Dependencia de JOIN

Se dice que una relación T formada por los atributos A1, A2, ..., An tiene una dependencia con sus proyecciones T1, T2, ..., Tn si T=T1 T2 T3 ... Tn

T	A	B	C	T1	A	B	T2	B	C
	a1	b1	c1		a1	b1		b1	c1
	a2	b1	c1		a2	b1		b2	c1
	a1	b2	c1		a1	b2		b1	c2
	a1	b1	c2						

T<> T1 T2

T1 T2	A	B	C
	a1	b1	c1
	a1	b1	c2
	a2	b1	c1
	a2	b1	c2
	a1	b2	c1

La 4ª fila (a2, b1, c2) es una tupla intrusa o espuria.
Hay otra proyección que hace que se cumpla:

T3	A	C
	a1	c1
	a2	c1
	a1	c2

T=T1 T2 T3

5ª Forma Normal: se dice que una tabla está en 5ªFN si está en 4ªFN y además toda dependencia de JOIN está implicada por las claves de la tabla. Las columnas de enlace deben ser los atributos que componen la clave.

En la siguiente tabla no existen dependencias funcionales. Bebida -/® Camarero.

T	Clientes	Bebidas	Camarero
	López	Cerveza	Juan
	López	Fanta	Luis
	Garcia	Fanta	Juan
	Pérez	Cerveza	Juan

Dependencias de JOIN:

T=T1 T2 T3

Columna enlace: T1 T2 = Cliente

Los tres atributos forman la clave. T no está en 5ª forma normal. Si las columnas de enlace son las columnas de la clave entonces está en 5ª forma normal. Para pasarlo a 5ª forma normal habrá que descomponer T en sus proyecciones.

EMPLEADOS (DNI, NOMBRE, DIRECCION, NSS, FISS, CATEGORIA)

T1 (DNI, NOMBRE, DIRECCION)
T2 (DNI, NSS, FISS)
T3 (DNI, CATEGORIA)

EMPLEADO = T1 t2 t3

T está en 5ª forma normal, aún así es factible descomponerla en sus proyecciones.

Ej: Competiciones

En una prueba hay varios árbitros.
Un árbitro puede arbitrar en diferentes pruebas.
Un atleta puede competir en diferentes pruebas.
En un único país no pueden existir nº de pasaportes iguales, pero sí si son de países diferentes.
Pueden existir distintos atletas con el mismo nombre.

Atributos:

NPa:	Nº de pasaporte del atleta
Na:	Nombre del atleta
da:	Dirección del atleta
dpa:	Dirección postal del atleta
ca:	Ciudad del atleta
Pra:	Provincia del atleta
Pa:	País del atleta
cpa:	Código de país del atleta
Fn:	Fecha de nacimiento del atleta
Sa:	Sexo del atleta
cp:	Código de la prueba
np:	Nombre de la prueba
ma:	Marca del atleta
NPar:	Nº de pasaporte del árbitro
Nar:	Nombre del árbitro

Ej: Barco de pasajeros

Un barco pertenece a un propietario.
Se construye en un sólo astillero.
El código es único dentro del país.
Pueden existir barcos diferentes con el mismo código si son de distintos países.
Un barco es de un sólo tipo.
Los marineros del barco sólo tienen una dirección y un código dentro del barco.
Un marinero puede tener varios teléfonos.
No existen ciudades repetidas.
Dos pasajeros pueden tener el mismo camarote en un barco.
Un pasajero puede embarcar en varios barcos en distintas fechas.
Dos barcos pueden haber tenido a un mismo marinero, pero nunca al mismo tiempo.

Ej: Coleccionistas de sellos

El código de sello particular es único para la colección del propietario.
El código internacional es único para cada tipo de sello emitido, del cual pueden existir varios ejemplares.
Dos coleccionistas pueden tener ejemplares de un mismo tipo de sello, pero cada uno tendrá un código particular del sello que podría ser el mismo.
No existen 2 DNIs, ciudades, provincias o países iguales.
Pueden existir nombres iguales.
Un propietario pude tener varios teléfonos.

Atributos:

CS: Código particular del sello para el propietario actual
CI: Código internacional del sello
PS: País del sello
CP: Código del país
VE: Valor de emisión del sello
AE: Año de emisión del sello
EP: Estado político del país en el año de emisión del sello
CE: Código del estado político
CLS: Clase de sello
CCL: Código de la clase de sello
EC: Estado de conservación del sello

CC: Código del estado de conservación del sello
DNIP: DNI del propietario del sello
NP: Nombre del propietario del sello
DP: Dirección del propietario del sello
CiP: Ciudad del propietario del sello
CPP: Código postal del propietario del sello
PP: Provincia del propietario del sello
PaP: País del propietario del sello
TP: Teléfono del propietario del sello
DNIA: DNI del antiguo propietario del sello

SQL Embedido

1. SQL autocontenido
2. SQL embedido

SQL embedido:

Sentencia de SQL que se utiliza dentro de un programa llamado anfitrión, escrito en cualquier lenguaje.
Tendremos tablas con datos de entrada y de salida.
Las sentencias de SQL serán sentencias embebidas en el programa anfitrión.

Características:

* Todas las sentencias SQL van a estar enmarcadas por:

```
EXEC SQL
        <sentencias>
END-EXEC
```

* Antes de utilizar un compilador para manejar SQL embebido es necesario pasar el programa fuente por un precompilador:

PROG FUENTE → precompilador → PROG FUENTE MODIFICADO → compilador → PROG OBJETO → LINKER

- ® sintaxis sentencias SQL
- ® comentar sentencias SQL (para que las ignore el compilador)
- ® sustituir sentencias SQL por llamadas a rutinas de librerías

* Manejo de variables de programa dentro de sentencias SQL:

Se pone dos puntos (:) delante del nombre de la variable.

Ej: Para un vuelo, visualizar plazas libres y nº de vuelo (datos de salida), sabiendo el origen, destino, hora de salida y fecha de salida (datos de entrada).

```
INICIO
        escribir 'Introducir ORIGEN, DESTINO, FECHA, HORA'
        leer ORIGEN, DESTINO, FECHA, HORA
        EXEC SQL
                SELECT RESERVAS.NUM_VUELO, PLAZAS_LIBRES
                INTO :VUELO, :PLAZAS
                FROM RESERVAS, VUELOS
                WHERE RESERVAS.NUM_VUELO=VUELOS.NUM_VUELO
                AND ORIGEN=:ORIGEN
                AND DESTINO=:DESTINO
                AND FECHA=:FECHA
                AND HORA=:HORA
        END-EXEC
        escribir VUELO, PLAZAS
FIN
```

INTO sirve para determinar las variables de salida.
Devuelve en la salida una sola fila. En caso de que la salida devuelva más de una fila tendremos los cursores.

Cursor:

- Es una estructura de datos tabular (en forma de tabla) que sirve para almacenar un número indeterminado de filas.
- Sólo permite manejar una única fila a la vez, mediante un puntero.
- Para manejar cursores en SQL embebido hay 4 etapas:

1) Definición del cursor

Se define junto a la declaración de variables del programa anfitrión:

```
EXEC SQL
        DECLARE
                <nom_cursor> CURSOR
        FOR
                <sent_select>
END-EXEC
```

2) Abrir cursor

Rellena de filas la estructura del cursor:

```
EXEC SQL
        OPEN <nom_cursor>
END-EXEC
```

3) Recuperar filas (1)

```
EXEC SQL
        FETCH <nom_cursor> {avanza el puntero una posición}
        INTO <lista_variables> {una sola variable por columna}
END-EXEC
```

Cuando se hace un OPEN se rellena la estructura y el puntero se posiciona anterior a la primera fila. El primer FETCH posiciona el cursor en la primera fila.

4) Cerrar cursor

```
EXEC SQL
        CLOSE <nom_cursor> {liberando memoria}
END-EXEC
```

Ej: Realizar un programa que presente todos aquellos vuelos existentes entre un origen y un destino determinados.

Pasos:

1. Declaración de variables
2. Petición de información de origen y destino
3. Abrir cursor
4. Recuperar primera fila del cursor
5. Comprobar si existe alguna fila. Si no existe ir a 9
6. Escribir fila en pantalla
7. Recuperar siguiente fila
8. Ir a 5
9. Fin

```
VARIABLES
        ORIGEN, DESTINO, HORA, VUELO: STRING

EXEC SQL
        DECLARE lista_vuelos CURSOR
        FOR SELECT NUM_VUELO, HORA_SALIDA
                FROM VUELOS
                WHERE ORIGEN=:ORIGEN
                        AND DESTINO=:DESTINO
END-EXEC

INICIO
        ESCRIBIR 'INTRODUZCA ORIGEN Y DESTINO'
        LEER ORIGEN, DESTINO
        EXEC SQL
                OPEN lista_vuelos
        END-EXEC
        EXEC SQL
                FETCH lista_vuelos
                INTO :VUELO, :HORA
        END-EXEC

{Variable predefinida SQL-CODE: nos da información sobre la ejecución de cada sentencia en SQL:
        SQL-CODE < 0 ® Error
        SQL-CODE = 100 ® CURSOR vacío
        SQL-CODE > 0 ® Warning}

        MIENTRAS SQL-CODE <> 100
                ESCRIBIR VUELO, HORA
                EXEC SQL
                        FETCH lista_vuelos
                        INTO :VUELO, :HORA
                END-EXEC
        FIN_MIENTRAS
        EXEC SQL
                CLOSE lista_vuelos
        END-EXEC
FIN
```

INTEGRIDAD E INTEGRIDAD REFERENCIAL

1. INTRODUCCION

Integridad: característica que nos permite tener coherencia y veracidad en la información.

Existen ciertas operaciones de SQL que pueden hacer peligrar la integridad de la operación:

- Inserción
- Borrado
- Modificación

T.PADREEMPLEADO DNI Clave Principal

depende

cod_fam

T.HIJA FAMILIARES DNI_EMP Clave Ajena

Puede que borremos un empleado y olvidemos eliminar los familiares.

Tipos de integridad:

Integridad de dominio: restringimos los valores que puede tomar un atributo respecto a su dominio, por ejemplo EDAD ® 18 - 65.

Integridad de entidad: la clave primaria de una entidad no puede tener valores nulos y siempre deberá ser única, por ejemplo DNI.

Integridad referencial: las claves ajenas de una tabla hija se tienen que corresponder con la clave primaria de la tabla padre con la que se relaciona. Por ejemplo, en familiares necesitaremos el DNI de empleado, que es la clave ajena de la tabla.

2. INTEGRIDAD REFERENCIAL

Clave principal: conjunto de atributos capaz de diferenciar cada tupla de la tabla.
Clave ajena: Clave en la tabla padre.

2.1 DDL

Se basa en la definición de la clave principal y de la clave ajena de la tabla.

```
CREATE TABLE EMPLEADO
        (PRIMARY KEY DNI,
                DNI CHAR(8) NOT NULL,
                NOMBRE VARCHAR(30) NOT NULL,   {columnas de la tabla, clave incluida}
                . . .
        )

CREATE TABLE FAMILIARES
        (PRIMARY KEY COD_FAMILIAR,
                COD_FAMILIAR CHAR(4) NOT NULL,
                NOM_FAMILIAR CHAR(30) NOT NULL,
                DNI_EMPL CHAR(8) NOT NULL,
                . . .
        )

        FOREIGN KEY REL_EMPL_FAM
                DNI_EMPL REFERENCE EMPLEADO
                                RESTRICT
                ON DELETE       CASCADE
                                SET NULL
```

2.2 DML

Tendremos una serie de reglas para:

2.2.1 Inserción
2.2.2 Actualización

Implícitas o determinadas por el sistema.

2.2.3 Borrado

Explícita o definida por el usuario de tres que proporciona el sistema.

2.2.4 Regla de inserción

Se ejecuta sobre la tabla hija.
Sólo se podrá insertar una fila en la tabla hija si el valor de clave ajena de esa fila es un valor nulo o un valor de los existentes en la clave primaria de la tabla padre.

2.2.5 Regla de actualización

Se ejecuta sobre las tablas padre o hija.

Tabla padre: no se puede modificar el valor de clave primaria de la tabla padre si existe alguna fila en la tabla hija que lo referencie en la clave ajena.

Tabla hija: el valor de clave ajena de la tabla hija sólo se puede modificar si el nuevo valor que va a adoptar es un valor nulo o es igual a un valor de clave primaria de la tabla padre.

2.2.6 Regla de borrado

Esta regla se especifica al crear la tabla padre y se activará cuando se intente eliminar una fila de la tabla.

```
CREATE TABLE
. . .
FOREIGN KEY
. . .
                RESTRICT

ON DELETE    CASCADE

                SET NULL
```

Es una regla explícita, ya que el usuario puede elegir la regla de borrado que desee, a partir de 3 reglas que nos va a proporcionar el sistema.

a) RESTRICT

No se puede borrar una fila de la tabla padre si existen filas en la tabla hija cuyos valores de clave ajena sean iguales a valores de clave primaria de la fila que queremos borrar.

b) CASCADE

Cada vez que se borre una fila de la tabla padre se eliminarán aquellas filas de la tabla hija que tengan como valor de clave ajena el mismo valor que el de la clave primaria de la fila que se quiere borrar.

c) SET NULL

Cada vez que se elimine una fila de la tabla padre se actualizarán los campos de la clave ajena de la tabla hija a valores nulos para aquellas filas que tengan en el campo de la clave ajena el mismo valor que la clave primaria de la fila de la tabla padre que se quiere borrar.

Casos específicos de integridad:

Integridad auto-referencial

Una tabla es al mismo tiempo padre e hija de si misma.

Ej: Dada la tabla T_EMP con las siguientes columnas:

COD_EMP NOMBRE COD_EMP_JEFE

Todos los valores que aparezcan en la última columna existirán también en la primera, por tanto COD_EMP_JEFE se convierte en clave ajena de la tabla.

Ciclos de integridad referencial

Conjunto de tablas que funcionará como tablas padres e hijas unas de otras, formando un camino cerrado.

Ej: Tenemos 3 tablas:

COCHES (MATRICULA, MARCA, COD_DIRECTOR)

DIRECTORES (COD_DIRECTOR, NOMBRE_DIR, COD_SECRETARIA)

SECRETARIAS(COD_SECRETARIA, NOMBRE_SEC, MATRICULA)

COCHES es tabla hija de DIRECTORES
DIRECTORES es tabla hija de SECRETARIAS
SECRETARIAS es tabla hija de COCHES

COCHES

DIRECTORES SECRETARIAS

Forma un ciclo de integridad referencial.

Por haber ciclos debemos incorporar restricciones de borrado, para un correcto funcionamiento de la integridad referencial.

Restricciones de borrado

a) Integridad auto-referencial

Siempre que tengamos una tabla con integridad auto-referencial debemos definir un borrado en cascada.

Tabla EMPLEADO

COD_EMP	COD_EMP_JEFE
A00	-
B01	A00
C02	B01
D03	C02

```
DELETE FROM EMPLEADO
        WHERE COD_EMP_JEFE >= 'B01'
```

Con RESTRICT no se puede borrar la 3ª fila, porque C02, clave primaria, está siendo referenciada en la fila 4ª como clave ajena. Salta a la 4 fila y la borra.

```
DELETE FROM EMPLEADO
        WHERE COD_EMP_JEFE IS NULL
```

Con SET NULL borraría la primera fila, porque su clave ajena es NULL, pasaría a la 2ª y como su clave ajena es igual a la clave primaria de la fila borrada pondría la clave ajena a NULL. Como le hemos dicho que borre aquellas filas que tengan por clave ajena un valor nulo, ahora borraría esta fila y así sucesivamente hasta eliminar la tabla completa.

b) Ciclos de integridad referencial

```
DELETE FROM T3
        WHERE FKT2 IS NULL
```

Restricciones de borrado en ciclos:

CICLO DE 2 TABLAS: ninguna de ellas podrá tener definida la opción de borrado en cascada.
CICLO DE MAS DE 2 TABLAS: Sólo una de las tablas podrá tener la opción de borrado en cascada.

El esquema anterior no cumple las restricciones expuestas.

c) Tablas conectadas por múltiples caminos (con más de una tabla padre)

```
DELETE FROM T2
        WHERE PKT2 = 'T2A'
```

Restricciones de borrado en tablas conectadas por múltiples caminos:

Los dos caminos que llegan a una tabla deben tener definida siempre la misma regla de borrado, que debe ser RESTRICT o CASCADE.

En el ejemplo funcionaría bien por el primer camino, pero por el segundo hay restricción.

Ej. Pruébese qué ocurriría si entre T3 y T1 hay SET NULL y entre T2 y T1 hay SET NULL.

Ventajas del uso de las reglas de integridad referencial proporcionadas por el sistema frente a la implementación de éstas por parte del usuario.

1. Al usar las proporcionadas por el sistema los analistas tienen menor responsabilidad.

2. Habrá un incremento en la productividad y un decremento en los costes de la aplicación, porque el número de pruebas a realizar será menor si la integridad referencial es implementada por el propio sistema, el cual nos garantizará el correcto funcionamiento de las reglas de integridad referencial. El tiempo de desarrollo de la aplicación será menor.

3. La información será más coherente, ya que está garantizado que todas las filas de las tablas cumplen las normas de integridad referencial.

3. DISPARADORES O 'TRIGGERS'

Son un conjunto de sentencias que el sistema ejecuta a la hora de efectuar una inserción, actualización o borrado en una tabla. Para definir un TRIGGER necesitamos:

- Nombre de la tabla sobre la que actuará el trigger
- Sentencia que activará el trigger (inserción, actualización o borrado)
- Acciones que realizará el trigger

DB2 no incorpora la posibilidad de definir triggers, pero SQL sí. Las reglas de integridad referencial se emulan en SQL mediante la implementación de triggers.

Emulación de inserción en tabla hija mediante el uso de trigger en SQL:

Según la regla de integridad referencial, no se puede efectuar una inserción en una tabla hija a menos que exista un valor de clave primaria igual al de clave ajena de la fila que se desea insertar.

Definición del trigger:

```
DEFINE TRIGGER insercion
        ON INSERT OF HIJA
```

El trigger comparará el valor de clave ajena de la fila a insertar con el valor de clave primaria de la tabla hija cada vez que se intente insertar una fila.

Implementación del trigger:

```
(if (SELECT COUNT (#)
        FROM PADRE, INSERTED
        WHERE PADRE.CL_PRIMARIA = INSERTED.CL_AJENA) = 0)      Condición que activa el trigger
begin
        PRINT 'ERROR'
        ROLLBACK TRANSACTION       Acciones que ejecuta el trigger si se activa
end
```

SELECT COUNT devuelve el número de filas de la tabla padre con un valor de clave primaria igual a la clave ajena de la fila a insertar.
El sistema proporciona una tabla de almacenamiento temporal (INSERTED) donde se guardarán las filas a insertar.

Si CL_AJENA = CL_PRIMARIA, permite la inserción, en caso contrario deshace la operación con ROLLBACK TRANSACTION.

El trigger correspondiente a la operación de actualización sería similar.