BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Apuntes Impresos

Bases de Datos en Informix

Primavera 2009

Dr. David Eduardo Pinto Avendaño

INTRODUCCIÓN

BASE DE DATOS Y MODELOS DE DATOS

Una base de datos está generalmente definida como un conjunto integrado de datos que modelizan un universo dado. Este universo está compuesto por objetos inter-relacionados, los objetos de un mismo tipo constituyen una entidad y el lazo habido entre entidades se le denomina asociación.

El proceso de descripción de asociaciones y entidades se llama modelización y se hace con la ayuda de un modelo de datos, existen actualmente cuatro modelos de datos diferentes:

1. **Modelo jerárquico.**
2. **Modelo en red**
3. **Modelo objeto**
4. **Modelo relacional:** el principio básico de este modelo consiste en representar tanto las entidades como las asociaciones con la ayuda de relaciones denominadas también tablas. Una tabla está compuesta por líneas y columnas, cada línea representa un objeto (proveedor-articulo) las columnas representan los atributos de dicho objeto. Una tabla es una estructura.

OBJETIVOS DE LOS SISTEMAS DE GESTIÓN DE BASE DE DATOS

Las funciones de los S.G.B.D. son:

1. Debe permitir la definición de todos los datos
2. Debe permitir manipular datos
3. Debe establecer controles para la seguridad de estos datos
4. Debe permitir los accesos concurrentes.

SUBLENGUAJES DEL SQL

LDD. Lenguaje de descripción de datos.
LMD. Lenguaje de manipulación de datos
LCM. Lenguaje de control de datos.

OBJETIVOS SGBD

1. **Definición de datos**: (modifica la estructura o añade campos). La misión del LDD es describir y definir todos los esquemas que participen en la base de datos. Esto consiste en la descripción de los objetos que vamos a representar. La descripción de todas las estructuras que formen nuestra base de datos.

Definición de vista: es una visión parcial de la tabla. "cuando en una tabla alguna parte de esta no quiero que tenga derecho a manipularla nadie"

2. **Manipulación de datos:** LMD recoge todas las operaciones de intercambio de datos entre las tablas, estas operaciones pueden ser de consulta o de puesta al día (inserción, modificación, supresión) estas operaciones se realizan con la ayuda del denominado LMD.

-consultas

Operaciones

- puesta al día
  - insertar datos
  - modificar datos
  - suprimir datos

3. **Sesguridad de los datos:** consiste en garantizar que sólo los usuarios autorizados puedan efectuar operaciones correctas sobre la Base de Datos para ello se dispone de 2 tipos.

- Control sobre la base de datos
- Control sobre las tablas

GESTION DE LOS ACCESOS CONCURRENTES

El principal objetivo de la implantación de una base de datos es poner a disposición de un gran numero de usuarios en conjunto integrado de datos, estos datos podrán ser manipulados por los diferentes usuarios y es ahora cuando se debe garantizar la coherencia de los datos después de las diversas manipulaciones. Esto se garantiza con la ayuda del concepto de transacción "se define como transacción a una unidad lógica de tratamiento que aplicada a un estado coherente de una base de datos restituye un nuevo estado coherente de la base de datos pero con estos modificados, únicamente puede ser modificada completamente anulado".

CONCEPCIÓN DE UNA BASE DE DATOS

El ciclo de vida de una base de datos puede descomponerse en 3 etapas:

1. **Concepción:** la fase de concepción consiste en reproducir en el mundo real con ayuda de uno de los modelos de datos conocidos (relacional). El resultado de esta fase en un esquema escrito según un formalismo cualquiera no interpretable por el S.G.B.D.

2. **Creación de la B.D. vacía:** La 2ª fase consiste en traducir este esquema en ordenes comprensibles para el S.G.B.D. como resultado se obtiene la estructura de la base de datos desprovista de cualquier tipo de información.

3. **Explotación:** Es en esta fase donde los registros serán manipulados con la ayuda de los lenguajes de programación. Es ahora cuando los usuarios pueden consultar los datos y ponerlos a punto durante el resto de la vida de la base de datos.

PROBLEMAS ADQUIRIDOS POR UNA MALA CONCEPCION DE UNA BASE DE DATOS

En las tablas hay que procurar que no haya duplicidad de datos:

1. **Redundancia de datos:** si un cliente ha realizado más de un pedido todos los datos de este cliente estarán repetidos tantas veces como pedidos haya, lo mismo sucede para los artículos esto es opuesto al principal objetivo de una base de datos que consiste en evitar la repetición de los mismos.

2. **Puestas al día múltiple:** Para poder asegurar la coherencia de los datos es necesario efectuar puestas a día multiples. ”Cuando un cliente cambia de dirección”

3. **Incoherencia de los datos:** Sí una operación de puesta al día múltiple no se ha realizado completamente el estado de la base de datos queda incoherente y puede producir errores importantes.

4. **Pérdida de datos:** La supresión de una línea en la tabla de pedidos entraña la pérdida de todos los datos relativos a un cliente si no ha efectuado ningún otro pedido. Esto es cierto también para un artículo que no ha sido pedido por ningún otro cliente. Estas anormalidades constituyen lo que se ha convenido en llamar “comportamiento anormal de las tablas”, para evitar esto existe un proceso llamado “normalización” que entre otras cosas intenta esclarecer los conceptos de “dependencia funcional y estado de las tablas”.

5. **Dependencia funcional:** Este concepto se aplica a las columnas y consiste en hacer corresponder un único valor a aquella columna o columnas que consideremos más significativas.

6. **Estado de la tabla:** Se dice que una tabla esta en estado de 1ª forma normal si toda columna de esta tabla no puede tener más que valores atómicos, un valor es atómica si él no es divisible.

EL MODELO RELACIONAL Y EL SQL

Un modelo relacional posee tres grandes aspectos:

Estructuras: Definición de objetos que contengan datos y que son accesibles a los usuarios.

Operaciones: Definir acciones que manipulen datos u objetos.

Reglas: Leyes para gobernar la información, como y quien manipular.

Una base de datos relacional simplifica y definida como un modelo de información es estrictamente visualizable por los usuarios mediante tablas. Una tabla esta compuesta por una matriz bidimencional de filas y columnas. En cualquier ocasión la información es cambiada en una base de datos relacional, cualquier información es el resultado de una consulta presentad por el usuario en el formato filas/columnas.

Operaciones sobre Tablas

Todas las operaciones que podamos realizar sobre las tablas, vistas o elementos de ellas, están integradas en el SGDBR (Sistema Gestor de Bases de Datos Relacional) como rutinas. Ejemplos de operaciones son:

- **Selección**: Obtiene un subconjunto de filas de la tabla o vista, que cumplen una determinada condición.
- **Proyección**: Obtiene un subconjunto de columnas de todas las filas de la tabla.
- **Unión**: Realizamos la unión de varias tablas, cuyo resultado será el conjunto de todas las filas de las tablas origen. Las columnas respectivas de dichas tablas deben ser iguales entre sí.
- **Diferencia**: Inversa a la anterior, devuelve las filas que estén en una tabla y no pertenezcan a una segunda tabla. Deben por tanto ser iguales también las columnas respectivas entre sí.
- **Producto cartesiano**: El resultado será una fila por cada combinación entre cada fila de una tabla y todas las de la otra. Los valores de ambas filas se concatenarán.
- **Intersección**: Obtiene aquellas filas que sean idénticas en ambas tablas.
- **Join**: Es la operación de unir filas de dos tablas a través de algún campo común (normalmente la clave), dando como resultado filas con la suma de columnas de amblas tablas cuando se cumpla la condición del Join a través del campo (o campos) relacionados.

SQL es una herramienta para organizar, gestionar y recuperar datos almacenados en una base de datos informática. El nombre "SQL" es una abreviatura de Structured Query Languaje (Lenguaje de consultas estructurado). Como su propio nombre indica, SQL es un lenguaje informático que se puede utilizar para interaccionar con una base de datos y más concretamente con un tipo especifico llamado base de datos relacional.

SQL es a la vez un lenguaje fácil de aprender y una herramienta completa para gestionar datos. Las peticiones sobre los datos se expresan mediante sentencias, que deben escribirse de acuerdo con unas reglas sintácticas y semánticas de este lenguaje.

Su aprendizaje no solo sirve para esta aplicación sino , también, para todas las existentes en el mercado que soporten este lenguaje ya que es un lenguaje estándar por haberse visto consolidado por el Instituto Americano de Normas (ANSI) y por la Organización de Estándares Internacional (ISO).

Ciertas definiciones serán necesarias y se presentan a continuación.

- **Base de datos:** Esta compuesta de un conjunto de tablas del sistema creadas implícitamente por él y por un conjunto de tablas y vistas creadas por el usuario.

- **Tablas:** En informix el universo está descrito con la ayuda de tablas, cada una representa a una entidad o a una asociación entre entidades. Las tablas están compuestas de columnas o de líneas o filas llamadas convencionalmente campos y registros. Una columna representa un atributo de la entidad y para describirla es necesario especificar un nombre y un tipo de datos, una particularidad de las columnas es que pueden permitir o no valores nulos.

- **Fila:** Es una combinación de los diferentes atributos del objeto (registro).

- **Vistas:** Es una tabla virtual definida sobre las tablas bases descritas por el usuario su objeto es permitir a los usuarios manipular un subconjunto de datos.

- **Usuarios:** uno de los objetivos de un sistema de gestión de base de datos es el de afianzar la seguridad de los datos, para hacer esto el lenguaje de programación genera todos los usuarios y sus permisos de acceso para acceder a una base de datos, cada usuario debe disponer de su autorización correspondiente y es el propietario de la base de datos el que debe dar y retirar permisos de acceso.

- **Indices:** Se utilizan para aumentar el rendimiento del sistema y asegurar la integridad de los datos. A cada tabla que se crea se la asocia automáticamente una tabla índice que contiene la posición del registro según la columna especificada como índice.

SQL EN INFORMIX

LENGUAJE PROGRAMACION RDSQL

Cesación de una Base de Datos
CREATE DATABASE nombre-base-datos;

Para borrar una Base de Datos
DROP DATABASE nombre-base-datos;

Abrir Base de Datos
DATABASE nombre-base-extensión [EXCLUSIVE];

CREAR TABLAS

CREATE TABLE nombre

Tipos de datos:

- **CHAR (N):**Definen una cadena de caracteres desde una posición. A-32767

- **SMALLINT:** Para un numero entero corto cuyo valor debe estar comprendido entre (+,-)32767

- **INTEGER:** Define un numero entero largo, su valor esta comprendido entre(+,-) 2.150.000.000

- **DECIMAL (M,N):** Permiten números con fracciones decimales, el valor máximo de (M) que indica las posiciones del campo es de 32 y (n) que indica posiciones decimales debe ser menor o igual que (M).

- **SMALLFLOAT:** Para un numero corto en coma flotante, es equivalente a DECIMAL (8).

- **FLOAT:** Para un numero en coma flotante largo. Es equivalente a DECIMAL(16).

- **SERIAL [(N)]:** Indica un numero entero único y secuencial que se incrementa de forma automática por defecto. El valor inicial es 1 a no ser que se le indique lo contrario. Sólo puede haber uno por tabla.

- **MONEY (M,N):** Albergan números de tipo decimal con dos cifras después de la coma para expresar una moneda (siempre que no digamos lo contrario).

- **MONEY:** Es equivalente a datos decimales.

Money (8) decimal l (8 posiciones, 2 decimales).

- **DATE:** Sirve para indicar un campo de fecha con el formato (dd-mm-AAAA).

Crear una tabla temporal

CREATE TEMP TABLE nombre-tabla (nombre-columna-timo [NOT NULL].

MODIFICACION DE LA EXTRUCTURA DE UNA TABLA

- ALTER TABLE nombre-tabla

ADD (nombre-columna-tipo [BEFORE nombre-columna],...);

- **Borrar columnas:**

ALTER TABLE nombre-tabla
DROP (nombre-columna,...);

- **Modificar columnas:**

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

- **Borrar tablas:**

DROP TABLE nombre: “borra la tabla por completo”

- **Renombrar el nombre de la tabla:**

RENAME TALBE nombre.antiguo TO nombre.nuevo;

- **Renombrar el nombre de una columna:**

RENAME COLUMN nombre-tabla.nombre-columna TO nombre.nuevo;

PERMISOS SOBRE BASE DE DATOS

La atribución de privilegios de acceso a una base de datos se agrupan en tres operaciones. Se puede así autorizar a uno o varios usuarios a conectarse a una base de datos y a manipular sus componentes sin la menor opción de modificación o creación de estructuras, esta autorización se realiza con el privilegio CONNECT que es la autorización mínima que se puede dar a un usuario.

Igualmente se puede atribuir a un usuario un privilegio que le permita manipular los objetos de una base de datos y crear o modificar sus estructuras, este privilegio se le conoce como RESOURCE.

El privilegio de más alto nivel es el DBA que significa Administrador de Base de Datos, este ofrece a su beneficiario la totalidad del poder sobre una base de datos, el poseedor de dicho privilegio podrá además atribuir y retirar privilegios a los demás usuarios.

INSTRUCCIÓNES PARA DAR PRIVILEGIOS

- GRANT privilegio-DB TO PUBLIC | usuario1, usuario2,...;

Privilegios DB:

1. **CONNECTER-DB:** para todos los usuarios
2. **RESOURCE-DB:** para varios usuarios en concreto.
3. **DBA:** para un usuario sólo. Para hacer lo que quieras.

Ejemplo:

Dar privilegios a más de uno:

GRANT RESOURCE TO INF110, INF111;

RETIRADA DE PRIVILEGIOS

El propietario o los beneficiarios del privilegio DBA tienen la posibilidad de retirar en todo momento a uno o varios usuarios los privilegios sobre dicha base.

- REVOKE privilegio-DB FROM PUBLIC |usuario1,..;

PRIVILEGIOS SOBRE TABLAS: "sólo para la tabla que estamos trabajando".

Además de los privilegios de acceso a una Base de Datos en S.Q.L. permite conceder y retirar privilegios a nivel de tabla, estos últimos los puede conceder por el administrador de la Base o por el propietario, y determinan las operaciones que podrán efectuar sus tenedores sobre la tabla.

- GRANT privilegio-tabla ON nombre-tabla TO PUBLIC | LISTA USUARIOS [WHIT GRANT OPCION];

Privilegios tabla:

- **ALTER:** este privilegio concede modificar la estructura sobre una tabla. Y sólo modificar la estructura. Poner, quitar columnas o cambiar características de la columna.

"EJ." GRANT ALTER ON clientes TO PUBLIC WITH GRANT OPCIÓN;

- **DELETE:** permite suprimir datos de una tabla.

Borra registros. Cuando en una tabla una persona se va, DELETE borra sus datos.

- **INDEX:** permiso de creación de índices.

- **INSERT:** permite insertar nuevos datos en la tabla. Inserta registros, cuando de da de alta alguien INSERT inserta sus datos en la tabla.

- **SELECT:** (lista de columnas). Permite consultar una parte o la totalidad de la tabla.

"ej." GRANT SELECT ON clientes TO PUBLIC;

Para especificar los datos de la columna que yo quiero que un usuario vea.

GRANT SELECT (nombre, ape1, ape2) ON clientes TO Inf116;

- **UPDATE:** actualizar poner al día. Permite modificar una parte o la totalidad de las columnas. "Sólo en contenido" no la estructura.

- **ALL PRIVILEGES:** permiso para todos los privilegios anteriores. Los privilegios se los puedo dar a uno o a todos.

RETIRADA DE PRIVILEGIOS:

- **REVOKE: Revocar el privilegio de tabla dado**.

REVOKE privilegios-tabla ON nombre-tabla FROM PUBLIC | LISTA USUARIOS;

- **Para retirar privilegios a todos los usuarios que se los hemos dado:**

REVOKE ALL PRIVILEGES ON clientes FROM PUBLIC;

CREAR Y BORRAR SINONIMOS

Son de gran utilidad porque facilitan la formulación de las sentencias. Deben ser diferentes a los nombres de las tablas, vistas, etc. Y tienen que ser únicos dentro de la base de datos.

- CREATE SYNONIM nombre FOR nombre-tabla;

Una vez creado, cuando quieres modificar algo en la tabla en vez de poner el nombre de la tabla pones el sinónimo que es igual.

"ejemplo"

RENAME COLUMN c.entidad TO enti;
RENAME COLUMN clientes.entidad TO enti;

- DROP SYNONIM nombre-sinonimo;

INDICES

Son utilizados para acelerar el acceso a los datos y asegurar la unicidad de líneas en la tabla. Un índice permite acceder rápidamente a las líneas de la tabla, contienen la dirección física de cada una de las líneas de la tabla.

Para buscar una línea de esa tabla el sistema accede al índice donde el tamaño es más pequeño y encuentran la dirección de la línea de búsqueda. En ese momento accede a la tabla.

La otra utilización de los índices es la de asegurar la unicidad de las líneas en una tabla, el S.Q.L. no ofrece ninguna manera que permita definir las claves de las tablas. No se dispone más que en esta opción para asegurar que los datos sean únicos.

- **Crear INDICES:**

CREATE [UNIQUE/DISTINCT] [CLUSTER] INDEX nombre del índice ON nombre tabla (nombre-columna [ASC/DES],...);

UNIQUE: los datos son únicos sin valores duplicados.
DISTINCT:
CLUSTER: Son índices agrupados. Físicamente reorganiza la tabla por las columna/as que yo he indicado. Y sólo puede haber una por programa.

- **Modificar INDICES:**

Solamente se utilizan para los índices agrupados y es frecuentemente utilizado ya que sólo se nos permite un índice CLUSTER pro base.
ALTER INDEX nombre-indice
TO [NOT] CLUSTER;

- **Con la opción NOT CLUSTER** desacemos el índice agrupado y se toman como si fuese un índice normal lo que nos permitiría volver a reorganizar la tabla con otro criterio.

- **Con la opción TO CLUSTER** exige que este índice sea un índice CLUSTER liberado y que no exista ningún otro índice CLUSTER, vuelve a crear el índice agrupado.

- **Borrar INDICES:**

DROP INDEX nombre-indice;

LENGUAJE DE MANIPULACION DE DATOS (LMD/DML)

CONSULTA DE DATOS:

Se hace a través de la instrucción SELECT y existen dos tipos de consultas diferenciadas.

- Cuando quieres visualizar algunas columnas en concreto

```
SELECT lista de columnas
FROM lista de tablas;
```

"ej" SELECT idcliente, nombre, ape1 ape2
FROM clientes;

- Cuando quieres visualizar todas las columnas creadas de una única tabla:

```
SELECT *
FROM clientes;
```

- Cuando quieres visualizar datos de dos o más tablas diferentes juntas, que no tienen columnas que se llaman igual:

```
SELECT nombre, ape1,ape2, idarticulo, descripción
FROM  clientes, artículos;
```

- Cuando quieres visualizar datos de dos o mas tablas diferentes juntas, y que tienen columnas que se denominan igual:

SELECT clientes.nombre, clientes.ape1, clientes.ape2, proveedores.nombre, proveedores.ape1, proveedores.ape2
FROM clientes, proveedores;

CONSULTA CALIFICADA:

Se hace a partir de una consulta simple y una cláusula WHERE, esta puede estar seguida de una o varias condiciones que a su vez están relacionadas entre sí por los operadores lógicos AND y OR y el operador lógico NOT, se puede utilizar para especificar la negación:

Existen 3 tipos de condiciones:

1. Condiciones de comparación.
2. Condiciones de combinación
3. Condiciones de subsistencia.

CONDICIONES DE COMPARACION:

Permite comparar una columna o una expresión con otra columna expresión o lista de columnas.

Su formato es:

```
SELECT nombre-columna
FROM lista de tablas
WHERE condición;
```

Condiciones:

- Exp. Operador relacional Exp.
- Exp [NOT] BETWEEN exp. AND exp.
- Exp [NOT] IN (lista de valores)
- Nombre-columna [NOT] LIKE “cadena de caracteres”
- Nombre-columna [NOT] MATCHES “cadena de caracteres”
- Nombre-columna IS [NOT] NULL

Exp.: representa una expresión que es un nombre de columna una constante o una combinación de las dos relacionadas por operadores aritméticos. Los operadores aritméticos utilizados en S.Q.L. son:

(*, /, +, -)

Una expresión puede incluir dos funciones pre-definidas:

TODAY: Fecha del sistema.
USER: nombre del usuario.

```
SELECNT *
FROM pedidos
WHERE fecha pedido=TODAY;
```

Operadores generales: Además podemos utilizar los siguientes operadores de relación:

(=, <>, >, <, >=, <=)

```
SELECT *
FROM pedido
WHERE cantidad > 100 AND fechapedido = TODAY;
```

Predicado BETWEEN:

Permite comparar el valor de la expresión situada a la izquierda de la palabra BETWEEN con los valores comprendidos en el intervalo definido por las expresiones comprendidas a la izquierda y derecha de la palabra clave AND.

"Ejemplo"
consultar todas las columnas de la tabla artículos donde el precio unitario sea mayor o igual que 100 y que sea menor o igual que 300.

```
SELECT *
FROM artículos
WHERE precio unitario >=100
AND  precio unitario <=300;
```

Predicado IN:

Permite comparar el valor de la expresión situado a la izquierda de la palabra IN con la lista de valores encerrados entre paréntesis situados a la derecha de dicha palabra.

```
SELECT *
FROM clientes
WHERE ciudad="Santander"
Ciudad = "Bilbao"
OR ciudad = "Torrelavega";
```

```
SELECT *
 FROM clientes
WHERE ciudad IN ("Santander","Bilbao",
  "Torrelavega");
```

Predicado LIKE:

Permite utilizar una comparación de semejanza entre el valor de una columna y el de una cadena de caracteres utilizando caracteres genéricos de sustitución, estos son:
(%, _).

```
SELECT *
FROM clientes
WHERE Ape1 LIKE "A%"
```

%: sustituye a varios caracteres.
_ : sustituye a un solo carácter.

```
SELECT *
FROM clientes
QHERE ape1 LIKE  "_TI%";
```

Predicado MATCHES:

Es un complemento al predicado LIKE y ofrecen los siguientes caracteres de sustitución (*, ?, %, _).
Nos va a permitir hacer una selección.

```
SELECT *
FROM clientes
WHERE ape1 LIKE "A%"
OR ape1 LIKE "B%"
OR ape1 LIKE "C%"
```

```
SELECT *
        FROM clientes
WHERE ape1 MATCHES "[A-C]*";
```

Predicado IS [NOT] NULL:

En S.Q.L. el valor nulo tiene un valor indefinido diferente de 0 y de una cadena vacía. El S.Q.L. permite una comparación sobre los valores nulos.

```
SELECT nombre, ciudad
FROM clientes
WHERE ciudad IS NOT NULL;
```

ORDENACION DEL RESULTADO:

El S.Q.L. permite operaciones de ordenación con los datos extraídos con una consulta SELECT. Se puede ordenar hasta por 8 columnas diferentes en una misma instrucción.

La sintaxis es la siguiente:

```
ORDER BY [ASC/DESC] lista de columnas
```

"ejemplo":

```
SELECT *
FROM clientes
WHERE ciudad IS NOT NULL
ORDER BY DESC ape1;
```

FUNCIONES DE GRUPO

S.Q.L. ofrece cinco funciones básicas para realizar cálculos estadísticos así se puede determinar el nº de líneas por tabla o por grupo que cumple una condición, calcular la suma y la media de columnas numéricas o determinar el valor máximo o mínimo de una columna.

- **COUNT (* | DISTINCT | nombre columna)**
  Cuenta el numero de líneas que satisface la sentencia.

Para sacar las distintas ciudades que tengo. Si tengo 2 iguales sólo me cuenta 1 al ser repetidas.

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

```
SELECT (DISTINCT CIUDAD) ciudad
FROM clientes
```

- **SUM ([DISTINCT] columna)**

Suma los valores de la columna que satisface la sentencia.
La opción DISTINCT suma los valores únicos de la columna.
La columna debe de ser de tipo numérico y puede ser una expresión.

Idarticulos	Existencias	precio
1	10	15
2	5	10
3	5	10

- Quiero sumar la columna artículos.

```
SELECT SUM (existencias*prunitario)
FROM artículos;
```

- Sumar los valores únicos de todos los registros en todas las existencias.

```
SELECT SUM (DISTINCT existencias)
FROM artículos;
```

- **AVG ([DISTINCT] columna)**

Calcula la media de los valores de la columna que satisfacen la sentencia.
La opción DISTINCT calcula la media de valores únicos.

"Ejemplo"

- calcular el precio de los artículos con precio único.

```
SELECT AVG (DISTINCT, precio)
    FROM artículos;
```

- **MAX (columna)**

Visualiza el máximo de los valores de la columna que satisfacen la sentencia.
La columna debe de ser de tipo numérico y puede ser una expresión.

"Ejemplo"

- Calcular el precio máximo que yo cobro por mis artículos.

```
SELECT MAX (precio)
FROM artículos
```

- **MIN (columna)**

Visualiza el mínimo de los valores de la columna que satisfacen la sentencia.
La columna debe de ser una expresión y tiene que ser numérico.
"Ejemplo"

- Calcular el precio mínimo que yo cobro por mis artículos

```
SELECT MIN (precio)
```

FROM artículos

En presencia de valores nulos la función COUNT toma en cuenta estas líneas las demás funciones NO:

FUNCIONES DE FECHA

El lenguaje de programación S.Q.L. facilita un conjunto de funciones aplicables a la fecha, que permiten por ejemplo extraer a partir de un dato tipo fecha al día del mes o de la semana así como convertir una expresión en tipo fecha.

- **DATE (expr)**

  Convierte una expresión en forma de carácter o columna numérica en un valor de tipo fecha.

  SELECT DATE (101097)
  FROM nombre tabla

- **DAY (fecha)**

  Devuelve el día del mes de la columna fecha.

  SELECT DAY (fecha pedido)
  FROM nombre tabla

- **MONTH (fecha)**

  Devuelve el mes de la expresión fecha

  SELECT MONTH (fecha pedido)
  FROM nombre tabla

- **WEEKDAY (fecha)**

  Devuelve el día de la semana de la expresión fecha.

  SELECT WEEKDAY (fecha pedido)
  FROM nombre tabla

- **YEAR (fecha)**

  Devuelve el año de la expresión fecha.

  SELECT YEAR (fecha pedido)
  FROM nombre tabla

- **MDY (Exp1, Exp2, Exp3)**

  Convierte las 3 expresiones en un valor de fecha. Las expresiones deben de ser numéricas y de tipo entero y limitadas a valores entre 1 y 12 para la 1ª expresión y entre 1 y 31 para la 2ª expresión.

  SELECT MPY (1-4,95) ⟶ nos saldrá (01.04.1995)

AGRUPACIÓN DE DATOS:

S.Q.L. permite agrupar las líneas de datos que tengan valores comunes, las columnas representan uno o varios nombres de columnas separados por comas y que deben obligatoriamente figurar en la lista de selección de la cláusula SELECT. Se puede reemplazar los nombres de las columnas por un entero que indique su posición relativa.

- Contar todos los clientes clasificados por ciudad.

```
SELECT COUNT (*), ciudad
FROM clientes
GROUP BY cuidad;
```

- Listar el numero de líneas por cada pedido

```
SELECT COUNT (*), nºpedido
FROM pedidos
GROUP BY nºpedido;
```

- Visualizar el importe de cada pedido.

```
SELECT NUMBER, SUM (precio*cantidad)
FROM linea_pedido
GROUP BY 1;
```

CONDICIONES DE COMPOSICION

Una composición es un enlace entre dos tablas que dispongan al menos de una columna en común, la operación de composición consiste en crear una tabla temporal compuesta por las líneas de ambas tablas que satisfagan la condición.

Formato → SELECT lista de columnas
FROM lista de tabla
WHERE condición;

"Ejemplo" hay una de ellas que la quieres ver y → Cuando en dos tablas que vas a comparar tienes que especificar de que tabla es

```
SELECT nombre,ape1,ape2, clientes.idcliente, fechapedido
FROM clientes, pedidos
WHERE clientes.idcliente=pedidos.idcliente;
```

TIPOS DE ENLACES o UNIONES

1. **Equicomposición**: es una composición donde la condición es una comparación de igualdad entre dos columnas de diferentes tablas.

2. **Thetacomposición:** es una composición donde la condición es una comparación de 2 columnas utilizando un operador distinto al de igualdad.

3. **Composición externa**: es una composición que favorece una tabla con respecto a otra así las líneas de la tabla dominante serán seleccionadas aunque la condición no se haya cumplido.

UNION DE SENTENCIAS:

S.Q.L. permite la fusión de datos pertenecientes a varias sentencias SELECT formuladas sobre una o varias tablas, una condición esencial para efectuar esto es la necesidad de la misma lista de selección.

Puede a su vez incluir una columna de ordenación y debe si se codifica obligatoriamente hacer referencia a una posición relativa de la lista de columnas.

SELECT "lista de columnas"
UNION (ALL)
SELECT "lista de columnas"
ORDER BY

"ejemplo"

- tenemos dos tablas clientes y clien.santander queremos visualizar las dos tablas a la vez.

```
SELECT ape1, ape2, nombre
FROM clientes
UNION
SELECT ape1,ape2,nombre
FROM clien.santander
ORDER BY 1;
```

INSERCIÓN DE DATOS

La inserción de nuevos datos se hace con dos tipos de sentencias, una permite la inserción de datos provenientes del mundo exterior y otra permite la inserción de datos entre tablas.

1. Inserción datos del exterior:

INSERT INTO "nombre tabla" [("lista columnas")]
VALUES (datos de las columnas metidos por orden)

"ejemplo"

```
INSERT INTO clientes
VALUES (5."Jorge","Diaz","Cabezas"....);
```

2. Insertar datos entre tablas:

INSERT INTO "nombre tabla" [(lista columnas)]
Sentencia SELECT

La sentencia SELECT debe recoger el mismo n º de columnas que se han especificado en la lista de columnas si esta está explícita o el mismo nº de columnas especificadas en la creación de la tablas si esta se ha omitido.

La sentencia SELECT no puede contener una cláusula ORDER BY ni una cláusula INTO TEMP.

"ejemplo"

- Copiar todos los clientes que sean de Santander de la tabla clientes a la tabla Cli_santander.

```
INSERT INTO cli_santander (idcliente,nombre)
SELECT idcliente,nombre
            O
            *
FROM clientes
WHERE ciudad="santander";
```

MODIFICACIÓN DE DATOS

S.Q.L. permite modificar los datos existentes en cualquiera de los registros existentes:

```
UPDATE nombre tabla
    SET nombre columna=Expr,...
    [WHERE condición]
```

"ejemplos"

- Aumentar en un5% el punitario de mis artículos cuando punitario sea menor de 1000.

```
UPDATE artículos
    SET punitario=punitario*1.05
    WHERE punitario<1000;
```

BORRADO DE DATOS

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

"ejemplo"

- Borrar al cliente nº 3

```
DELETE FROM  clientes
    WHERE  idcliente=3;
```

EXTRUCTURA DE UN ARCHIVO DE FORMATOS

Un formato puede ser monotabla o multitabla dependiendo de las tablas a las que hace referencia pero en ambos casos está compuesto obligatoriamente de:

1. DATABASE
2. SCREEN
3. TABLES
4. ATTRIBUTES
5. Instructions ⟶ esta sección no es obligatoria es (opcional).

1. **DATABASE:** permite especificar el nombre de la base de datos donde está especificado el formato, es obligatoria.

DATABASE gestión [WITHOUT NULL IMPUT]
[SCREEN SIZE 24 BY 80]
[END]

2. **SCREEN:** Describe el aspecto externo del formato, es decir, la forma como va a mostrase durante la ejecución, cada ocurrencia de la palabra SCREEN se corresponde con la descripción de una página y todo lo que está entre las dos llaves constituye el aspecto de la pagina.

Cuando todas las páginas se han especificado se acaba la sección con la palabra END.

```
        SCREEN
        {_____ Aquí se ponen los
}_____    campos de las tablas
        END
```

3. **TABLES:** en esta sección se indican las tablas utilizadas en el formato puede contener uno o varios nombres de tablas dependiendo de si el formato es momotabla o multitabla. Termina opcionalmente con la palabra END.

```
TABLES
  Clientes
  Artículos
END
```

4. **ATTRIBUTES:** tiene un doble cometido:

- Es permitir realizar un enlace entre los campos designados en la sección TABLES.
- Determinar las características o atributos de cada campo. Estas características se refieren a:

- ♦ Introducción obligatoria u optativa de datos en un campo.
- ♦ Lista de valores aceptables.
- ♦ Valor por defecto.
- ♦ Mensajes de ayuda.
- ♦ Conversión automática de una cadena de caracteres en mayúsculas o minúsculas.

Esta sección de especificaciones de campo. Cada una de estas especificaciones describe uno de los campos de la sección SCREEN. En orden de la aparición de las especificaciones en esta sección determina el orden de desplazamiento del cursor durante la ejecución del formato.

```
    Attributes

    Identificador de campo = identificador de columna, [Lista de Atributos];
```

- Identificador de campo —F000, f001, f002, ...

- Identificador de columna tabla.columna ej. Clientes.idcliente
- Lista de atributos Se pueden poner tantos como se quieran.

[A]
A= display only TYPE CHAR (IT), RESERVE;

ESPECIFICACION DE ATRIBUTOS

1. **AUTONEXT:** Implica que durante la ejecución del formato el cursos pasa automáticamente al campo siguiente cuando se llena el campo actual.

Se utiliza cuando:
- El campo tenga siempre la misma longitud.
- Cuando el campo está partido.

F007 = clientes.codpostal, AUTONEXT;

2. **COMMENTS:** Permite especificar un mensaje de ayuda que será mostrado cuando el cursor se posiciona sobre el campo durante la ejecución del formato.

F007 = clientes.codpostal, COMMENTS = ”introducir código postal”, AUTONEXT;

3. **DEFAULT:** Permite atribuir un valor por defecto a un campo. Se utiliza con los campos que pueden tener un valor probable.

F007 = clientes.ciudad, DEFAULT “SANTANDER”, AUTONEXT, COMMENTS = “introducir la ciudad”;

4. **VPSHIFT:** Sólo se permite en los campos de tipo carácter y transforma automáticamente todos los caracteres en mayúsculas.

F007 = cliente.ciudad, DEFAULT “SANTANDER”, AUTONEXT, COMMENTS = “introducir la ciudad”, VPSHIFT;

5. **DOWNSHIFT:** Sólo para datos de tipo carácter y permite transformar automáticamente todos los caracteres a minúsculas.

F007 = cliente.ciudad, DEFAULT “SANTANDER”, AUTONEXT, COMMENTS = “introducir la ciudad”, DOWNSHIFT;

6. **REQUIRED:** hace que la entrada de datos sea obligatoria, no se puede utilizar con el atributo DEFAULT.

7. **NOENTRY:** Es utilizado para impedir la entrada de datos en un campo, esta prohibición sólo se aplica en el caso de añadir nuevas líneas, es a menudo utilizado con el atributo DEFAULT y está aplicado por defecto a los campos de tipo SERIAL.

8. **NOUPDATE:** Impide la modificación de los datos del campo que lleven el atributo asociado cuando se ejecute un formato en modo de actualización. No impide la entrada de datos en modo inserción y es la poción tomada por defecto en los campos de tipo SERIAL.

9. **RESERVE:** La especificación de este atributo entraña la aparición del campo en vídeo invertido, tiene efecto en las cuatro funciones del generador de formatos.

10. **RIGHT:** Este atributo entraña la justificación a la derecha de los datos introducidos en modo adición o modificación, sólo es aplicable en los datos de tipo CHAR.

11. **ZEROFILL:** Se utiliza para justificar los datos numéricos a la derecha y completar el resto del campo a ceros.

12. **INCLUDE:** En ciertos campos se debe no autorizar la entrada más que de ciertos valores conocidos. Este atributo permite mediante una lista de valores cumplir este objetivo.
    INCLUDE = (100 TO 400)

13. **VERIFY:** Verifica la correcta introducción de un dato para ello obliga a repetirlo 2 veces.

14. **FORMAT:** Se utiliza para los campos de tipo DECIMAL, FLOAT, SMALLFLOAT y de tipo DATE. La cadena de caracteres es una serie de signos # y eventualmente un punto decimal si es necesario el numero es redondeado antes de su visualización.

FORMAT= "cadena de caracteres"
F001= clientes.artículos, FORMAT = "###.#"

Para los campos de tipo fecha se utilizan los siguientes símbolos:

- Mm: Numero del mes en dos cifras. (1-12)
- Mmm: Abreviatura del nombre del mes. (escribirla en ingles).
- Dd: Numero en el día del mes en 2 cifras. (1-31).
- Ddd: Abreviatura del nombre del día. (escribirla en ingles).
- Yy: El año con dos cifras.
- Yyyy: El año con cuatro cifras.

Se pueden combinar estos símbolos con el espacio en blanco y los caracteres especiales (\, /, -).

Ej. F001=pedidos.fpedido, FORMAT = "símbolos"

15. **PICTURE:** Tiene un efecto similar a atributo format, pero se utiliza para los campos tipo CHAR. La cadena de caracteres está compuesta por los caracteres "A, #, X", espacios en blanco" y caracteres especiales "/, -, :, ;".

A: Una letra cualquiera de la A-Z, minúsculas y mayúsculas.
#: Representa un dígito cualquiera 0-9.
X: Un carácter cualquiera.

F001 = clientes.teléfono, PICTURE= "(###)######".

En pantalla saldrá
[( )]

FORMATOS MULTITABLA:

A menudo es necesario manipular datos provenientes de más de una tabla, a estos formatos se los denomina formatos multitabla. Las tablas que constituyen dichos formatos deben pertenecer obligatoriamente a la misma base de datos.

Generalmente las tablas de un formato multitabla están relacionados entre sí, el enlace entre las tablas de un formato se realiza con la ayuda de la noción campo común y la relación maestro detalle.

Un campo común es un campo del formato que se corresponde con columnas de diferentes tablas, para establecer esta relación se describen 3 tipos diferentes de enlace:

1. Enlace simple o de composición.
2. Enlace de verificación.
3. Enlace de referencia.

1. Enlace simple o de composición:

Para precisar este tipo de enlace es necesario definir el campo de enlace y las columnas asociadas en la sección de atributos.

Identificador de campo= columna1=columna2;

Esto significa que al campo designado por identificador de campo serán asociadas las dos columnas pertenecientes a 2 tablas diferentes. Cuando un valor se introduce en este campo sera atribuida simultáneamente a las 2 columnas.

F000=clientes.idcliente=pedido.idcliente;

2. Enlace de verificación:

Mediante el atributo INCLUDE es posible imponer una lista o intervalo de valores aceptables para una columna dada. Esta posibilidad no siempre es suficiente porque la lista de los valores aceptables es dinámica y proviene de una tabla. Para especificar un enlace de verificación se especifica el nombre del campo, la columna que va a recibir el valor a introducir y la columna con la que será realizada la verificación.

Identificador de campo=columna1=*columna de verificación;
F001=linea_pedido.idartículo=*artículos.idartículo;

3. Enlace de referencia:

Este enlace permite mostrar los datos que tengan relación con aquellos que acaban de introducirse con el campo. A su vez también es posible verificar la existencia de ese dato. Para definir un enlace de este tipo hace falta especificar en primer lugar el campo de la columna, que será utilizado como base de búsqueda, a continuación y para campo de referencia la columna de referencia corresponde y a la columna que nos permita hacer el enlace.

Identificador de campo=columna-base
LOOKUP campo de referencia=columna de referencia
JOINING [*] columna de enlace;

INSTRUCTIONS:

Es la secuencia opcional, comienza con la palabra clave INSTRUCTION y termina con END y permite las siguientes opciones.

1. Elegir como delimitadores de campo otros distintos de los de por defecto.

2. Definir las relaciones maestro detalle en las formas multitabla.

3. Definir bloques de control que permitan ejecutar ciertas acciones sobre los campos a continuación de un suceso dado.

1. Elección de delimitadores:

INSTRUCTIONS
DELIMITERS "XY"

Con esta opción podemos cambiar los delimitadores:

X: corresponde al delimitador de apertura del campo. Ej. (
Y: corresponde al delimitador de cierre del campo. Ej.)
Son permitidos aunque no aconsejados los espacios en blanco.

2. Definición de las relaciones maestro detalle:

Una relación maestro detalle o principal secundaria entre dos tablas de un mismo formato permiten manipularlas conjuntamente, esto es útil en el caso en el cual a una línea de una tabla le corresponda un conjunto de líneas de otra tabla. La 1ª tabla es llamada maestra o principal y la segunda tabla detalle o secundaria. Para establecer este tipo de relaciones es necesaria la existencia de un campo de enlace entre las tablas. Una tabla maestro puede tener varias tablas de detalle mientras que una tabla de detalle sólo puede tener una tabla maestra.

END
INSTRUCTIONS
DELIMITERS "( )"
Tabla maestra MASTER OF tabla detalle

3. Generalidades de los bloques de control:

Los bloques de control permiten definir acciones que serán emprendidas durante la ejecución del formato, después de la realización de un suceso dado. Estas acciones pueden ser ejecutadas antes o después de dicho suceso, para indicar esto utilizamos las palabras reservadas:

BEFORE: antes.

AFTER: después.

BEFORE | AFTER
Lista de sucesos OF lista de columnas o tablas
Acción 1
Acción 2

Listas de sucesos:

1. EDITADD
2. EDITUPDATE
3. REMOVE
4. DISPLAY
5. ADD
6. UPDATE
7. QUERY

EDITADD inserta datos nuevos EDITUPDATE modifica datos:

Corresponden a la entrada o modificación de datos en una tabla, permiten ejecutar una serie de acciones antes o después de la entrada de datos en un campo o en una tabla en modo de entrada o modo de actualización.

Para estos dos tipos de sucesos se distinguen los siguientes casos:

1. La lista de tablas o columnas sólo contiene nombres de columnas.

- Si el bloque es de tipo BEFORE las acciones serán ejecutadas desde que el cursos está situado sobre el campo correspondiente a la columna.

BEFORE EDITADD OF nombre, ape1
BEFORE EDITADD EDITUPDATE OF nombre, ape1

- Si el bloque es de tipo AFTER las acciones serán ejecutadas después de activar la tecla de retorno. Los controles fijados para los atributos son ejecutados antes de las acciones.

2. La lista de tablas o columnas sólo contiene tablas

- Si el bloque es de tipo BEFORE las acciones serán ejecutadas antes de que ningún dato sea introducido en el formato.

BEFORE EDITADD OF clientes

BEFORE EDITUPDATE OF clientes

- Si el bloque es de tipo AFTER las acciones serán ejecutadas cuando todos los campos hayan sido introducidos pero antes de insertar la línea en la tabla.

REMOVE: (borrar)

Permite ejecutar las acciones antes o después de la supresión de una línea en una tabla, no se aplica más que a tablas.

Ej. ⟶
```
INSTRUCTIONS
DELIMITERS "( )"
Pedidos MASTER OF L_pedidos
AFTER EDITADD EDITUPDATE OF cantpedida
LET C1=F008*cantped ⟶ visualizar en c1
AFTER REMOVE OF pedidos
COMMENTS "no olvidarse borrar sus correspondientes líneas"
```

DISPLAY:

Corresponde a la salida de datos sobre pantalla con una de las opciones PERFORM, no se aplica más que a tablas y no puede estar precedido más que de la palabra clave AFTER.

```
AFTER DISPLAY OF clientes
```

QUERY:

Corresponde a la consulta de una tabla, permite ejecutar las acciones después del empleo de la opción QUERY. No se aplica más que a tablas y no puede estar precedido más que de la palabra clave AFTER.

```
ATFER QUERY REMOVE OF clientes
LET C1=F001*F002
```

ADD:

Corresponde a la inserción de una línea en una tabla, permite ejecutar acciones después del empleo de la opción ADD del menú de PERFORM, no se aplica más que a tablas y no puede estar precedido más que de la palabra clave AFTER.

La diferencia entre este suceso y el suceso EDITADD es que para el primero las acciones serán ejecutadas después de la inserción de datos en la tabla, mientras que para el segundo las acciones son ejecutadas antes de la inserción de datos en la tabla.

```
AFTER EDITADD OF clientes
COMMENTS "cliente dado de alta"
```
Se visualizará después de que haya rellena*do todos los datos de la tabla* pero antes *hay que dar* ***ESC.***

```
AFTER ADD OF clientes
COMMENTS "cliente dado de alta"
```
se ejecuta después de dar ***ESC.***

UPDATE:

Corresponde a la modificación de una línea en una tabla, permite ejecutar acciones después del empleo de la opción UPDATE, no se aplica más que a tablas y no puede estar precedido más que de la palabra clave AFTER. La diferencia entre el suceso UPDATE y EDITUPDATE es semejante a la existente entre ADD y ADITADD.

ACCIONES DE BLOQUE DE CONTROL:

Se pueden definir en 5 tipos de acciones: * Editadd, * Editupdate

1. Desplazamiento del cursor (NEXTFIELD)
2. Visualización de un mensaje (COMMENTS).
3. Asignación de un valor a un campo (LET).
4. Alternativa entre 2 opciones (IF – THEN – ELSE).
5. Anulación de las últimas modificaciones y retorno al menú.

1. Desplazamiento del cursor (NEXTFIELD):

Por defecto el desplazamiento del cursor sigue el mismo orden que el de los campos en la sección de atributos. La acción NEXTFIELD permite modificar este orden y forzar al cursor a pasar a un campo dado, esta acción no es utilizable más que con los sucesos EDITADD y EDITUPDATE.

Su formato es:

NEXTFIELD=nombre.campo [EXITNOW]

- **Exitnow:** fuerza a terminar la operación en curso (inserción o modificación y después retornar al menú principal.

Ej.: Después de añadir en el campo cantidad ir a campo f004.

```
AFTER EDITADD OF cantidad
NEXTFIELD=F004
```

2. Visualización de un mensaje (COMMENTS)

Permite mostrar un mensaje sobre la línea de estado, su sintaxis es la siguiente.

COMMENTS [BELL] [REVERSE] “mensaje”

Ej.

```
BEFORE REMOVE OF compras
COMMENTS BELL  “debe borrar...”
NEXTFIELD=F001
```

3. Asignación de un valor a un campo (LET).

Permite asignar un valor simple o el resultado de una expresión a un campo del formato, el campo puede ser un campo de solo salida "DISPLAYONLY" o cualquier campo de la tabla activa. Se puede utilizar para atribuir valores en un campo con el atributo NO ENTRY.

La sintaxis es la siguiente:

LET campo=EXPRESIÓN.

E. AFTER EDITADD EDITUPDATE OF cantidad
LET c1=f007 * c2

- Una expresión puede ser:
- Un identificador de campo: (LET c1=f003).
- Una constante incluyendo TODAY para la fecha actual (LET c1=1995) o (LET c1=TODAY). Para eso C1 lo he tenido que haber descrito como campo tipo DATE.
- Una función de conjunto de la forma. (LET c1=FUNCTION OF campo)

FUNCTIONS:
* AVG
* MAX
* MIN
* COUNT
* TOTAL

4. Alternativa entre 2 opciones (IF – THEN – ELSE)

Permite ejecutar acciones en función del contenido de ciertos campos.

Su formato es:

IF condición THEN accion1 [ELSE accion2].

La condición: puede estar formulada utilizando los operadores (=, <, >, <>, <=, >=). Además de los operadores lógicos (AND, OR, NOT) y los operadores (IS NULL, IS NOT NULL).

Las acciones: pueden ser elementales o compuestas. En este segundo caso debe de estar precedidas por la palabra clave BEGIN y terminar en END.

Ej. Campo cantidad=f03
AFTER EDITADD EDITUPDATE OF cantpedida
IF F003 IS NULL THEN NEXTFUELD=F003

IF F003 IS NULL THEN BEGIN
COMMENTS "no puede estar vacío el campo"
NEXTFIELD=F003
END
ELSE LET C1=F003 * C2

5. Anulación de las últimas modificaciones y retorno al menú (ABORT)

Permite anular la acción en curso y retornar al menú sin modificar la tabla. Permite anular la inserción, modificación y supresión que esté en trámite de realizar. Solamente se va a poder utilizar con 3 sucesos:

- EDITADD
- EDITUPDATE
- REMOVE

Ej. BEFORE REMOVE OF lineapedido
ABORT
BEFORE EDITADD EDITUPDATE OF lineapedido
ABORT

LISTADOS:

Secciones que podemos utilizar en el listado:

* DATABASE
DEFINE
INPUT
OUTPUT
* SELECT
* FORMAT

*Son obligatorias.

DATABASE:

Es obligatoria, sirve para indicar el nombre de la base de datos para la cual se define el informe. Como todas las demás secciones terminan con la palabra reservada END.

El formato de la sección es el siguiente:

DATABASE
Nombre de la base de datos
END

DEFINE:

Permite definir los parámetros (valores que se dan en el momento de ejecutar la variable) y variables usadas en un informe. Los parámetros permiten especificar valores cuando se toma la ejecución del informe a partir del Sistema Operativo. No es posible usarolos desde el menú de S.Q.L.

Las variables se utilizan para permitir la entrada interactiva de ciertos valores, en el curso de la ejecución de informe.

El formato de definición de variables es el siguiente:

DEFINE

VARIABLE [1] nombre.tipo de datos
PARAM [1] nombre.tipo de dato

Las variables y los parámetros no pueden ser de tipo SERIAL.

INPUT:

Sección opcional está estrechamente ligada a la sección DEFINE, dado que permite introducir los valores a las variables, consiste en invitar al usuario a introducir un valor que será asignado a la variable. Para hacer esto se muestra un mensaje para indicar sobre el valor a introducir.

El formato es el siguiente:

INPUT
PROMPT FOR variable USING "mensaje"
END

Ej.

IMPUT
PROMPT FOR fechalistado USING "introduzca fecha"
END

OUTPUT:

Es opcional, permite fijar las dimensiones de una página del listado así como el medio de salida, consta de las siguientes cláusulas:

- **TOP MARGIN n:** Margen superior (por defecto toma valor 3).

N numero de líneas al comienzo de la página que me deje libres.

- **BOTTOM MARGIN n:** Margen inferior (por defecto toma valor 3).
- **LEFT MARGIN n:** Margen izquierdo (valor por defecto 5)
- **RIGHT MARGIN n:** Margen derecho (valor por defecto 132)
- **PAGE LENGTH n:** Longitud de la página (valor por defecto 60).

Por donde queremos que salga el listado

- REPORT TO [PRINTER| "nombre fichero"] [por omisión sale sólo por pantalla]
  → Sale a la vez por pantalla e impresora.

END

SELECT:

Esta sección obligatoria permite definir los datos del informe precisando las tablas, las columnas, las condiciones y el orden de selección eventual.

Cuando se utiliza la cláusula ORDER BY no está permitido la calificación de las columnas, si el nombre de la columna no es suficiente para asegurar la identificación única de la misma sé dbe asociar un alias a esta columna y utilizarlo posteriormente en la cláusula ORDER BY.

Ej.

```
SELECT x.idcliente identificación, nombre, npedido
FROM  clientes.x, pedidos.y
WHERE x.idcliente = y.idcliente
ORDER BY identificación, npedido
END
```

FORMAT:

Sección obligatoria permite fijar la forma y el aspecto del informe, es aquí donde se especifica cuales de los datos seleccionados en la sección SELECT serán presentados en el informe. Esta presentación consiste en:

1. Definir el contenido de la cabecera de la 1ª página y de las otras páginas del informe.
2. Definir el contenido del Pie de página.
3. Definir el contenido y la forma de cada línea del informe.
4. Definir lo que será mostrado antes y o después de cada grupo de líneas.

Esta sección tiene dos formatos:

El 1º se le atribuye por defecto y permite presentar todas las columnas seleccionadas en la sección SELECT, su sintaxis es:

```
FORMAT
EVERY ROW
END
```

El 2º formato de esta sección es el más utilizado ya que permite el formateo de los datos de salida. Puede soportar las siguientes cláusulas:

```
FORMAT
        FIRST PAGE HEADER
        PAGE HEADER
        PAGE TRAILER
        ON EVERY ROW (es obligatoria)
        ON LAST ROW
        BEFORE GROUP OF
        AFTER GROUP OF
END
```

- **First page header:** especificación de la cabecera de la 1ª página, cuya sintaxis es:

FIRST PAGE HEADER (acción) es una instrucción o serie de instrucciones que permite imprimir generalmente el título del informe.

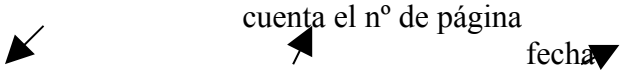
```
FORMAT
        FIRST PAGE HEADER
        PRINT column 5, "listado de clientes"
                column 60, fecha.listado
```

- **Page header:** especificación de la cabecera de cada página, la cláusula permite especificar el contenido de la cabecera de cada página, salvo la 1ª si se ha definido la cláusula FIRST PAGE HEADER.
  Su sintaxis es:
  PAGE HEADER acción

- **Page trailer:** especificación del pie de página, es generalmente usada para poner el número de página.

Su sintaxis es:

PAGE TRAILER acción




```
PRINT column 60, PAGENO, column 70, TODAY
```

- **On every row:** (es obligatoria). La cláusula ON EVERY ROW permite especificar el contenido de cada línea del informe correspondiente a la sección de una línea de la tabla.

Las acciones de esta cláusula son ejecutadas para cada línea del resultado de la sentencia especificada en la cláusula SELECT. Permite fijar la forma y la disposición de cada uno de los valores mostrados.

Algunas de las columnas seleccionadas en la sección SELECT pueden no ser mostradas.

```
ON EVERY ROW acción          no se ponen comillas para que salga el contenido de esos nombre.
        PRINT column 5, nombre,
                column 20, ape1,
                column 40, ape2,
                column 60, ciudad,
```

- **On last row:** permite especificar las acciones que serán ejecutados después de la salida de la última línea del informe.
  Generalmente se utiliza para mostrar los resultados de cálculos hechos sobre todo el informe.

```
ON LAST ROW acción
        PRINT column 20, "Total de pedidos",
                column 60, TOTAL OF cantpedida*precio.
```

Especificación de la salida antes y después de cada grupo.

Cuando la sentencia especificada en la sección SELECT contiene una cláusula ORDER BY, los datos mostrados en el informe serán agrupados según las columnas dadas como argumento en esta cláusula.

Las cláusulas BEFORE, GROUP OF y AFTER GROUP OF permiten efectuar acciones respectivamente antes y después de cada grupo de líneas.

Las acciones de la cláusula BEFORE GROUP OF son ejecutadas al inicio del informe y cada vez que la columna dada como argumento cambia de valor.

```
BEFORE GROUP FO nombre
PRINT column 10, "pedidos clientes"
        column 60,  nombre,
        column 70, ape1,
```

```
SELECT nombre, ape1, ape2, ciudad
        FROM clientes
WHERE ciudad = $ provincia
        ORDER BY nombre
END
```

Si se utiliza la cláusula AFTER GROUP OF, las acciones serán ejecutadas cada vez que la columna asociada como argumento en esta cláusula cambie de valor y al final del informe.

Se utiliza a menudo para mostrar cálculos que se realizan sobre un grupo de líneas.

```
AFTER GROUP FO npedido
        PRINT column 20, "Total del pedido",
                column 40, GROUP TOTAL FO cantped*precio
```

```
AFTER GROUP OF nombre
        PRINT column 10, "Total pedidos del cliente",
                column 40, GROUP FO cantped*precio
```

ACCIONES EJECUTABLES EN UN INFORME:

El generador de informes hace, permite y ejecuta una gran variedad de acciones que pueden clasificarse en tres categorías:

a) Acciones de salida.
b) Expresiones de calculo
c) Estructuras de control.

a) Acciones de salida.

El principal cometido de un informe es imprimir los datos provenientes de la B. De datos o de cálculos efectuados, esto se realiza con la ayuda de la instrucción PRINT que su formato es:

PRINT expresión| columna n| n spaces| ascii n [using "formato" | clipper],...

- **Expresión:** Puede convinar nombre de columnas constantes, funciones y operadores aritméticos, en el caso más normal la expresión se reduce a un nombre de columna o a una constante.

```
ON EVERY ROW
PRINT nombre
PRINT "nombre"
```

- **Columna n:** Especifica que el próximo carácter a describir se imprimirá en la columna n. Contando a partir del margen izquierdo especificado en la sección OUTPUT.

```
PRINT  column 5, nombre, column 30, ape1
```

- **N spaces:** Entraña la salida de n espacios contando a partir de la columna actual.

- **Ascii n:** Permite mostrar un carácter cuyo código ascii es n, se utiliza para sacar caracteres de control que permitan modificar el modo de salida estándar (subrayado, registro, etc.)

	ACTIVAR	DESACTIVAR
NEGRITA ascii 27, ascii 69	ascii27, ascii 70	
SUBRAYADO	ascii 27, ascii45, ascii 49	ascii27, ascii45, ascii49
L. COMPRIMIDA	ascii 27, ascii 15	ascii 18
L. EXPANDIDA	ascii 14	ascii 20

- **Using:** Permite formatear la salida de un valor nº o tipo DATE, el formato de salida se indica en la cadena de caracteres que sigue a la palabra clave USING, esta cadena es una combinación de valores que pueden estar formados por los siguientes caracteres.

- **Tipo fecha:** la misma combinación de caracteres que permita el generador de formatos:
dd, ddd, mm, mmm, yy, yyyy.

- **Valores nº:** (+), (-), (;), (*), (#), (<), (&).

- **(#):** Sustituye una posición del campo numérico que contenga ceros por espacios en blanco, además permite indicar las posiciones que van a ocupar en ese campo numérico.
PRINT column 30, numero2 USING "##"

- **(+):** Permite mostrar un signo (+) si el valor es superior o igual a cero. Un signo (-) en caso contrario.
PRINT column 30, numero2 using"+##"

- **(-):** Se mostrará este signo si el valor es inferior a cero, en caso contrario no se mostrará nada.

- **(;):** Se utilizan como separadores entre la parte entera y la parte decimal de un número.
PRINT column 16, precio USING "####,##"

- **(*):** Sustituye los blancos situados a la izquierda de un número. (*posiciones en blanco, ejemplo un cheque)*
PRINT column 16, precio USING "**1.000**
- **(£):** Este carácter se muestra a la izquierda del número, lo mismo que los signos (+ ,-).

- **(<):** Este carácter entraña una justificación del número a la izquierda.
PRINT column 16, precio USING "<<<#"

- **Clipped:** Situada a continuación de un nombre de columna de tipo carácter, permiten la impresión de esta columna suprimiendo todos los espacios situados a la derecha .

- **Skip [n LINES | TO TOP OF PAGE] :** Permite saltar un grupo de líneas antes de escribir la línea siguiente.
- **n LINES:** Entraña un salto de (n líneas) a partir de la línea actual.
- **TO TOP OF PAGE:** Entraña un salto a la 1ª línea de la página siguiente, esta opción no puede ser usada ni en las cabeceras de página ni en el pie de página.

- **Need:** En ciertos casos se desea que un grupo de líneas sea sacado en la misma página, esta acción permite cumplir esa función. Su formato es el siguiente:

    NEED n LINES

Significa que las (n líneas) siguientes deben ser sacadas en la misma página. Si el resto de la página actual no es suficiente para albergar estas líneas se pasa a la página siguiente

```
BEFORE NEED 6 lines GROUP OF id
        PRINT column id, "nombre"
```

- **Pause:** permite interrumpir la salida del informe justo hasta que se pulsa la tecla ENTER.

Su formato es el siguiente:

```
        PAUSE ["message"]
```

- **Message:** Es el texto que se muestra cuando se ejecuta la acción, esta acción sólo tiene efecto cuando la salida se hace sobre la pantalla y permite evitar una salida rápida.

- **Print File:** Cuando un texto generalmente bastante largo se tiene que insertar en un informe es deseable escribir este texto en un fichero aparte y después incluirle en el informe.

El formato es el siguiente:

```
PRINT FILE "nombre del fichero"

ON EVERY ROW
        PRINT column 10, "Santander",
        TODAY USING "dd-mm-yy"
PRINT column 10, nombre, column 30, ape1, column  l        50, ape2
                                PRINT column 10, dirección
                                        Column 40, cod. Postal
                                PRINT FILE "carta.cl"
```

EXPRESIONES DE CALCULO

Los datos que componen un informe pueden ser de 3 tipos:

1. Valores provenientes de la base de datos.
2. Constantes.
3. Valores calculados.

Estos últimos son los resultados de las expresiones formuladas en combinación con los valores provenientes de la Base de Datos, variables, constantes, operadores aritméticos y funciones.

Estas expresiones pueden utilizarse bien con la opción PRINT o con la acción LET, cuyo formato es el siguiente:

LET variable = expresión

La variable obligatoriamente debe estar definida en la sección DEFINE.

Los operadores aritméticos que se pueden utilizar son:

+, -, *, /, **

FUNCIONES MATEMÁTICAS:

El generador de informes .ACE dispone de ciertas funciones matemáticas que permiten efectuar cálculos tales como el TOTAL, la MEDIA, MAX., MIN., etc.

Estas funciones pueden aplicarse a todas las líneas seleccionadas a un grupo de ellas o a un subconjunto de líneas que verifican una condición dada.

El formato de la utilización de estas funciones es el siguiente:

```
[GROUP]
    [COUNT | PERCENT]
    [TOTAL | AVG | MIN. | MAX. | OF EXPRESIÓN]
    [WHERE condición]
```

ej.

```
AFTER GROUP OF npedidos
PRINT column 50, "Total Pedido"
PRINT column 70, GROUP OF cantidad*precio
USING "<<<<<##,Ptas."
WHERE npedido=2
```

(*el mismo pero de otra forma).*

```
AFTER GROUP OF idcliente
    PRINT column 50, "Total Cliente",
        Column 70, GROUP TOTAL OF cantidad*precio
ON LAST ROW
    PRINT column 40, "Total General"
```

Column 60, TOTAL OF cantidad*precio

FUNCIONES DE MANIPULACIÓN DE FECHA:

- **DATE (expresión):** convierte la expresión en un valor tipo DATE (fecha).

PRINT column 10, DATE (28-02-98)

- **DAY (expresión):** Devuelve el numero del día del mes de la fecha dada como argumento.

- **MONTH (expresión):** Devuelve el numero del mes de la fecha dada como argumento.

- **YEAR (expresión):** Devuelve un número representando el año de la fecha dada como argumento.

- **WEEKDAY (expresión):** Devuelve un numero entero correspondiente al día de la semana de la fecha dada como argumento. El 0 corresponde al domingo, el 1 al lunes, ..., y el 6 al sábado.

VARIABLES PRE-DEFINIDAS:

Además de las variables definidas en la sección define disponemos de un cierto número de ellas cuyo contenido es modificado automáticamente por informes. El usuario no debe ni definir ni asignar valores a estas variables.

- **PAGENO:** Contiene él numero de página actual.

```
PAGE TRAILER
        PRINT column 70, PAGENO
        USING "pagina <<##"
```

- **LINENO:** Contiene el numero de la línea actual en la página.

- **TINE:** Contiene la hora actual bajo la siguiente forma:

hh:mm:ss

EXTRUCTURAS DE CONTROL:

Las acciones definidas en las diferentes cláusulas de las acciones FORMAT se pueden ejecutar de un modo secuencial bajo una condición dada o repetidas un cierto número de veces.

La ejecución secuencial es la secuencia normal de ejecución, con ella las instrucciones se ejecutan según han sido codificadas.

Con los diferentes tipos de ejecución debemos de indicar las palabras claves: (BEGIN, END) para definir las acciones múltiples.

- **Estructura condicional.**

En ciertos casos es deseable ejecutar una acción o una serie de acciones cuando una condición dada se cumple, en otros casos la verificación de una condición, entraña la ejecución de un primer grupo

de acciones, mientras que la no verificación de la condición entraña la ejecución de un segundo grupo de acciones, se dice entonces que es una ejecución condicional o alternativa.

El formato de la instrucción que nos permite esta verificación de la condición es la siguiente:

```
IF condición
        THEN acción 1
        ELSE acción 2
```

Donde:
Acción: Es una acción simple o una serie de acciones que serán ejecutadas cuando la condición se verifique o no.

En el caso de una serie de acciones deben de estar delimitadas por las palabras claves (BEGIN, END). Si una instrucción de este tipo es utilizada en una cabecera o un pie de página y tanto el cumplimiento o no de la condición lleva la instrucción PRINT el mismo número de líneas de salida deben de aparecer en cada una de las acciones.

Ej.

```
FIRST PAGE HEADER
        IF count=0
                BEGIN
                        PRINT column 30, "no existen pedidos"
                        SKIP 2 LINES
                END
        ELSE
                BEGIN
                        PRINT column20, "listado de pedidos"
                        SKIP 2 LINES
                END
```

*(cuando solo se hace una instrucción no hace falta poner (begin end) si son más de una hay que hacerlo).

- **<u>Estructura repetitiva:</u>**

El generador de informes permite la ejecución de una acción simple o una serie de acciones un numero determinado de veces o en tanto que una condición es verificada.

Estas posibilidades se ofrecen con ayuda de las instrucciones (FOR, WHILE).

El formato FOR:

```
FOR variable = valor inicial TO valor final [STEP incremento]
        DO acción  ──▶ cuando es una sola acción

                   { BEGIN
```

Cuando hay más Accion1
de una acción Accion2
END

El formato WHILE:

WHILE condición
DO acción

BEGIN
Accion1
Accion2
END

TRANSACCIONES Y ACCESOS CONCURRENTES:

Una transacción es una unidad de lógica de tratamiento que agrupa un conjunto de operaciones elementales, estas operaciones deben ejecutarse en conjunto o ninguna de ellas.

INSERT INTO línea pedido VALUES (100.1,1200.36);
UPDATE artículos SET CANSTOCK=CANSTOCK – 30 WHERE idartículo=200;

CREACION E INICIALIZACION DE UN DIARIO DE TRANSACCIONES:

Un diario de transacciones puede ser creado en el momento de la creación de la Base de Datos.

CREATE DATABASE nombre.dbf WITH LOG IN “Nombre diario transacciones”;

- Cuando tengo la base de datos ya creada y quiero crear un diario de transacciones.

(antes de crear el diario es necesario que la base de datos
CLOSE DATABASE; esté cerrada.
STAR DATABASE nombre.dbf WITH LOG IN “transacción”;

COMANDOS DE TRANSACCIONES:

BEGIN WORK
Instrucciones 1
“ 2
“ 3
COMMIT WORK ⟶ Cuando quiero que las instrucciones se ejecuten
ROLL BACK WORK ⟶ Cuando no quiero que se ejecuten las intrucciones

CONTROL DE CONCURRENCIA:

Informix es un sistema multiusuario que permite a un numero de usuarios acceder al mismo tiempo a una base de datos. En estos casos y para evitar operaciones conflictivas Informix prevee un control de concurrencia que son los siguientes:

1. Bloqueo a nivel de tablas.
2. Bloqueo a nivel de líneas.

1. BLOQUEO DE TABLAS:

Con esta opción se permite impedir el acceso a los datos de una tabla a los demás usuarios o sólo se les permite el acceso en modo lectura. El comando para bloquear una tabla es el siguiente.

LOCK TABLE nombre-tabla → Los demás usuario no tienen ningún acceso.
IN SHARE | EXCLUSIVE MODE;
→ Modo compartido los demás usuarios pueden leer.

Para desbloquear la tabla el formato es:

UNLOCK TABLE nombre-tabla;

2. BLOQUEO DE LÍNEAS:

Los bloqueos de línea se obtienen mediante el comando BEGIN WORK si previamente se ha creado un diario de transacciones.

Cada línea de la tabla afectada en una transacciones está bloqueada mientras dura esa transacción.

Los comandos COMMIT WORK o ROLLBACK WORK liberan todos los bloqueos de la transacción. Cuando se utiliza individualmente una instrucción UPDATE, el bloqueo de líneas a las que haga referencia dicha instrucción es automáticamente efectuado por el sistema.

COPIA DE SEGURIDAD DE LA BASE DE DATOS:

DATABASE EXCLUSIVE
(copiar directorio base de datos en un directorio).

TAR -UV gestion.dbs ⟶ copiar todo en cinta
TAR –UV transa ⟶ copia diario transacciones
STAR DATABASE nombre WITH LOG IN “transa”

RECUPERAR BASE DE DATOS PARA ACTUALIZARLA:

- Para copiar de cinta a directorio.
  TAR –XV gestion.dbs
  TAR –XV home/inf.../gestion.dbs

ROLLFORWARD DATABASE gestión ⟶ actualiza copia de base datos con diario transacciones

VISTAS:

Una vista es una tabla virtual que no tiene existencia física como una tabla base aunque es percibida y tratada como si así fuera. Constituye una ventana sobre los datos de una o varias tablas.

Posee una definición análoga al de una tabla y está almacenada en el diccionario de datos pero no tiene un archivo físico que soporte los datos.

Creación de Vistas:

```
CREATE VIEW nombre-lista [(Lista de columnas)]
AS
SELECT lista de selección
[WITH CHECK OPCION]
```

- La lista de columnas: es opcional, pero se hace obligatoria en el caso de presencia de ambigüedad de columnas. En caso contrario la vista está compuesta de las columnas seleccionadas en la cláusula SELECT.
- La sentencia SELECT no puede contener la cláusula ORDER BY ni el operador UNION.
- La opción WITH CHECK OPCION permite mantener la integridad referencial de los datos.

Ejemplo.

- Crear VISTA cli_san.

```
CREATE VIEW cli_san
AS
SELECT * FROM clientes
WHERE ciudad=”Santander”
```

- Visualizar datos VISTA.

```
SELECT *
FROM cli_san
```

(es lo mismo que si lo haces desde la tabla clientes y realizas una cláusula SELECT)

```
SELECT *
FROM clientes
WHERE ciudad=”santander”
```

SQL EN ORACLE

Oracle es el mayor y mas usado Sistema Manejador de Base de Dato Relacional (RDBMS) en el mundo. La Corporación Oracle ofrece este RDBMS como un producto incorporado a la línea de producción. Además incluye cuatro generaciones de desarrollo de aplicación, herramientas de reportes y utilitarios.

Oracle corre en computadoras personasles (PC), microcomputadoras, mainframes y computadoras con procesamiento paralelo masivo. Soporta unos 17 idiomas, corre automáticamente en más de 80 arquitectura de hardware y software distinto sin tener la necesidad de cambiar una sola línea de código. Esto es porque más el 80% de los códigos internos de Oracle son iguales a los establecidos en todas las plataformas de sistemas operativos.

El manejador de Base de datos ORACLE, surgió a final de los años 70 y principio de los años 80. George Koch y su equipo de tropas de asalto de técnicos fue el primero en desembarcar en el terreno de Oracle en 1982, durante un proceso de evaluación de sistema de gestión de base de datos para una importante aplicación comercial que George estaba diseñando y construyendo. Cuando termino, la evaluación fue descrita en Computer World como el estudio más severo de SGBD que se había hecho nunca. El estudio fue tan riguroso con los vendedores cuyos productos había estudiado George, que la prensa hizo eco de sus palabras en lugares tan distantes como Nueva Zelandia y en publicaciones muy alejadas del campo como el Christian Sciencia Monitor.

Oracle conocida entonces como Relational Software, tenía poco más de 25 empleados en aquel tiempo y solo unos pocos clientes importantes. Sin embargo, cuando se completo el estudio, Oracle fue declarada vencedora. George afirmo que el SGBD Oracle era técnicamente el mejor producto del mercado. Estas declaraciones fueron hecha en una época en la que muy poca gente conocía el significado del término “Relacional”, y los que lo conocían (o creían conocerlo) no tenían muchas cosas favorables que decir de él.

La compañía de Oracle Corporation estaba trabajando entonces para perfeccionar su joven producto, para comprender los tipos de características y funcionalidad que podría hacerlo útil y productivo en el mundo de los negocios. El esfuerzo contribuyo a su refinamiento. Algunas de las características de Oracle, tales como las salidas de SQL*FORMS fueron el resultado de dicho esfuerzo.

A continuación se presenta una serie de notas sobre la sintaxis utilizada para realizar las consultas en Oracle.

Literales (tipo de dato de valor fijo), existen los siguientes tipos:

- Texto: se escribe entre comillas, por ejemplo 'esto es un texto válido'.
- Entero: número acompañado de un +/- con un máximo de 38 dígitos de precisión, por ejemplo -10.

- Número: igual que entero pero con decimales. Puede estar en notación científica, por ejemplo `25e-03`.
- Intervalo: intervalo de tiempo, por ejemplo INTERVAL '10' HOUR.

TIPOS DE DATOS

- **Caracteres (CHAR),** existen 4 tipos:

1. **CHAR:** crea una columna de largo de 1 a 2000 bytes.
2. **NCHAR:** similar al anterior pero el dato queda almacenado con el correspondiente lenguaje de la DB (NLS).
3. **NVARCHAR2:** similar al anterior con un máximo de 4000 bytes.
4. **VARCHAR2:** similar al anterior pero sin NLS.

- **Números (NUMBER)** con el formato NUMBER(p,s) donde p es la cantidad máxima de dígitos con un máximo de 38 y s es la escala de -84 a 127.

Ejemplos:

- 756123.89 NUMBER 756123.89
- 756123.89 NUMBER(9) 756124
- 756123.89 NUMBER(9,1) 756123.9
- 756123.89 NUMBER(7,-2) 756100
- 756123.89 NUMBER(6) y NUMBER(-7,2) exceden la precisión

También existe el tipo FLOAT.

- **Largo (LONG):** para almacenar strings de datos largos, con un máximo de carácteres de hasta 2 gigabytes o 2^31-1 bytes.
- **Fecha (DATE)**
- **RAW y LONG RAW:** datos no interpretados por Oracle.

Otros tipos menos ocupados

- Large Object Datatypes (LOB), permite el almacenamiento de archivos como texto, imagenes, video hasta 4 gigabytes.
- ROWID: almacena la dirección de la ubicación de una columna.
- UROWID: igual a ROWID pero que viene de otra DB que no sea Oracle.
- ANSI, DB2, SQL/DS: tipos de datos de la DB DB2 de IBM.

Además se pueden crear datos creados por el usuario mediante el comando CREATE TYPE que debe tener nombre, atributos y métodos.

Existen comandos que permiten el cambio de un tipo de dato a otro como TO_CHAR(conversión de fecha) o TO_CHAR(conversión de número).

Otro tipo de dato de uso común es NULL para interpretar el vacío.

OPERADORES DE SQL

- **Aritméticos:**

Operador	Propósito	Ejemplo
		`SELECT * FROM orders WHERE qtysold = -1;`
+-	Operador unario, denota si es positiva o negativa una expresión	`SELECT * FROM emp WHERE -sal < 0;`
*/	Multiplica divide, operador binario	`Update emp SET sal = sal * 1.1;`
+-	Suma, resta, binario	`SELECT sal + comm FROM emp WHERE SYSDATE - hiredate > 365;`

- **Concatenadores:** || une strings de datos. Ejemplo: `SELECT 'Name is ' || ename FROM emp;`

- **Comparadores:**

Operador	Propósito	Ejemplo
=	Igualdad.	`SELECT * FROM emp WHERE sal = 1500;`
!=, ^=, <>, ¬=	Desigualdad.	`SELECT * FROM emp WHERE sal != 1500;`
<, >	Mayor, menor.	`SELECT * FROM emp WHERE sal > 1500;` `SELECT * FROM emp WHERE sal < 1500;`

<=, >=	Mayor, menor o igual.	`SELECT * FROM emp WHERE sal >= 1500; SELECT * FROM emp WHERE sal <= 1500;`
IN	Igual a cualquier miembro a probar, igual a =ANY.	`SELECT * FROM emp WHERE job IN ('CLERK','ANALYST'); SELECT * FROM emp WHERE sal IN (SELECT sal FROM emp WHERE deptno = 30);`
NOT IN	Igual a !=ANY.	`SELECT * FROM emp WHERE sal NOT IN (SELECT sal FROM emp WHERE deptno = 30); SELECT * FROM emp WHERE job NOT IN ('CLERK', ANALYST');`
ANY SOME	Compara un valor a cada valor en una lista, precedido por =, !=, >, <, <=, >=.	`SELECT * FROM emp WHERE sal = ANY (SELECT sal FROM emp WHERE deptno = 30);`
ALL	Similar al anterior.	`SELECT * FROM emp WHERE sal >= ALL ( 1400, 3000);`
[NOT] BETWEEN X AND Y	Entre X e Y.	`SELECT * FROM emp WHERE sal BETWEEN 2000 AND 3000;`
EXISTS	Verdadero si una subquery entrega al menos una fila.	`SELECT ename, deptno FROM dept WHERE EXISTS (SELECT * FROM emp WHERE dept.deptno = emp.deptno);`
X [NOT] LIKE Y [ESCAPE 'z']	Verdadero si x está en un patrón y. Dentro de y el carácter "%" calza con	`SELECT * FROM tab1 WHERE col1 LIKE`

	cualquier carácter menos NULL. "_" calza con cualquier carácter simple.	`'A_C/%E%' ESCAPE '/';`

IS [NOT] NULL	Prueba si hay [o no] NULL	`SELECT ename, deptno`
		`FROM emp`
		`WHERE comm IS NULL;`

NOT IN	Igual a !=	`SELECT 'TRUE'`
		`FROM emp`
		`WHERE deptno NOT IN (5,15,null);`
		`Igual a:`
		`deptno != 5 AND deptno != 15 AND deptno != null`

Otro operador importante es LIKE, que busca patrones dentro de un string:

```
SELECT sal
FROM emp
WHERE ename LIKE 'SM%';
```

Este ejemplo entrega sal de los emp que empiezan con SM.

OPERADORES LÓGICOS

Los típicos AND, OR, NOT.

OPERADORES DE UNIÓN

- **UNION:** une queries.
- **UNION ALL:** mismo, incluyendo las que se repiten.
- **INTERSECT:** unión de queries.
- **MINUS:** las filas de la primera query menos los que están en la segunda.

Uso: Query1 Operador_de_union Query2;

Creación de operadores: existe el comando CREATE OPERATOR.

COMANDOS GENÉRICOS DE SQL

Veremos someramente los comandos más ocupados en SQL.

- **Create**

1. TABLE: crea un objeto tabla relacional. El formato es darle un nombre y luego los tipos y tamaños de los datos que va a contener.
2. TRIGGER: crea un disparador antes o después de una determinada acción (se verá con más detalle en otro tutorial).
3. PROCEDURE: crea una función que puede realizar operaciones sobre datos de una DB.
4. VIEW: crea una vista basada en query y que recibe un nombre determinado bajo el que es invocado.
5. MATERIALIZED VIEW: similar al anterior pero permite operaciones mucho más complejas.
6. INDEX: pone índice a algunos tipos de objetos de una DB para un acceso más expedito.

La mayoría de estos comandos pueden ser modificados con ALTER COMANDO.

- **Drop**: permite borrar un objeto de la DB.

Create, drop y alter se clasifican en la categoría de lenguaje de definición de datos.

- **Insert**: permite insertar datos a una tabla ya creada.
- **Select**: el gran comando de queries.
- **Delete**: borra datos existentes en una fila.
- **Update**: actualiza los valores de una tabla.

Estos datos forman parte del lenguaje de manipulación de datos (DML).

Otros comandos como GRANT y REVOKE pertenecen al lenguaje de control de datos (DCL).

Estas tres familias de comandos existen dentro de todo SQL, independientemente de qué sistema de DB se trate.

Más ejemplos...

- **Creación de tablas**

Crearemos la tabla EMPLOYEE con los siguientes comandos:

SQL> CREATE TABLE EMPLOYEE
(EMPNO NUMBER (4) NOT NULL PRIMARY KEY,
NAME CHAR (8) NOT NULL,
JOB CHAR (4) ,
SALARY NUMBER (8,2) NOT NULL,
COMM NUMBER (8,2) ,
DEPTNO NUMBER (4) NOT NULL,
SEX CHAR (1));

Luego revise la definición de la tabla EMPLOYEE con:

SQL> DESCRIBE EMPLOYEE;

En general, usará CREATE TABLE de la siguiente manera:

SQL> CREATE TABLE <table_name>
(column_name data_type [not null],?
PRIMARY KEY (column_name, ?)
FOREIGN KEY (column_name, ?)
REFERENCES table2_name (table2column_name,?));

Donde 2 representa a datos en otra tabla.

También se puede crear una tabla a partir de un query, por ejemplo:

SQL> CREATE TABLE MANAGER AS SELECT EMPNO, NAME FROM EMPLOYEE WHERE JOB = 'Mngr';

Puede alterar la estructura de una tabla de la siguiente manera:

SQL> ALTER TABLE EMPLOYEE ADD TEST CHAR (2));

O puede modificar la definición de un campo:

SQL> ALTER TABLE EMPLOYEE MODIFY (TEST CHAR(2) NOT NULL);

Este cambio puede ser también de claves primarias y foráneas:

SQL>ALTER TABLE <nombre_tabla> ADD PRIMARY KEY (<nombre_columna,? >);

Ingreso de datos

Vía INSERT, por ejemplo:

SQL> INSERT INTO EMPLOYEE VALUES
(106,'Spears','Slsm',3000,NULL,40,'M');

o de otra forma:

SQL> INSERT INTO EMPLOYEE (EMPNO, NAME, SALARY, DEPTNO)
VALUES (107,'Kiel',4000,50);

Puede verificar el contenido de su tabla con **SELECT * FROM <nombre_tabla>**;

Para actualizar un campo en una tabla use el comando **UPDATE** y **SET** combinado con una condición:

SQL> UPDATE EMPLOYEE SET SALARY = 1000 WHERE EMPNO = '214';

Otro ejemplo:

SQL> UPDATE EMPLOYEE SET COMM = 500 WHERE COMM IS NULL;

Borrar datos

Si quiere borrar todos los datos de una tabla hágalo con **DELETE FROM <nombre_tabla>;**

Si quiere borrar sólo algunos campos, ocupe un condicional:

SQL> DELETE FROM EMPLOYEE WHERE EMPNO = 107;

COMMIT y ROLLBACK

Estos comandos no son permanentes y se almacenan temporalmente en un buffer. Para ingresarlos definitivamente a la DB debe ejecutar el comando **COMMIT**. Algunos comandos llevan implícito el COMMIT, estos son QUIT, EXIT, CREATE TABLE, CREATE VIEW, DROP TABLE, DROP VIEW, GRANT, REVOKE y ALTER. Cuando pedimos que nos muestre una tabla, Oracle nos mostrará la tabla actualizada.

Para volver atrás un **COMMIT**, puede usar el comando **ROLLBACK**.

Queries

Todas las queries y subqueries se basan en el uso del comando **SELECT** y condiciones lógicas.

Para ver el contenido de una tabla debe ingresar: **SELECT * FROM <nombre_tabla>;**

Para ver una o varias columnas, debemos ingresar **SELECT <nombre_de_la_columna1, nombre_de_la_columna2,...> FROM <nombre_tabla>;**

Para ordenar alfabéticamente añada un **ORDER BY <nombre_de_la_columna>** al final.

Una condición lógica puede ser agregada después del nombre de la tabla, esto es por ejemplo:

SQL> SELECT NAME, JOB FROM EMPLOYEE

WHERE

SEX = 'F' AND (JOB = 'Slsm' OR JOB = 'Mngr'); <- condición lógica

Otros SELECTs

:

- **SQL> SELECT DISTINCT JOB FROM EMPLOYEE;** <- muestra los valores de la columna JOB que no aparecen más de una vez en la tabla EMPLOYEE
- **SQL> SELECT NAME, EMPNO FROM EMPLOYEE WHERE JOB [NOT] IN ('Slsm','Clrk');** <- muestra NAMEs y EMPNOs de la tabla EMPLOYEE en los que JOB[!]='Slsm' o JOB[!]='Clrk'

La mayoría de los comparadores ya fueron descritos anteriormente en una de las tablas. También se pueden efectuar cálculos entre columnas por fila con los operadores matemáticos

FUNCIONES CON QUERIES

En vez de * o nombre_de_columna, puede efectuar operaciones sobre su **SELECT** como:

- **AVG(**nombre_de_columna**):** promedio de nombre_de_columna.
- **COUNT(*):** cuenta el número de filas.
- **MIN(**nombre_de_columna**):** el menor de nombre_de_columna.
- **MAX(**nombre_de_columna**):** el máximo de nombre_de_columna.
- **SUM(**nombre_de_columna**):** la sumatoria de nombre_de_columna.

Claro que no es siempre de toda la columna, sino más bien que del resultado del query.

Subqueries

Pueden ser definidas recursivamente como queries de queries. El query principal (el que se desplegará en pantalla) es el primero que se ingresa:

SQL> SELECT NAME, SALARY FROM EMPLOYEE

WHERE SALARY = <-query principal

(SELECT MIN(SALARY) FROM EMPLOYEE); <- query segundario

Nos muestra el NAME y SALARY de la tabla EMPLOYEE que tiene un SALARY= (hasta aquí el query principal) al menor SALARY de la tabla EMPLOYEE.

BIBLIOGRAFÍA

- **Oracle 8i SQL Reference Release 2 (8.1.6) A76989-01.**

- **Apuntes de Oracle/SQL J. Warren, Kennesaw State University**

- **SANTOS, E. et al; *"Bases de Datos"***
  Ed. Servicio Publicaciones de la E.U. de Informática, 1998.

- **ELMASRI, R.A.; NAVATHE, S.B.; *"Fundamentos de Sistemas de Bases de Datos"* (3ª Edición).**
  Ed Addison-Wesley 2002

- **DATE, C.J.; *"Introducción a los Sistemas de Bases de Datos"* (Vol.I, 5ª Edición)**

  Ed. AddisonWesley Iberoamericana, 1990.

- **RIVERO CORNELIO, E. et al. *"Introducción al SQL para Usuarios y Programadores"* (2ª edición)**
  Ed. Thomson, 2002

- **DE MIGUEL, A.; PIATTINI, M.; *"Concepción y Diseño de Bases de Datos. Del Modelo E/R al Modelo Relacional"***
  Ed. RaMa, 1993.

- **HURSCH, C.; HURSCH, J.; *"SQL. El Lenguaje de Consulta Estructurado"* (2ª edición)**
  Ed. RaMa, 1998.

- **KORTH, H.F.; SILBERSCHATZ, A.; *"Fundamentos de Bases de Datos"* (4ª edición)**
  Ed. McGrawHill, 2002.

- **ULLMAN, J.D.; *"Principles of Database Systems"* (2ª edición),**
  Ed. Computer Science Press, 1988.