1. Introducción.

Un RDBMS es un Sistema Gestor de Bases de Datos Relacionales. Se trata de software capaz de producir, manipular y gestionar bases de datos de tipo relacional.

Es un software que se antepone a los datos de una base de datos, de modo que cualquier acceso a los datos pasa por una petición al RDBMS que éste gestiona con el fin de realizar la operación más conveniente sobre esa petición.

Prácticamente es un Sistema Operativo diseñado para el control del acceso a los datos. Para conseguir este control, todo RDBMS posee una serie de subsistemas que se encargan de gestionar cada servicio. Algunos de estos subsistemas son:

Sistema de gestión de la memoria. Encargado de decidir que parte de la memoria se dedica a cada tarea del RDBMS. Su función es que haya suficiente memoria para que el RDBMS funcione eficazmente y a la vez nunca dejar menos memoria de la que necesita el Sistema Operativo para que la máquina funcione.

Gestión de Entrada y Salida. Para conseguir que los accesos a los datos sean adecuados.

Procesador de lenguajes. Para interpretar las instrucciones SQL (o de otros lenguajes válidos) que los usuarios lanzan a la base de datos.

Control de procesos. Gestiona los programas en ejecución necesarios para el funcionamiento de la base de datos.

Control de la red. Para gestionar las conexiones a la base de datos desde la red y evitar problemas a la base de datos en caso de desconexión.

Control de transacciones. Permite gestionar las transacciones (series de operaciones que se pueden anular o llevar a cabo al final.

Diccionario De Datos

La gran ventaja de los RDBMS consiste en que permiten gestionar los datos de forma lógica, se utilizan estructuras más abstractas para los datos, a fin de evitar utilizar el complicado entramado físico que posee una base de datos.

El diccionario de datos agrupa los metadatos de una base de datos. En este diccionario aparecen todos los objetos de la base de datos; con su nombre, función, control de acceso (seguridad) y correspondencia física en los archivos de datos.

Cada vez que llega al gestor de bases de datos una petición sobre datos de una base de datos, el RDBMS abre el diccionario de datos para comprobar los metadatos relacionados con la petición y resolver si hay permiso de uso y donde localizar físicamente los datos requeridos.

Conexión A Un Sistema Gestor De Bases De Datos

Normalmente cualquier DBMS funciona como servidor, programa que está en ejecución esperando peticiones de conexión al sistema. En cada intento de conexión el sistema verificará qué usuario intenta conectar y si tiene permiso se produce la conexión.

En la conexión el usuario puede ejecutar peticiones sobre la base de datos en el lenguaje, o lenguajes, que el DBMS sea capaz de traducir. Esto permite centralizar la información ya que el servidor se puede encontrar absolutamente alejado del usuario que intenta acceder. De modo que el usuario puede estar en un ordenador y el servidor en otro.

a. Conexión Local

Se trata de una conexión en la cual el servidor de base de datos y el usuario que intenta conectar están en la misma máquina. No hace falta control de red, pero limita el uso de la base de datos a la máquina en la que el servidor está instalado.

b. Cliente / Servidor

Se trata del método más común de trabajo. El servidor de bases de datos lanza un proceso en la máquina central (servidor) desde la que se gestionan las bases de datos. Este proceso está a la escucha de nuevos usuarios, cuando estos llegan se produce una conexión que permite que el servidor y el cliente se comuniquen.

La ventaja de esta implementación reside en que se permite centralizar el sistema de datos, lo que facilita su control. Por otro lado eso permite una accesibilidad a la base de datos desde distintas máquinas.

Herramientas de los RDBMS.

Para el uso de las bases de datos, los RDBMS proporcionan diversas herramientas, que además tienen finalidades distintas en función de qué tipo de usuario las utiliza:

Herramientas de instalación. Instaladores para facilitar la tarea de realizar la siempre difícil instalación del producto de base de datos.
Herramientas de gestión de red. Que permiten que el gestor de base de datos sea correctamente accesible desde la red, así como gestionar el correcto flujo de información sobre la red que integra al RDBMS.
Herramientas de conexión en el lado del cliente. Los programas que permiten a los usuarios conectar a la base de datos para lanzar las instrucciones que se deseen.
Herramientas de desarrollo. Que facilitan la labor de crear aplicaciones para una base de datos.
Herramientas CASE. Para poder realizar diseños completos de aplicaciones de datos.
Herramientas de administración. Que permiten una más fácil realización de las tareas administrativas.
Herramientas de recuperación en caso de desastre.
Herramientas de copia de seguridad
Herramientas de importación de datos.
Herramientas de globalización
Herramientas de creación de aplicaciones hacia Internet

2. SQL (Structure Query Language)

SQL es el lenguaje de consulta **universal** para bases de datos.

A partir de aquí vamos a tratar los temas relacionados con **SQL ANSI 92,** que es el estándar SQL , ya que luego existen variantes como, por ejemplo, **T-SQL** (Transact-SQL) para Microsoft SQL-SERVER y **PL/SQL** (Procedure Language / SQL) para ORACLE que serán tratados en sus propias opciones.

SQL proporciona métodos para definir la base datos, para manipular la información y para gestionar los permisos de acceso a dicha información.

Para que un gestor de bases de datos sea considerado como relacional, debe soportar **SQL**, independientemente de las características particulares que dicho gestor pueda aportar.

Los mandatos de **SQL** se dividen en tres grandes grupos diferenciados:

DDL (Data Definition Language), es el encargado de la definición de Bases de Datos, tablas, vistas e índices entre otros.

Son comandos propios de este lenguaje:
CREATE TABLE
CREATE INDEX
CREATE VIEW
CREATE SYNONYM

DML (Data Manipulation Language), cuya misión es la manipulación de datos. A través de él podemos seleccionar, insertar, eliminar y actualizar datos. Es la parte que más frecuentemente utilizaremos, y con la que se construyen las consultas.

Son comandos propios de este lenguaje:
SELECT
UPDATE
INSERT
INSERT INTO
DELETE FROM

DCL (Data Control Laguage), encargado de la seguridad de la base de datos, en todo lo referente al control de accesos y privilegios entre los usuarios.

Son comandos propios de este lenguaje:

CREATE USER
GRANT
REVOKE

3. Componentes del lenguaje SQL.

Tipos de datos.

SQL admite una variada gama de tipos de datos para el tratamiento de la información contenida en las tablas. Los tipos de datos pueden ser numéricos (con o sin decimales), alfanuméricos, de fecha o booleanos (si o no). Según el gestor de base de datos que estemos utilizando los tipos de datos varían, pero se reducen básicamente a los expuestos anteriormente, aunque en la actualidad casi todos los gestores de bases de datos soportan un nuevo tipo, el BLOB (Binary Large Object), que es un tipo de datos especial destinado a almacenar archivos, imágenes ...

Dependiendo de cada gestor de bases de datos el nombre que se da a cada uno de estos tipos puede variar. Básicamente tenemos los siguientes tipos de datos.

Numéricos	Alfanuméricos	Fecha	Lógico	BLOB
Integer	char(n)	Date	Bit	Image
Numeric(n.m)	varchar(n,m)	DateTime		Text
Decimal(n,m)				
Float				

Más detalladamente tenemos:

Tipos de datos numéricos		
Tipo	**Definición**	**Bytes**
Integer	Valores enteros con signo.	4
Numeric(n,m)	Números reales de hasta 18 dígitos (con decimales), donde **n** representa el total de dígitos admitidos (normalmente denominado precisión) y **m** el número de posiciones decimales (escala).	5-17
Decimal(n,m)	Igual que el tipo numeric.	5-17
Float	Número de coma flotante, este tipo de datos se suele utilizar para los valores en notación cientifica.	4-8
Tipos de datos alfanuméricos		
Tipo	**Definición**	**Bytes**
char(n)	Almacena de 1 a 255 caracteres alfanuméricos. Este valor viene dado por **n**, y es el tamaño utilizado en disco para almacenar dato. Es decir si defino un campo como char(255), el tamaño real del campo será de 255, aunque el	0-255

	valor solo contenga 100.	
varchar(n)	Igual que el tipo char, con la salvedad que varchar almacena únicamente los bytes que contenga el valor del campo.	0-255

Nota: El tamaño del campo varía en función de cada base de datos, siendo 255 el valor estándar. En realidad el tamaño viene delimitado por el tamaño de las páginas de datos, para SQL Server el límite esta en 8000 bytes (8000 caracteres), siempre y cuando tengamos definido el tamaño de la página de datos a 8K

Tipos de datos fecha		
Tipo	**Definción**	**Bytes**
Date	Almacena fechas, con día, mes y año.	8
Datetime	Almacena fechas con fecha y hora	4

Tipos de datos lógicos		
Tipo	Definición	Bytes
Bit	Tipo bit. Almacena un 0 ó no cero, según las bases de datos serán 1 ó -1. Se aplica la lógica booleana, 0 es falso y no cero verdadero.	1 bit

Tipos de datos BLOB		
Tipo	Definición	Bytes
Image	Almacena imágenes en formato binario, hasta un máximo de 2 Gb de tamaño.	0-2Gb
Text	Almacena texto en formato binario, hasta un máximo de 2 Gb de tamaño.	0-2Gb

Tipos de datos en ORACLE.

Los más comunes son:

NUMBER (*Numérico*): Almacena números enteros o de punto flotante, virtualmente de cualquier longitud, aunque puede ser especificada la precisión (Número de dígitos) y la escala que es la que determina el número de decimales.

```
-- NUMBER [(precisión, escala)]
saldo NUMBER(16,2);
/* Indica que puede almacenar un valor numérico de 16
  posiciones, 2 de ellas decimales. Es decir, 14 enteros y dos decimales o 16 enteros */
```

La precisión indica el número de dígitos (contando los decimales) que

contendrá el número como máximo. Oracle garantiza los datos con precisiones de 1 a 38. La escala indica el máximo de dígitos decimales. Hay que tener en cuenta que una columna definida NUMBER(10,5), podrá contener como máximo cualquier número siempre y cuando el número de dígitos enteros más el número de dígitos decimales no supere 10 (y no 15).

Para definir número enteros, se puede omitir el parámetro s o bien poner un 0 en su lugar.

Se puede especificar una escala negativa, esto lo que hace es redondear el número indicado a las posiciones indicadas en la escala. Por ejemplo un número definido como NUMBER(5,-2), redondeará siempre a centenas. Así si intentamos introducir el valor 1355, en realidad se almacenará 1400.

VARCHAR2 (*Carácter de longitud variable*): Almacena datos de tipo carácter empleando sólo la cantidad necesaria aún cuando la longitud máxima sea mayor.

```
-- VARCHAR2 (longitud_maxima)
nombre VARCHAR2(20);
/* Indica que puede almacenar valores alfanuméricos de hasta 20 posiciones */

/* Cuando la longitud de los datos sea menor de 20 no se rellena con blancos */
```

CHAR (*Caracter*): Almacena datos de tipo carácter con una longitud máxima de 32767 y cuyo valor de longitud por defecto es 1.

```
-- CHAR [(longitud_maxima)]
nombre CHAR(20);
/* Indica que puede almacenar valores alfanuméricos de 20 posiciones*/
```

BOOLEAN (*lógico*): Se emplea para almacenar valores TRUE o FALSE.

```
hay_error BOOLEAN;
```

DATE (*Fecha*): Almacena datos de tipo fecha. Las fechas se almacenan internamente como datos numéricos, por lo que es posible realizar operaciones aritméticas con ellas.

Oracle almacena internamente el siguiente formato: Siglo / Año / Mes / Día / Hora / Minuto / Segundo ; aunque, el formato por defecto de las fechas es: **'DD-MM-YYYY' .**

Tipos de datos binarios

Permiten almacenar información en formato "crudo", valores binarios tal y como se almacenan en el disco duro o como residen en memoria.

Estas columnas se pueden utilizar tanto para almacenar grandes cantidades de datos (hasta 4Gb.), como para almacenar directamente cualquier tipo de fichero (ejecutables, sonidos, videos, fotos, documentos Word, DLLs…) o para transportar datos de una base de datos a otra, ya que el formato binario es el único formato común entre cualquier sistema informático. En **Oracle** se usan los tipos de datos **CLOB** y **NLOB** para almacenar grandes cantidades de datos alfanuméricos.

Atributos de tipo.

Un atributo de tipo PL/SQL es un modificador que puede ser usado para obtener información de un objeto de la base de datos. El atributo **%TYPE** permite conocer el tipo

de una variable, constante o campo de la base de datos. El atributo **%ROWTYPE** permite obtener los tipos de todos los campos de una tabla de la base de datos, de una vista o de un cursor.

```
DECLARE -- declare variables
-- declare record variable that represents a row fetched from the employees table
  emp_rec employees%ROWTYPE; -- declare variable with %ROWTYPE attribute
BEGIN
 SELECT * INTO emp_rec FROM EMPLOYEES WHERE employee_id = 120; -- retrieve record
 DBMS_OUTPUT.PUT_LINE('Employee name: ' || emp_rec.first_name || ' '
          || emp_rec.last_name); -- display
END;
```

Registros

Un registro es una estructura de datos en PL/SQL, almacenados en campos, cada uno de los cuales tiene su propio nombre y tipo y que se tratan como una sola unidad lógica. Los campos de un registro pueden ser inicializados y pueden ser definidos como NOT NULL. Aquellos campos que no sean inicializados explícitamente, se inicializarán a NULL.

La sintaxis general es la siguiente:

TYPE <*nombre*> **IS RECORD**
(
campo <*tipo_datos*> [NULL | NOT NULL]
[,<*tipo_datos*>...]
);

Los registros son un tipo de datos, por lo que podremos declarar variables de dicho tipo de datos.

```
DECLARE
TYPE PAIS IS RECORD
(
        CO_PAIS NUMBER ,
        DESCRIPCION VARCHAR2(50),
        CONTINENTE VARCHAR2(20)
);
```

/* Declara una variable miPAIS de tipo PAIS. Esto significa que la variable miPAIS tendrá los campos ID, DESCRIPCION y CONTINENTE. */

```
miPAIS PAIS;
BEGIN
/* Asignamos valores a los campos de la variable.*/
miPAIS.CO_PAIS := 27;
miPAIS.DESCRIPCION := 'ITALIA';
miPAIS.CONTINENTE := 'EUROPA';
END;
```

Los registros pueden estar anidados. Es decir, un campo de un registro puede ser de un tipo de dato de otro registro.

Pueden asignarse todos los campos de un registro utilizando una sentencia SELECT.

```
SELECT CO_PAIS, DESCRIPCION, CONTINENTE INTO miPAIS
FROM PAISES WHERE CO_PAIS = 27;
```

Colecciones o Tablas de PL/SQL.

Las tablas de PL/SQL o colecciones son tipos de datos que nos permiten almacenar varios valores del mismo tipo de datos.

Una tabla PL/SQL :

a. Es similar a un array
b. Tiene dos componentes: Un índice de tipo BINARY_INTEGER que permite acceder a los elementos en la tabla PL/SQL y una columna de escalares o registros que contiene los valores de la tabla PL/SQL
c. Puede incrementar su tamaño dinámicamente.

La sintaxis general es la siguiente:

```
TYPE <nombre_tipo_tabla> IS TABLE OF
<tipo_datos> [NOT NULL]
INDEX BY BINARY_INTEGER ;
```

Una vez que hemos definido el tipo, podemos declarar variables y asignarle valores.

```
DECLARE
/* Definimos el tipo PAISES como tabla PL/SQL */
TYPE PAISES IS TABLE OF NUMBER  INDEX BY BINARY_INTEGER  ;
/* Declaramos una variable del tipo PAISES */
tPAISES PAISES;
BEGIN
        tPAISES(1) := 1;
        tPAISES(2) := 2;
        tPAISES(3) := 3;
END;
```

No es posible inicializar las tablas en la inicialización. Pero si es posible declarar elementos de una tabla PL/SQL como de tipo registro.

```
DECLARE
TYPE PAIS IS RECORD
(
        CO_PAIS NUMBER  NOT NULL ,
        DESCRIPCION   VARCHAR2(50),
        CONTINENTE VARCHAR2(20)
);
TYPE PAISES IS TABLE OF  PAIS INDEX BY BINARY_INTEGER  ;

tPAISES PAISES;
BEGIN
        tPAISES(1).CO_PAIS := 27;
        tPAISES(1).DESCRIPCION := 'ITALIA';
        tPAISES(1).CONTINENTE := 'EUROPA';
END;
```

Funciones para el manejo de tablas PL/SQL

Cuando trabajamos con tablas de PL podemos utilizar las siguientes funciones:

a. FIRST. Devuelve el menor índice de la tabla. NULL si está vacía.
b. LAST. Devuelve el mayor índice de la tabla. NULL si está vacía.
c. EXISTS(i). Utilizada para saber si en un cierto índice hay almacenado un valor. Devolverá TRUE si en el índice i hay un valor.
d. COUNT. Devuelve el número de elementos de la tabla PL/SQL.
e. PRIOR (n). Devuelve el número del índice anterior a n en la tabla.
f. NEXT (n). Devuelve el número del índice posterior a n en la tabla.
g. TRIM. Borra un elemento del final de la tabla PL/SQL.
h. TRIM(n) borra n elementos del final de la tabla PL/SQL.
i. DELETE. Borra todos los elementos de la tabla PL/SQL.
j. DELETE(n) borra el correspondiente al índice n.
k. DELETE(m,n) borra los elementos entre m y n.

```
DECLARE
        TYPE ARR_CIUDADES  IS TABLE OF VARCHAR2(50)  INDEX BY BINARY_INTEGER;
        misCiudades ARR_CIUDADES;
BEGIN
        misCiudades(1) := 'MADRID';
        misCiudades(2) := 'BILBAO';
        misCiudades(3) := 'MALAGA';
        FOR i IN misCiudades.FIRST..misCiudades.LAST LOOP
                dbms_output.put_line(misCiudades(i));
        END LOOP;
END;
```

ROWID.

Representa una dirección de la base de datos, ocupada por una única fila. El ROWID de una fila es un identificador único para una fila dentro de una base de datos. No hay dos filas con el mismo ROWID. Este tipo de dato sirve para guardar punteros a filas concretas.

Operadores

Los operadores se pueden definir como combinaciones de caracteres que se utilizan tanto para realizar asignaciones como comparaciones entre datos.

Los operadores se dividen en aritméticos, relacionales, lógicos y de concatenación y suelen ser los mismos para todos los SGBD.

Operadores SQL		
Aritméticos	+	Suma
	-	Resta
	*	Producto
	/	División
	** ^	Exponenciación
Relacionales	<	Menor que

	<=	Menor o igual que
	>	Mayor que
	>=	Mayor o igual que
	<> !=	Distinto
	!<	No menor que
	!>	No mayor que
Lógicos	**AND**	Los operadores lógicos permiten comparar expresiones lógicas devolviendo siempre un valor verdadero o falso. Los operadores lógicos se evalúan de izquierda a derecha.
	OR	
	NOT	
Concatenación	**+, ||**	Se emplea para unir datos de tipo alfanumérico. **+** se emplea en Microsoft SQLSERVER y **||** en ORACLE.

Operadores en ORACLE.

Los más comunes son:

Operador de asignación	:= (dos puntos + igual)
Operadores aritméticos	+ (suma) - (resta) * (multiplicación) / (división) ** (exponente)
Operadores relacionales o de comparación	= (igual a) <> (distinto de) < (menor que) > (mayor que) >= (mayor o igual a) <= (menor o igual a)
Operadores lógicos	AND (y lógico) NOT (negación) OR (o lógico)
Operador de concatenación	||

Palabras Clave

Las palabras clave son identificadores con un significado especial para SQL, por lo que no pueden ser utilizadas para otro propósito distinto al que han sido pensadas.

SQL dispone de muy pocas órdenes, pero de múltiples palabras clave, lo que le convierten en un lenguaje sencillo pero tremendamente potente para llevar a cabo su función.

Palabras Clave

ALL	AND	ANY	ASC
AVG	BEGIN	BY	CHAR
CHECK	CLOSE	COUNT	COMMIT
CREATE	CURSOR	DECIMAL	DECLARE
DELETE	DESC	DISTINCT	DEFAULT
EXISTS	FETCH	FLOAT	FOR
FROM	GRANT	GROUP	HAVING
IN	INDEX	INSERT	INTEGER
INTO	LIKE	MAX	MIN
NOT	NUMERIC	ON	OPEN
OR	ORDER	REVOKE	ROLLBACK
SELECT	SET	SUM	TABLE
UNION	UNIQUE	UPDATE	USER
VALUES	VIEW	WHERE	WITH

Funciones Agregadas

Las funciones agregadas proporcionan a SQL utilidades de cálculo sobre los datos de las tablas. Estas funciones se incorporan en las consultas **SELECT** y retornan **un único valor** al operar sobre un grupo de registros.

MAX()	Devuelve el valor máximo.
MIN()	Devuelve el valor mínimo.
SUM()	Devuelve el valor de la suma de los valores del campo.
COUNT()	Devuelve el número de filas que cumplen la condición
AVG()	Devuelve el promedia de los valores del campo

Predicados

Los predicados son condiciones que se indican en clausula **WHERE** de una consulta SQL. La siguiente tabla ilustra los predicados de SQL.

BETWEEN...AND	Comprueba que al valor esta dentro de un intervalo
LIKE	Compara un campo con una cadena alfanumérica. LIKE admite el uso de caracteres comodines
ALL	Señala a todos los elementos de la selección de la consulta

ANY	Indica que la condición se cumplirá si la comparación es cierta para al menos un elemento del conjunto.
EXISTS	Devuelve un valor verdadero si el resultado de una subconsulta devuelve resultados.
IN	Comprueba si un campo se encuentra dentro de un determinado rango. El rango puede ser una sentencia SELECT.

Bibliografía:

www.devjoker.com