4. Realización de consultas.

Class	Bases de datos
Column	Xerach Casanova
Lass Edited time	@Dec 28, 2020 8:53 AM

1. Introducción

SQL es el lenguaje utilizado por la mayoría de las aplicaciones donde se trabaja con datos, para acceder a ellos, crearlos y actualizarlos.

Nació a partir de la publicación " A relational model o data for large shared data banks" de Edgar Frank Codd. IBM creó un primer lenguaje llamado SEQUEL (Structured English Query Language), que con el tiempo evolucionó a SQL (Structured Query Language). En 1979 Relational Software (actualmente Oracle Corporation), saca al mercado la primera implementación comercial de SQL.

En 1992 ANSI e ISO estandarizan SQL definiendo las sentencias básicas de SQL (SQL92 o ANSI-SQL). Todas las bases de datos comerciales cumplen con ese estándar, incluyendo sus propias mejoras.

La primera fase del trabajo comienza con sentencias DDL (Lenguaje de definición de datos), para crear la estructura de la base de datos: tablas.

La siguiente fase es la de manipular los datos con sentencias DML (lenguaje de manipulación de datos). Consta de 4 sentencias básicas: INSERT, DELETE, UPDATE y SELECT.

2. La sentencia SELECT

La sentencia SELECT sirve para recuperar o seleccionar datos, de una o varias tablas. Consta de cuatro partes básicas:

- Cláusula SELECT seguida de la descripción de lo que se desea ver (columnas), separadas por comas simples. Obligatoria.
- Cláusula FROM seguida del nombre de la tabla o tablas de las que proceden las columnas del SELECT. Obligatoria.
- Cláusula WHERE seguida de un criterio de selección condición. Opcional.
- Cláusula ORDER BY seguida de un criterio de ordenación. Opcional.

```
SELECT [ALL | DISTINCT] columna1, columna2, ... FROM tabla1, tabla2, ... WHERE condición1, condición2, ... ORDER BY ordenación;
```

Las cláusulas ALL (muestra todas las filas aunque estén repetidas) y DISTINCT (suprime las filas que tengan igual valor que otras), son opcionales.

2.1. Cláusula SELECT

Se debe tener en cuenta lo siguiente:

- Se pueden nombrar las columnas anteponiendo el nombre de la tabla. Es opcional si no existe el mismo nombre de columna en dos tablas sobre las que se está realizando el SELECT. Ejemplo: NombreTabla.NombreColumna.
- Se usa el asterisco para incluir todas las columnas de una o varias tablas: SELECT * FROM NombreTabla;
- Podemos poner alias a los nombres de las columnas poniendo entre comillas el alias que demos a la columna: SELECT F_Nacimiento "Fecha de Nacimiento" FROM USUARIOS;
- Se puede sustituir el nombre de la columna por constantes, expresiones o funciones SQL: SELECT 4*3/100 "MiExpresión", Password FROM USUARIOS;

2.2. Cláusula FROM

En la cláusula FROM se definen los nombres de las tablas de las que proceden las columnas, si se utiliza más de una tabla, deben aparecer separadas por comas y se le denomina consulta combinada o join, para las cuales se necesita usar la cláusula WHERE. Ejemplo: SELECT LOGIN,NOMBRE,APELLIDOS FROM USUARIOS;

También puedes añadir el nombre del usuario que es propietario de las tablas: USUARIO.TABLA. Ya que a veces las tablas pueden tener el mismo nombre, perteneciendo a usuarios distintos.

Se pueden asociar alias a tablas para abreviar, en este caso no es necesario entrecomillar. Ejemplo: SELECT * FROM USUARIOS U;

2.3. Cláusula WHERE

Sirve para restringir la selección a un subconjunto de filas, para lo cual debemos especificar una condición que deben cumplir aquellos registros que queremos seleccionar.

El criterio de búsqueda puede ser más o menos sencillo y se pueden conjugar operadores de diversos tipos, funciones o expresiones.

```
SELECT nombre, apellidos
FROM USUARIOS
WHERE sexo = 'M';
```

2.4. Cláusula ORDER BY

Se utiliza para especificar un criterio de ordenación en la respuesta a nuestra consulta. Solo se puede ordenar por campos de tipo carárcter, número o fecha.

```
SELECT [ALL | DISTINCT] columna1, columna2, ...
FROM tabla1, tabla2, ...
WHERE condición1, condición2, ...
ORDER BY columna1 [ASC | DESC], columna2 [ASC | DESC], …, columnaN [ASC | DESC];
```

El tipo de ordenación se incluye con las palabras reservadas ASC o DESC. Por defecto la ordenación es ascendente.

Se puede ordenar por más de una columna. También se puede ordenar a través de expresiones creadas con columnas, funciones o constantes.

```
SELECT nombre, apellidos
FROM USUARIOS
ORDER BY apellidos, nombre;
```

También se puede colocar el número de orden del campo por el que quieres ordenar, en vez de indicar el nombre de la columna, si se coloca un número mayor al número de campos en el select, nos dará error.

```
SELECT nombre, apellidos, localidad
FROM usuarios
ORDER BY 3;
```

3. Operadores

Los operadores en SQL se dividen en:

- Relacionales o de comparación.
- Aritméticos.
- De concatenación.
- Lógicos.

3.1. Operadores de comparación

Nos permiten comprar expresiones, que pueden ser valores concretos, campos, variables, etc.

Si el resultado de la comparación es correcto la expresión se considera verdadera, en caso contrario se considera falsa.

OPERADOR	SIGNIFICADO
=	Igualdad.
!=, <>, ^=	Desigualdad. Distinto
<	<
>	Mayor que.
<=	Menor o igual que.
>=	Mayor o igual que.
IN	Igual que cualquiera de los miembros entre paréntesis.
NOT IN	Distinto que cualquiera de los miembros entre paréntesis.
BETWEEN	Entre. Contenido dentro del rango.
NOT BETWEEN	Fuera del rango.
LIKE '_abc%'	Se utiliza sobre todo con textos y permite obtener columnas cuyo valor en un campo cumpla una condición textual. Utiliza una cadena que puede contener los símbolos "%" que sustituye a un conjunto de caracteres o "_" que sustituye a un carácter.
IS NULL	Devuelve verdadero si el valor del campo de la fila que examina es nulo.

NULL significa valor inexistente o desconocido, es tratado distinto a otros valores. Si queremos verificar un valor null, se debe utilizar IS NULL O IS NOT NULL y devolverá verdadero o falso.

Los valores nulos se representan en primer lugar en ORDER BY de manera ascendente y en último lugar en descendente.

```
SELECT nombre FROM EMPLEADOS WHERE SALARIO > 1000;
```

```
SELECT nombre FROM EMPLEADOS WHERE APELLIDO1 LIKE 'R %';
```

Nombres de empleados que empiecen por R.

3.2. Operadores aritméticos y de concatenación

Con los operadores aritméticos es posible obtener salidas en las cuales una columna sea el resultado de un cálculo y no un campo de una tabla.

En este ejemplo seleccionamos el salario aumentado un 5% de los empleados que cobran menos de 1000.

```
SELECT SALARIO*1,05
FROM TRABAJADORES
WHERE SALARIO<=1000;
```

Una expresión aritmética sobre null devuelve null.

Para concatenar cadenas existe el operador de concatenación ||. Oracle puede convertir automáticamente valores numéricos a cadena para una concatenación.

En este ejemplo mostraríamos los dos apellidos juntos:

```
SELECT Nombre, Apellido1 || Apellido2
FROM EMPLEADOS;
```

Podemos poner un alias al resultado de la concatenación para que se utilice como cabecera en la salida. En el siguiente ejemplo, además, colocamos un espacio entre los dos apellidos:

```
SELECT Nombre, Apellido1 || ' ' ||Apellido2 Apellidos
FROM EMPLEADOS;
```

3.3. Operadores lógicos

Se utilizan para comprobar si se cumple una u otra condición, ninguna o varias de ellas.

Operadores lógicos y su significado.

OPERADOR	SIGNIFICADO
AND	Devuelve verdadero si sus expresiones a derecha e izquierda son ambas verdaderas.
OR	Devuelve verdadero si alguna de sus expresiones a derecha o izquierda son verdaderas.
NOT	Invierte la lógica de la expresión que le precede, si la expresión es verdadera devuelve falsa y si es falsa devuelve verdadera.

```
SELECT empleado_dni
FROM HISTORIAL_SALARIAL
WHERE salario <=800 OR salario>2000;
```

3.4. Precedencia

Para saber que expresiones se evalúan primero en una consulta, se debe conocer el orden de precedencia en Oracle. En casos de igualdad se evalúa de izquierda a derecha.

1. multiplicación (*) y división (/) al mismo nivel
2. A continuación sumas (+) y restas (-).
3. Concatenación (| |).
4. Todas las comparaciones (<, >, ...).
5. Después evaluaremos los operadores IS NULL, IS NOT NULL, LIKE Y BETWEEN
6. NOT
7. AND
8. OR

Para variar el orden se debe trabajar con paréntesis.

4. Consultas calculadas

Son operaciones realizadas con algunos campos para obtener información derivada de ellos. Para ello haremos uso de la creación de campos calculados.

Estos campos calculados se obtienen a través de la sentencia SELECT, poniendo a continuación la expresión que queramos. Esta consulta no modifica los valores originales de la columna o de la tabla, solo se muestra una columna nueva, a la cual se le puede añadir un alias añadiendo detrás de la expresión la palabra AS y el alias.

```
SELECT Nombre, Credito, Credito + 25 as CreditoNuevo
FROM USUARIOS;
```

5. Funciones

Las funciones son operaciones que se realizan sobre los datos y que realizan un determinado cálculo. Para ello se necesitan parámetros de entrada o argumentos y en función de estos, se realiza el cálculo de la función utilizada. Los parámetros se especifican entre paréntesis.

Se pueden incluir en las cláusulas SELECT, WHERE Y ORDER BY

```
NombreFunción [(parámetro1, [parámetro2, …)]
```

Se pueden anidar funciones dentro de funciones y existen una gran variedad para cada tipo de datos:

- numéricas
- de cadenas de caracteres
- de manejo de fechas
- de conversión
- otras...

Oracle tiene una tabla en la que se pueden hacer pruebas, esta tabla se llama Dual y contiene un campo llamado DUMMY con una sola fila.

5.1. Funciones numéricas

- **ABS(n)** - Calcula el valor absoluto de n.

```
SELECT ABS(-17) FROM DUAL; -- Resultado: 17
```

- **EXP(n)** - Calcula el exponente en base e del número n.

```
SELECT EXP(2) FROM DUAL; -- Resultado: 7,38
```

- **CEIL(n)** - Calcula el valor entero inmediatamente superior o igual al argumento n.

```
SELECT CEIL(17.4) FROM DUAL; -- Resultado: 18
```

- **FLOOR(n)** - Calcula el valor entero inmediatamente inferior o igual al parámetro n

```
SELECT FLOOR(17.4) FROM DUAL; -- Resultado: 17
```

- **MOD(m,n)** - Calcula el resto resultante de dividir m entre n

```
SELECT MOD(15, 2) FROM DUAL; --Resultado: 1
```

- **POWER(valor, exponente) -** Eleva el valor al exponente indicado

```
SELECT POWER(4, 5) FROM DUAL; -- Resultado: 1024
```

- **ROUND(n, decimales)** - Redondea el número n al siguiente número con el número en decimales que se indican.

```
SELECT ROUND(12.5874, 2) FROM DUAL; -- Resultado: 12.59
```

- **SQRT(n)** - Calcula la raíz cuadrada de n

```
SELECT SQRT(25) FROM DUAL; --Resultado: 5
```

- **TRUNC(m,n)** - Trunca un número a la cantidad de decimales especificada en el segundo argumento, si se omite el segundo argumento se truncan todos los decimales. Si n es negativo, el número es truncado desde la parte entera.

```
SELECT TRUNC(127.4567, 2) FROM DUAL; -- Resultado: 127.45
SELECT TRUNC(4572.5678, -2) FROM DUAL; -- Resultado: 4500
SELECT TRUNC(4572.5678, -1) FROM DUAL; -- Resultado: 4570
SELECT TRUNC(4572.5678) FROM DUAL; -- Resultado: 4572
```

- **SIGN** - Si el argumento n es positivo retorna 1, si es negativo devuelve -1 y si es 0 devuelve 0

```
SELECT SIGN(-23) FROM DUAL; – Resultado: -1
```

5.2. Funciones de cadena de caracteres

Se utilizan para manipular campos de tipo carácter o cadena de caracteres.

- **CHR(n)** - Devuelve el carácter cuyo valor codificado es n.

```
SELECT CHR(81) FROM DUAL; --Resultado: Q
```

- **ASCII(n)** - Devuelve el valor ascii de n.

```
SELECT ASCII('O') FROM DUAL; --Resultado: 79
```

- **CONCAT (cad1, cad2)** - Devuelve las dos cadenas concatenadas o unidas, es equivalente al operador II.

```
SELECT CONCAT('Hola', 'Mundo') FROM DUAL; --Resultado: HolaMundo
```

- **LOWER(cad)** - Devuelve la cadena cad con todos los caracteres en minúsculas.

```
SELECT LOWER('En MInúsculAS') FROM DUAL; --Resultado: en minúsculas
```

- **UPPER(cad)** - Devuelve la cadena cad con todos los caracteres en mayúsculas.

```
SELECT UPPER('En MAyúsculAS') FROM DUAL; --Resultado: EN MAYÚSCULAS
```

- **LOWER Y UPPER** son muy utilizadas, sobre todo para comparar cadenas de caracteres de las que desconocemos si están en mayúsculas o minúsculas:

```
SELECT * FROM JUEGOS WHERE UPPER(NOMBRE)='AJEDREZ';
SELECT * FROM JUEGOS WHERE LOWER(NOMBRE)='ajedrez';
```

- **INITCAP(cad)** - Devuelve la cadena cad con su primer carácter en mayúscula.

```
SELECT INITCAP('hola') FROM DUAL; --Resultado: Hola
```

- **LPAD(cad1, n, cad2)** - Devuelve cad1 con longitud n, ajustada a la derecha, rellenando por la izquierda con cad2.

```
SELECT LPAD('M', 5, '*') FROM DUAL; --Resultado: ****M
```

- **RPAD(cad1, n, cad2)** - Devuelve cad1 con longitud n, ajustada a la izquierda, rellenando por la derecha con cad2.

```
SELECT RPAD('M', 5, '*') FROM DUAL; --Resultado: M****
```

- **REPLACE(cad, ant, nue)** - Devuelve cad en la que cada ocurrencia de la cadena ant ha sido sustituida por la cadena nue

```
SELECT REPLACE('correo@gmail.es', 'es', 'com') FROM DUAL; --Resultado: correo@gmail.com
```

- **SUBSTR(cad, m, n)** - Obtiene una subcadena de una cadena, devuelve la cadena cad compuesta por n caracteres a partir de la posición m.

```
SELECT SUBSTR('1234567', 3, 2) FROM DUAL; --Resultado: 34
```

- **LENGTH(cad)** - Devuelve la longitud de cad.

```
SELECT LENGTH('hola') FROM DUAL; --Resultado: 4
```

- **TRIM(cad)** - Elimina los espacios en blanco a izquierda y derecha, así como los espacios dobles del interior de cad.

```
SELECT TRIM(' Hola de nuevo ') FROM DUAL; --Resultado: Hola de nuevo
```

- **LTRIM(cad)** - Elimina los espacios a la izquierda de cad.

```
SELECT LTRIM(' Hola') FROM DUAL; --Resultado: Hola
```

- **RTRIM(cad)** - Elimina los espacios a la derecha de cad.

```
SELECT RTRIM('Hola ') FROM DUAL; --Resultado: Hola
```

- **INSTR(cad, cadBuscada [, posInicial [, nAparición]])** - Obtiene la posición en la que se encuentra la cadena buscada en la cadena inicial cad. Se puede comenzar a buscar a partir de una posición inicial concreta e incluso indicar el número de aparición de la cadena buscada. Si no encuentra nada devuelve cero.

```
SELECT INSTR('usuarios', 'u') FROM DUAL; --Resultado: 1
SELECT INSTR('usuarios', 'u', 2) FROM DUAL; --Resultado: 3
SELECT INSTR('usuarios', 'u', 2, 2) FROM DUAL; --Resultado: 0
```

5.2. Funciones de manejo de fechas

Oracle tiene dos tipos de datos para manejar fechas:

- DATE almacena fechas concretas incluyendo a veces la hora.
- TIMESTAMP almacena un instante de tiempo más concreto, hasta fracciones de segundo.

Se pueden realizar operaciones numéricas con fechas:

- Sumar números suma días, si ese número tiene días, se suman días, horas, minutos y segundos.
- Restar números.

- La diferencia o resta entre dos fecha nos dará el número de días entre esas fechas.

En Oracle tenemos las siguientes funciones

- **SYSDATE** - Devuelve fecha y hora actuales

```
SELECT SYSDATE FROM DUAL; --Resultado: 15/08/20
```

- **SYSTIMESTAMP** - Devuelve fecha y hora actuales en formato TIMESTAMP

```
SELECT SYSTIMESTAMP FROM DUAL; --Resultado: 15/08/20 11:40:41,969000 +02:00
```

- **ADD_MONTHS(fecha, n)** - Añade a la fecha el número de meses indicado con n

```
SELECT ADD_MONTHS('27/07/11', 5) FROM DUAL; --Resultado: 27/12/11
```

- **MONTHS_BETWEEN(fecha1, fecha2)** - Devuelve el número de meses que hay entre fecha1 y fecha 2.

```
SELECT MONTHS_BETWEEN('12/07/11','12/03/11') FROM DUAL; --Resultado: 4
```

- **LAST_DAY(fecha)** - Devuelve el último día del mes al que pertenece la fecha en tipo DATE

```
SELECT LAST_DAY('27/07/11') FROM DUAL; --Resultado: 31/07/11
```

- **NEXT_DAY(fecha, d)** - Indica el día que corresponde si añadimos a la fecha el día d, puede ser texto o número de la semana dependiendo de la configuración.

```
SELECT NEXT_DAY('31/12/11','LUNES') FROM DUAL; --Resultado: 02/01/12
```

- **EXTRACT(valor FROM fecha)** - Extrae un valor de una fecha concreta. El valor puede ser day, month, year, hours, etc...

```
SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL; --Resultado: 8
```

En Oracle los operadores aritméticos de más y menos se pueden emplear en fechas:

```
SELECT SYSDATE – 5 FROM DUAL; -- Devuelve la fecha correspondiente a 5 días antes de la fecha actual
```

```
SQL> SELECT SYSDATE HOY FROM DUAL;

HOY
--------
15/08/20

SQL> SELECT SYSTIMESTAMP INSTANTE FROM DUAL;

INSTANTE
---------------------------------------------------------------------------
15/08/20 11:53:47,879000 +02:00

SQL> SELECT ADD_MONTHS('27/07/20', 5) FROM DUAL;

ADD_MONT
--------
27/12/20

SQL> SELECT MONTHS_BETWEEN('12/07/20','12/03/20') FROM DUAL;

MONTHS_BETWEEN('12/07/20','12/03/20')
-------------------------------------
                                    4

SQL> SELECT LAST_DAY('27/07/20') FROM DUAL;

LAST_DAY
--------
31/07/20

SQL> SELECT NEXT_DAY('31/12/20','LUNES') FROM DUAL;

NEXT_DAY
--------
04/01/21

SQL> SELECT EXTRACT(MONTH FROM SYSDATE) FROM DUAL;

EXTRACT(MONTHFROMSYSDATE)
-------------------------
                        8

SQL> SELECT SYSDATE - 5 FROM DUAL;

SYSDATE-
--------
10/08/20

SQL>
```

5.4. Funciones de conversión

Se puede pasar un tipo de dato a otro, Oracle convierte automáticamente datos de manera que el resultado de una expresión tenga sentido, pero a veces debemos realizar conversiones de modo explícito:

- **TO_NUMBER(Cad_Formato)** - Convierte textos en números con formato:

Formatos para números y su significado.

Símbolo	Significado
9	Posiciones numéricas. Si el número que se quiere visualizar contiene menos dígitos de los que se especifican en el formato, se rellena con blancos.
0	Visualiza ceros por la izquierda hasta completar la longitud del formato especificado.
$	Antepone el signo de dólar al número.
L	Coloca en la posición donde se incluya, el símbolo de la moneda local (se puede configurar en la base de datos mediante el parámetro NSL_CURRENCY)
S	Aparecerá el símbolo del signo.
D	Posición del símbolo decimal, que en español es la coma.
G	Posición del separador de grupo, que en español es el punto.

- **TO_CHAR(d, formato)** - Convierte un número o fecha d a cadena de caracteres. Se suele utilizar con fechas.
- **TO_DATE(cad, formato)** - Convierte textos a fechas con el formato que queramos.

Para ambas funciones con fechas usamos los siguientes formatos:

Formatos para fechas y su significado.

Símbolo	Significado
YY	Año en formato de dos cifras
YYYY	Año en formato de cuatro cifras
MM	Mes en formato de dos cifras
MON	Las tres primeras letras del mes
MONTH	Nombre completo del mes
DY	Día de la semana en tres letras
DAY	Día completo de la semana
DD	Día en formato de dos cifras
D	Día de la semana del 1 al 7
Q	Semestre
WW	Semana del año
AM PM	Indicador a.m. Indicador p.m.
HH12 HH24	Hora de 1 a 12 Hora de 0 a 23
MI	Minutos de 0 a 59
SS SSSS	Segundos dentro del minuto Segundos dentro desde las 0 horas

5.2. Otras funciones: NVL y DECODE

Cualquier operación con NULL devuelve NULL, de esta manera, cuando se divide por cero, también se devuelve nulo.

El resultado de una función también puede dar nulo. Las funciones con nulos nos permiten hacer algo en caso de que aparezca un valor nulo.

- **NVL(valor, expr1)** - Si el valor es nulo, devuelve expr1, el cual debe ser del mismo tipo que el valor.

```
SELECT NVL(correo, 'No tiene correo') FROM USUARIOS;
```

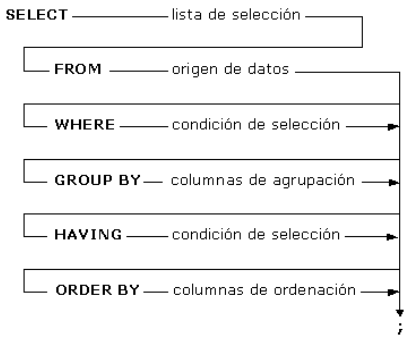
- **DECODE(expr1, cond1, valor1 [, cond2, valor2,...], default)** - Evalúa una expresión expr1 y si se cumple la primera condición cond1 devuelve el valor1, en caso contrario evalúa la siguiente condición hasta que una de ellas se cumpla, si no, devuelve el valor por defecto.

6. Consultas de resumen

La sentencia SELECT nos permite obtener resúmenes de datos de modo vertical con las cláusulas específicas GROUP BY y HAVING. Además tenemos unas funciones llamadas de agrupamiento o de agregado que indican que cálculos queremos realizar sobre la columna, las cuales toman un grupo de datos de una columna y producen un único dato que resume al grupo

El resultado de estas consultas no corresponde a ningún valor de la tabla, sino a un total calculado sobre los datos de la tabla.

El simple hecho de utilizar una función de agregado en una consulta la convierte en consulta de resumen.



Las funciones tienen una estructura muy parecida: FUNCIÓN ([ALL | DISTINCT]) Expresión:

- **ALL** - toma todos los valores de la columna (valor por defecto).
- **DISTINCT** - Se consideran las repeticiones del mismo valor como uno solo.
- El grupo de valores sobre el que se actúa determina el resultado de la expresión con el nombre de una columna o una expresión basada en una o varias columnas. En la expresión no puede aparecer una función de agregado o una subconsulta.
- Todas las funciones aplicadas a filas del origen de datos se aplican una vez ejecutada la cláusula WHERE si existiera.
- Todas las funciones excepto COUNT ignoran valores NULL.
- No se pueden anidar funciones de este tipo. Podemos encontrarlas dentro de una lista de selección en cualquier sitio donde pueda aparecer el nombre de una columna.
- No se pueden mezclar funciones de columna con nombres de columna ordinarias.

6.1. Funciones de agregado: SUM y COUNT

Son funciones para sumar los datos de una columna o contar sus filas.

- **Función SUM**

  SUM([ALL|DISTINCT] expresión) - Devuelve la suma de los valores de la expresión, se utiliza en columnas numéricas, el resultado es del mismo tipo aunque puede tener una precisión mayor.

```
SELECT SUM( credito) FROM Usuarios;
```

- **Función COUNT**

  COUNT([ALL|DISTINCT] expresión) - Cuenta los elementos de un campo. Expresión es el nombre del campo que queremos contar. Los operandos de expresión pueden incluir el nombre del campo, una constante, una función o el carácter *.

  Puede contar cualquier tipo de datos, ya que no tiene en cuenta los valores almacenados. No cuenta los valores NULL a menos que la expresión tenga el comodín asterisco.

```
SELECT COUNT(nombre) FROM Usuarios;
SELECT COUNT(*) FROM Usuarios;
```

6.2. Funciones de agregado: MIN y MAX

- **Función MIN**

  MIN ([ALL| DISTINCT] expresión)

  Devuelve el valor mínimo sin contar valores nulos. En expresión se incluye el nombre de una tabla, constante o función, pero no otras funciones.

```
SELECT MIN(credito) FROM Usuarios;
```

- **Función MAX:**

  MAX ([ALL| DISTINCT] expresión)

  Exactamente el mismo funcionamiento que MIN, pero devolviendo el valor máximo.

6.3. Funciones de agregado: AVG, VAR y STDEV

Obtienen datos estadísticos a partir de los datos de nuestra base de datos.

- **Función AVG:**

  AVG ([ALL| DISTINCT] expresión)

  Devuelve el promedio o media de los valores de un grupo, se omiten los valores null. La expresión puede ser nombre de columna o expresión basada en una o varias columnas de la tabla. Se aplica a campos numéricos y el resultado puede variar según necesidades del sistemas para representar el valor.

```
SELECT AVG(CREDITO) FROM USUARIOS;
```

- **Función VAR**

  VAR ([ALL| DISTINCT] expresión)

  Devuelve la varianza estadística de todos los valores de la expresión, solo admite columnas numéricas y omite valores null.

- **Función STDEV**

  STDEV ([ALL| DISTINCT] expresión)

  Devuelve la desviación típica estadística de todos los valores de la expresión. solo admite columnas numéricas y omite valores null.

7. Agrupamiento de registros

En muchas ocasiones, utilizaremos consultas de resumen agrupados según un determinado campo.

En estos casos en lugar de una única fila de resultados, necesitaremos una fila por cada agrupamiento.

Por ejemplo, de una tabla EMPLEADOS en la que se guarda sueldo y actividad, podemos obtener el valor medio del sueldo en función de la actividad realizada en la empresa.

Estos subtotales se obtienen con la cláusula GROUP BY y podemos poner condiciones a esos grupos con la cláusla HAVING.

```
SELECT columna1, columna2, ...
FROM tabla1, tabla2, ...
[WHERE condición1, condición2, …]
[GROUP BY columna1, columna2, …
[HAVING condición ]]
[ORDER BY ordenación];
```

En la cláusula GROUP BY colocamos las columnas por las que vamos a agrupar y en la cláusula HAVING se especifica la condición que han de cumplir los grupos. El orden de las cláusulas siempre será:

1. WHERE - Filtramos filas a partir de las condiciones que pongamos.
2. GROUP BY - Crea una tabla de grupos nueva.
3. HAVING - Filtra los grupos.
4. ORDER BY - Ordena la salida.

Las columnas que aparecen en SELECT y no aparecen en GROUP BY deben tener una orden de agrupamiento para que no se produzca error.

```
SELECT provincia, SUM(credito) FROM Usuarios GROUP BY provincia;
```

Obtenemos la suma de créditos de nuestros usuarios agrupados por provincia. Si estuviéramos interesados en la suma de créditos agrupados por provincia pero únicamente de las provincias de Sevilla y Badajoz nos quedaría:

```
SELECT provincia, SUM(credito) FROM Usuarios
GROUP BY provincia
HAVING UPPER(provincia) = 'SEVILLA' OR UPPER(provincia)= 'BADAJOZ';
```

8. Consultas multitablas

Es frecuente consultar datos que se encuentren en distintas tablas en la misma sentencia SELECT, lo cual permite realizar distintas operaciones como son:

- La composición interna.
- La composición externa.

```
SELECT tabla1.columna1, tabla1.columna2, …, tabla2.columna1, tabla2.columna2, …
FROM tabla1
     [CROSS JOIN tabla2] |
     [NATURAL JOIN tabla2] |
     [JOIN tabla2 USING (columna) |
     [JOIN tabla2 ON (tabla1.columna=tabla2.columna)] |
     [LEFT | RIGTH | FULL OUTER JOIN tabla2 ON (tabla1.columna=tabla2.columna)]
```

Por ejemplo: Disponemos de una tabla USUARIOS cuya clave principal es Login y está relacionada con la tabla PARTIDAS a través del campo Cod_Creador_Partida. Si queremos obtener el nombre de los usuarios y las horas de las partidas de cada jugador, necesitamos coger los datos de ambas tablas (las horas se guardan en la tabla partida), debiendo coger filas de una y otra tabla. También podríamos tener una tabla de usuarios en servidores distintos y quisiéramos unirlas.

8.1. Composiciones internas

Cuando combinamos dos o más tablas, el resultado es un producto cartesiano, dando como resultado la combinación de todas las filas de esas dos tablas, relaciona una fila de una tabla con todas y cada una de las filas de otra tabla, aunque no tengan relación ninguna.

Se obtiene poniendo en la cláusula FROM las tablas que queramos poner, separadas por comas.

Hay que tener en cuenta que el resultado es la suma de todas las combinaciones posibles y nos podemos encontrar con una operación muy costosa a medida que se aumenta el número de tablas o de filas.

Es necesario discriminar para que aparezcan filas de una tabla que estén relacionadas con la otra (asociar tablas - JOIN)

Lo importante en las composiciones internas es emparejar campos que han de tener valores iguales, con las siguientes reglas:

- Se combinan tantas tablas como se desee.
- El criterio de combinación puede estar formado por más de una pareja de columnas.
- La cláusula SELECT puede citar columnas de ambas tablas, condicionen o no la combinación.
- Si hay columnas del mismo nombre en distintas tablas, se identifican especificando la tabla de procedencia seguida de un punto o utilizando alias.

Las columnas relacionadas de la cláusula WHERE se denominan columnas de join o emparejamiento. No tienen que estar incluidas en la lista de selección. Emparejamos tablas que estén relacionadas entre sí utilizando la

clave principal y la ajena.

NombreTabla1. Camporelacionado1 = NombreTabla2.Camporelacionado2.

También se puede combinar una tabla consigo misma, pero se debe crear un alias a uno de los nombres de la tabla que se va a repetir.

En el siguiente ejemplo obtenemos el historial laboral de los empleados, incluyendo nombre y apellidos, fecha en la que entraron a trabajar y fecha de fin de trabajo:

```
SELECT Nombre, Apellido1, Apellido2, Fecha_inicio, Fecha_fin
FROM EMPLEADOS, HISTORIAL_LABORAL
WHERE HISTORIAL_LABORAL.Empleado_DNI= EMPLEADOS.DNI;
```

Y en la siguiente obtenemos el historial con nombres de departamento, nombre y apellidos del empleado.

```
SELECT Nombre_Dpto, Nombre, Apellido1, Apellido2
FROM DEPARTAMENTOS, EMPLEADOS, HISTORIAL_LABORAL
WHERE EMPLEADOS.DNI= HISTORIAL_LABORAL. EMPLEADO_DNI
AND HISTORIAL_LABORAL.DPTO_COD = DEPARTAMENTOS. DPTO_COD;
```

8.2. Composiciones externas

Sirven para seleccionar filas de una tabla aunque no tengan correspondencia con las filas de otra.

Por ejemplo. En una base de datos donde tenemos empleados de la empresa (Cod_empleado, Nombre, Apellidos, salario y Cod_dpto) por otro lado los departamentos (Codigo_dep, Nombre), en un momento dado se incluyen varios departamentos más a los que todavía no se le han asignado empleados. Si tenemos que obtener un informe de los empleados por departamento y queremos que aparezcan los datos de esos departamentos aunque no tengan empleados usamos composiciones externas.

Para ello se añade un signo (+) entre paréntesis en la igualdad entre campos que ponemos en la cláusula WHERE. El carácter (+) irá detrás del nombre de tabla en la que deseamos aceptar valores nulos.

En el ejemplo sería: Cod_dpto (+)= Codigo_dep (en la tabla empleado es donde aparecerán valores nulos.

8.3. Composiciones en la versión SQL99

CROSS JOIN. Crea un producto cartesiano de las filas de ambas tablas, olvidándonos de la cláusula WHERE

NATURAL JOIN: Detecta automáticamente las claves de unión, se requiere que el nombre de unión sea el mismo en cada tabla. Funciona aunque no estén definidas las claves primarias o ajenas.

JOIN USING: permite establecer relaciones indicando que campo o campos comunes se quieren utilizar, si no queremos que se relacionen todos.

JOIN ON: une tablas en las que los nombres de columna no coinciden en ambas tablas o se necesita establecer asociaciones más complicadas.

OUTER JOIN. Elimina el uso del signo (+) para composiciones externas.

LEFT OUTER JOIN. Composición externa izquierda, todas las filas de la tabla de la izquierda se devuelven, aunque no haya ninguna columna correspondiente en la tabla combinada.

RIGHT OUTER JOIN. Composición externa derecha.

FULL OUTER JOIN. Se devuelven todas las filas de los campos no relacionados de ambas tablas.

Por ejemplo, obtenemos el historial laboral de los empleados: nombre y apellidos, la fecha en la que entraron a trabajar y la fecha de fin de trabajo si no continúan en la empresa. Es una composición interna con JOIN ON

```
SELECT E.Nombre, E.Apellido1, E.Apellido2, H.Fecha_inicio, H.Fecha_fin
FROM EMPLEADOS E JOIN HISTORIAL_LABORAL H ON (H.Empleado_DNI= E.DNI);
```

En este otro ejemplo, obtenemos los distintos departamentos y sus jefes con sus datos personales, deben aparecer los departamentos aunque no tengan asignados jefes. Es una composición externa con OUTER JOIN.

```
SELECT  D.NOMBRE_DPTO, D.JEFE, E.NOMBRE, E.APELLIDO1, E.APELLIDO2
FROM DEPARTAMENTOS D LEFT OUTER JOIN EMPLEADOS E ON ( D.JEFE = E.DNI);
```

9. Otras consultas multitablas: Unión, intersección y la diferencia de consultas

UNION: Combina filas de un primer SELECT con otro SELECT, desapareciendo las filas duplicadas. Ejemplo: Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania.

```
SELECT NombreCia, Ciudad FROM  PROVEEDORES WHERE Pais = 'Alemania'
UNION
SELECT NombreCia, Ciudad  FROM CLIENTES WHERE Pais = 'Alemania';
```

INTERSECT: examina las filas de dos SELECT y devuelve aquellas que aparezcan en ambos conjuntos. Las filas duplicadas se eliminan. Ejemplo: Una academia de idiomas da clases de inglés, frances y portugues; almacena los datos de los alumnos en tres tablas distintas una llamada "ingles", en una tabla denominada "frances" y los que aprenden portugues en la tabla "portugues". La academia necesita el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles información sobre los exámenes.

```
SELECT nombre, domicilio FROM ingles INTERSECT
SELECT nombre, domicilio FROM frances INTERSECT
SELECT nombre, domicilio FROM portugues;
```

MINUS: devuelve aquellas filas que están en el primer SELECT pero no en el segundo. Las filas del primer SELECT se reducen antes de comenzar la comparación. Ejemplo: Ahora la academia necesita el nombre y domicilio solo de todos los alumnos que cursan inglés (no quiere a los que ya cursan portugués pues va a enviar publicidad referente al curso de portugués).

```
SELECT nombre, domicilio FROM INGLES
MINUS
SELECT nombre,domicilio FROM PORTUGUES;
```

Se utilizan en los dos SELECT el mismo número y tipo de columnas y en el mismo orden.

10. Subconsultas

A veces se deben utilizar en consultas resultados de otras a las que llamamos subconsulta o consulta subordinada:

```
SELECT listaExpr
FROM tabla
WHERE expresión_o_columna OPERADOR
(SELECT expresion_o_columna
FROM tabla);
```

Puede ir dentro de WHERE, HAVING O FROM.

- >, <, >=, <=, !=, =, IN. Las subconsultas que utilizan estos operadores devuelven un único valor.

Ejemplo: Obtenemos el nombre de los empleados y el sueldo de aquellos que cobran menos que Ana, si hay más de un empleado llamado Ana, se devuelve error.

```
SELECT Nombre, salario
FROM EMPLEADOS
WHERE salario <
(SELECT salario FROM EMPLEADOS
WHERE Nombre= 'Ana');
```

Si necesitamos que la subconsulta devuelva más de un valor y queremos comparar el campo con todos esos valores se utilizan palabras reservadas:

- **ANY.** Compara con cualquier fila de la consulta. La instrucción es válida cuando un registro de la subconsulta permite que la comparación sea cierta.
- **ALL.** Compara con todas las filas de la consulta y será válida si es cierta la comparación con todas las filas.
- **IN.** No utiliza comparador, comprueba que el valor se encuentra en la subconsulta.
- **NOT IN.** Comprueba que un valor no se encuentre en la subconsulta.

Ejemplo. En la siguiente consulta obtenemos al empleado que menos cobra:

```
SELECT nombre, salario
FROM EMPLEADOS
WHERE salario <= ALL (SELECT salario FROM EMPLEADOS);
```

y en esa misma consulta se podría realizar utilizando función de agregado o colectiva MIN de la misma forma.

```
SELECT nombre, salario
FROM EMPLEADOS
WHERE salario =  (SELECT MIN(salario) FROM EMPLEADOS);
```