Manual SQL

***Nivel Básico / Intermedio***





***Manual SQL***

Contenido

[Introducción a lenguaje SQL 4](#_Toc101610991)

[¿Que es SQL? 4](#_Toc101610992)

[Componentes 4](#_Toc101610993)

[Comandos 4](#_Toc101610994)

[Comandos DLL 4](#_Toc101610995)

[Comandos DML 5](#_Toc101610996)

[DDL - Lenguaje de declaración de datos. 5](#_Toc101610997)

[El comando CREATE TABLE. 5](#_Toc101610998)

[El comando DROP TABLE. 5](#_Toc101610999)

[El comando ALTER. 6](#_Toc101611000)

[Comentarios 6](#_Toc101611001)

[Scripts para crear tablas 7](#_Toc101611002)

[Scripts para crear índices 7](#_Toc101611003)

[DML – Lenguaje de manipulación de datos 7](#_Toc101611004)

[Cláusulas 7](#_Toc101611005)

[Operadores lógicos 8](#_Toc101611006)

[Operadores de Comparación 8](#_Toc101611007)

[Funciones de Agregado 8](#_Toc101611008)

[Consultas Simples 8](#_Toc101611009)

[La sentencia SELECT y FROM 8](#_Toc101611010)

[Alias 9](#_Toc101611011)

[La cláusula WHERE 9](#_Toc101611012)

[Comodines 10](#_Toc101611013)

[La cláusula ORDER BY 11](#_Toc101611014)

[ALL 11](#_Toc101611015)

[Consultas Multitablas 11](#_Toc101611016)

[Unión de tablas 11](#_Toc101611017)

[Producto cartesiano 12](#_Toc101611018)

[El outer join 12](#_Toc101611019)

[El operador UNION 12](#_Toc101611020)

[La cláusula GROUP BY 13](#_Toc101611021)

[La cláusula HAVING 14](#_Toc101611022)

[Subconsultas 14](#_Toc101611023)

[Subconsultas en Cláusula WHERE 14](#_Toc101611024)

[Referencias Externas 15](#_Toc101611025)

[Operador EXISTS 15](#_Toc101611026)

[Subconsultas en Cláusula FROM 15](#_Toc101611027)

[Subconsultas en Cláusula SELECT 16](#_Toc101611028)

[DML – Lenguaje de manipulación de datos. 17](#_Toc101611029)

[SELECT 17](#_Toc101611030)

[INSERT 18](#_Toc101611031)

[UPDATE 19](#_Toc101611032)

[MERGE 20](#_Toc101611033)

[DELETE 20](#_Toc101611034)

[Funciones SQL predefinidas 21](#_Toc101611035)

[TO\_DATE 21](#_Toc101611036)

[TO\_CHAR 21](#_Toc101611037)

[LAST\_DAY 23](#_Toc101611038)

[ADD\_MONTHS 23](#_Toc101611039)

[INITCAP 23](#_Toc101611040)

[LOWER 23](#_Toc101611041)

[UPPER 24](#_Toc101611042)

[DECODE 24](#_Toc101611043)

[CASE 24](#_Toc101611044)

[NVL 25](#_Toc101611045)

[CEIL 25](#_Toc101611046)

[FLOOR 25](#_Toc101611047)

[LPAD 25](#_Toc101611048)

[RPAD 26](#_Toc101611049)

[LTRIM 26](#_Toc101611050)

[RTRIM 26](#_Toc101611051)

[TRIM 26](#_Toc101611052)

[SUBSTR 27](#_Toc101611053)

[LENGTH 27](#_Toc101611054)

[ASCII 27](#_Toc101611055)

[CHR 27](#_Toc101611056)

[ROWNUM 28](#_Toc101611057)

[Ejercicios 29](#_Toc101611058)

[Ejercicios adicionales 31](#_Toc101611059)

[Bibliografía utilizada y recomendada 34](#_Toc101611060)

# Introducción a lenguaje SQL

# ¿Que es SQL?

SQL es un lenguaje de base de datos normalizado, eficiente y flexible, utilizado por los diferentes motores de bases de datos para la manipulación y examen de datos relacionales sobre los datos o sobre la estructura de los mismos.

Cada motor de base de datos tiene sus peculiaridades y lo hace diferente de otro motor por lo que el lenguaje SQL normalizado (ANSI) no nos serviría para resolver todos los problemas que se nos pueda presentar, lo que sí se puede garantizar es que cualquier sentencia escrita en ANSI es soportada o interpretada por cualquier motor de bases de datos.

Oracle PL/SQL es una extensión de SQL (ANSI) que amplia la funcionalidad del SQL agregando estructuras, como variables, tipos definidos por el usuario, estructuras de control como bucles, procedimientos, funciones, etc.

Servidor

Servidor

SQL

SQL

SQL SQL SQL

Cliente

Cliente

En la primer figura se dan múltiples comunicaciones con la red, una por cada orden SQL, en cambio en la segunda figura se empaquetan todas las ordenes SQL en un único paquete que se envía al servidor como una unidad, con lo cual se disminuye él trafico de red y se incrementa la velocidad de la aplicación.

# Componentes

El lenguaje SQL está compuesto por comandos, cláusulas, operadores y funciones de agregado. Estos elementos se combinan en las instrucciones para crear, actualizar y manipular las bases de datos.

### Comandos

Existen dos tipos de comandos SQL:

* Los DDL que permiten crear y definir nuevas bases de datos, campos e índices.
* Los DML que permiten generar consultas para ordenar, filtrar y extraer datos de la base de datos.

### Comandos DDL

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| CREATE | Para crear nuevas tablas, campos e índices |
| DROP | Para eliminar tablas e índices |
| ALTER | Para agregar columnas a las tablas, modificar tipos de datos |

### Comandos DML

|  |  |
| --- | --- |
| **Comando** | **Descripción** |
| SELECT | Utilizado para consultar registros de la base de datos que satisfagan un criterio determinado |
| INSERT | Utilizado para cargar lotes de datos en la base de datos en una única operación. |
| UPDATE | Utilizado para modificar los valores de los campos y registros especificados |
| DELETE | Utilizado para eliminar registros de una tabla de una base de datos |

# DDL - Lenguaje de declaración de datos.

### El comando CREATE TABLE.

Para crear una tabla en la base de datos se realiza con el comando CREATE TABLE, la sintaxis es la siguiente:

Sintaxis:

CREATE TABLE [ nombre\_de\_tabla ] (

columna\_1 tipo\_de\_datos, columna\_2 tipo\_de\_datos,

....

....

columna\_n tipo\_de\_datos

)

Ejemplo:

CREATE TABLE empleados (

emp\_codigo number(4), emp\_apellido varchar2(50), emp\_nombre varchar2(50), emp\_domicilio varchar2(100), emp\_cp number(4)

);

CREATE TABLE localidades (

loc\_codigo number(4), loc\_localidad varchar2(50), loc\_provincia varchar2(50)

);

### El comando DROP TABLE.

Para eliminar una tabla en la base de datos se realiza con el comando DROP TABLE. Sintaxis:

DROP TABLE [ nombre\_de\_tabla ] Ejemplo:

DROP TABLE empleados;

DROP TABLE localidades;

### El comando ALTER.

Para alterar la estructura de una tabla, ya sea agregar columnas, modificar el tipo de datos de una columna, agregar una clave primaria o una clave foránea utilizamos el comando ALTER, la sintaxis es la siguiente:

Sintaxis:

ALTER TABLE [ nombre\_de\_tabla ]

ADD [ nombre\_de\_columna ] [ tipo\_de\_datos ];

ALTER TABLE [ nombre\_de\_tabla ]

MODIFY [ nombre\_de\_columna ] [ tipo\_de\_datos ];

ALTER TABLE [ nombre\_de\_tabla ]

CHANGES [ nombre\_de\_columna ] [ nombre\_de\_columna ];

ALTER TABLE [ nombre\_de\_tabla ] DROP COLUMN [ nombre\_de\_columna ];

ALTER TABLE [ nombre\_de\_tabla ]

ADD CONSTRAINT [ nombre\_constraint ] PRIMARY KEY ( [ nombre\_columna ] );

Ejemplos:

ALTER TABLE empleados ADD barrio varchar2(2);

ALTER TABLE empleados MODIFY barrio varchar2(200);

ALTER TABLE empleados

CHANGES barrio barrios;

ALTER TABLE empleados DROP COLUMN barrio;

ALTER TABLE empleados

ADD CONSTRAINT empleados\_pk PRIMARY KEY (emp\_codigo);

### Comentarios

Para hacer comentarios existen dos formas: Comentario *monolínea*

-- comentario

comentario *multilínea*

/\* comentario comentario comentario \*/

### Scripts para crear tablas

CREATE TABLE empleado (

emp\_codigo number(4) CONSTRAINT empleados\_pk PRIMARY KEY, emp\_apellido varchar2(50),

emp\_nombre varchar2(50), emp\_domicilio varchar2(100)

);

CREATE TABLE localidad (

loc\_codigo number(4), loc\_localidad varchar2(50), loc\_provincia varchar2(50), loc\_cp number(4)

);

Agregamos la columna loc\_codigo en la tabla empleado ALTER TABLE empleado

ADD loc\_codigo number(4);

Definimos loc\_codigo como clave primaria en la tabla localidad ALTER TABLE localidad

ADD CONSTRAINT localidad\_pk PRIMARY KEY (loc\_codigo);

Definimos loc\_codigo como clave foránea en la tabla empleado, referenciando dicho código a la tabla localidad.

ALTER TABLE empleado

ADD CONSTRAINT empleado\_fk FOREIGN KEY (loc\_codigo) REFERENCES localidad (loc\_codigo) ;

### Scripts para crear índices

Indice por código de empleado de la tabla empleado (si ya es clave primaria da un error) CREATE INDEX empleado\_1\_idx on empleado (emp\_codigo);

Indice por código postal de la tabla localidad

CREATE INDEX localidad\_1\_idx on localidad (loc\_cp);

Indice por código de la tabla localidad (si ya es clave primaria da un error) CREATE INDEX localidad\_2\_idx on localidad (loc\_codigo);

# DML – Lenguaje de manipulación de datos

### Cláusulas

Las cláusulas son condiciones de modificación utilizadas para definir los datos que desean seleccionar o manipular.

|  |  |
| --- | --- |
| **Cláusula** | **Descripción** |
| FROM | Utilizada para especificar la tabla de la cual se van a seleccionar los registros. |
| WHERE | Utilizada para especificar las condiciones que deben reunir los registros que se van a seleccionar. |
| GROUP BY | Utilizada para separar los registros seleccionados en grupos específicos |
| HAVING | Utilizada para expresar la condición que debe satisfacer cada grupo |
| ORDER BY | Utilizada para ordenar los registros seleccionados de acuerdo con un orden específico |

### Operadores lógicos

|  |  |
| --- | --- |
| **Operador** | **Uso** |
| AND | Es él "y" lógico. Evalúa dos condiciones y devuelve un valor de verdad sólo si ambas son ciertas. |
| OR | Es él "o" lógico. Evalúa dos condiciones y devuelve un valor verdadero si alguna de las dos es cierta. |
| NOT | Negación lógica. Devuelve el valor contrario de la expresión. |

### Operadores de Comparación

|  |  |
| --- | --- |
| **Operador** | **Uso** |
| < | Menor que |
| > | Mayor que |
| <> | Distinto de |
| <= | Menor ó Igual que |
| >= | Mayor ó Igual que |
| = | Igual que |
| BETWEEN | Utilizado para especificar un intervalo de valores |
| LIKE | Utilizado en la comparación de un modelo |
| IN | Utilizado para especificar registros de una base de datos |

### Funciones de Agregado

Las funciones de agregado se usan dentro de una cláusula SELECT en grupos de registros para devolver un único valor que se aplica a un grupo de registros.

|  |  |
| --- | --- |
| **Función** | **Descripción** |
| AVG | Utilizada para calcular el promedio de los valores de un campo determinado |
| COUNT | Utilizada para devolver el número de registros de la selección |
| SUM | Utilizada para devolver la suma de todos los valores de un campo determinado |
| MAX | Utilizada para devolver el valor más alto de un campo especificado |
| MIN | Utilizada para devolver el valor más bajo de un campo especificado |

# Consultas Simples

### La sentencia SELECT y FROM

Sintaxis:

SELECT Campos FROM Tabla;

La sentencia SELECT nos permite definir las columnas de la tabla que queremos consultar, las columnas de la sentencia se deben delimitar por comas.

La sentencia FROM nos permite definir la tabla a la que queremos acceder.

Los campos que se quieren consultar se expresan en la sección “campos” de la sintaxis, separándose por comas los campos que se quieren consultar.

Por ejemplo:

SELECT cellular\_number, esn FROM cellulars;

Esta consulta devuelve dos columnas con n cantidad de filas, donde n es la cantidad de filas que posee la tabla.

### Alias

En determinadas circunstancias es necesario asignar un nombre a alguna columna determinada de un conjunto devuelto, otras veces por simple capricho o por otras circunstancias. Para resolver todas ellas tenemos la palabra reservada AS que se encarga de asignar el nombre que deseamos a la columna deseada. Tomando como referencia el ejemplo anterior podemos hacer que la columna devuelta por la consulta, en lugar de llamarse cellular\_number (igual que el campo devuelto) se llame “Nro. De Celular”. En este caso procederíamos de la siguiente forma:

SELECT cellular\_number AS “Nro. De Celular” FROM cellulars;

AS no es una palabra reservada del ANSI, existen diferentes sistema de asignar los alias en función del motor de bases de datos. En ORACLE para asignar un alias a un campo hay que hacerlo de las siguientes formas:

SELECT cellular\_number AS “Nro. De Celular” FROM cellulars; O también:

SELECT cellular\_number “Nro. De Celular” FROM cellulars;

También podemos asignar alias a las tablas, en este caso hay que tener en cuenta que en todas las referencias que deseemos hacer a dicha tabla se ha de utilizar el alias en lugar del nombre. Esta es útil cuando se utilizan varias tablas en una misma consulta.

Por ejemplo:

SELECT c.client\_id AS “Cliente” FROM clients c;

El alias de la tabla clients es c, este alias se debe utilizar en las columnas de selección debido a que una columna se puede repetir en varias tablas de la consulta.

### La cláusula WHERE

La cláusula WHERE nos permite condicionar las filas o registros de datos en los resultados de la consulta, para esto los registros deben cumplir las condiciones que se especifican en dicha cláusula.

Esta cláusula es opcional y debe ir a continuación del FROM, si no se emplea esta cláusula, la consulta devolverá todas las filas de la tabla.

Sintaxis:

WHERE expresion1 operador expresion2

Expresion1 y expresion2 pueden ser valores constantes o columnas de una tabla. Operador es un operador relacional que une dos expresiones.

Ejemplo:

La siguiente consulta nos devuelve los datos de los clientes con client\_id mayor a 30000. SELECT \* FROM clients WHERE client\_id > 30000;

La siguiente consulta nos devuelve los datos de los clientes con client\_id = 21000 SELECT \* FROM clients WHERE client\_id = 21000;

La siguiente consulta nos devuelve los datos de los clientes cuyo apellido comienza con la letra ‘A’. SELECT \* FROM clients WHERE surname like ‘A%’;

El caracter % indicar cualquier cadena después de la letra A.

La siguiente consulta nos devuelve los datos de los clientes cuyos apellidos comienzan con MA y luego cualquier carácter.

SELECT \* FROM clients WHERE surname like ‘MA%’;

La siguiente consulta nos devuelve los datos de los clientes con client\_id mayor e igual a 10000 and client\_id menor a igual 20000.

SELECT \* FROM clients WHERE client\_id BETWEEN 10000 and 20000; o

SELECT \* FROM clients WHERE client\_id >= 10000 AND client\_id <= 20000;

La siguiente consulta nos devuelve los datos de los clientes con client\_id igual a 21000 o client\_id igual a 21001 o client\_id igual a 21002.

SELECT \* FROM clients WHERE client\_id = 21000 or client\_id = 21001 or client\_id = 21002; o

SELECT \* FROM clients WHERE client\_id = 21000 UNION

SELECT \* FROM clients WHERE client\_id = 21001 UNION

SELECT \* FROM clients WHERE client\_id = 21002

### Comodines

|  |  |
| --- | --- |
| **Comodín** | **Descripción** |
| **\*** | Sustituye a todos los campos. |
| **%** | Sustituye a cualquier cosa o nada dentro de una cadena. |
| **\_** | Sustituye un solo carácter dentro de una cadena. |

### La cláusula ORDER BY

Cuando se recuperan registros de una tabla se puede especificar el orden en que se desean ver los datos, para esto utilizamos la cláusula ORDER BY.

Ejemplo:

SELECT client\_id AS “Cliente” FROM clients ORDER BY client\_id; o

SELECT client\_id AS “Cliente” FROM clients ORDER BY 1;

También podemos ordenar registros por más de una columna, como por ejemplo:

SELECT client\_id, name, surname AS “Cliente” FROM clients ORDER BY name, surname;

El orden por defecto es ascendente y se lo puede especificar mediante la cláusula ASC, para el ordenamiento descendente se lo especifica mediante la cláusula DESC.

SELECT client\_id, surname, name FROM clients ORDER BY surname DESC, name ASC;

**Las cláusulas DISTINCT/ALL DISTINCT**

La cláusula DISTINCT omite los registros que contienen datos duplicados en los campos seleccionados.

Para que los valores de cada campo listado en la instrucción SELECT se incluyan en la consulta deben ser únicos.

Por ejemplo, la tabla cellulars contiene la columna accounts\_id donde se repite para los distintos celulares, si necesitamos obtener únicamente las distintas cuentas lo obtenemos con la siguiente sentencia:

SELECT DISTINCT account\_id FROM cellulars;

Con otras palabras el predicado DISTINCT devuelve aquellos registros cuyos campos indicados en la cláusula SELECT posean un contenido diferente.

## ALL

Para seleccionar todas las columnas de la/s tablas/s a consultar podemos utilizar el comodín (\*) asterisco.

# Consultas Multitablas

### Unión de tablas

Cuando una consulta esta compuesta por más de una tabla se debe buscar un campo que contengan información en común en las tablas que interactúan en la consulta, este campo en común nos determina una relación entre las tablas. Estos campos son los que tenemos que relacionar en la cláusula WHERE, esto se lo conoce con el término de JOIN.

El JOIN es indicado en la cláusula WHERE como una condición, en el ejemplo que se muestra a continuación se van a mostrar todas las columnas de la tabla clients y todas las columnas de la tabla accounts, los registros que se van a mostrar son aquellos donde la el campo client\_id de la tabla clients sea igual al campo client\_id de la tabla accounts.

Sintaxis:

SELECT columnas FROM tabla1 a, tabla2 b WHERE a.columna1 = b.columna1

Ejemplo:

SELECT \* FROM clients a, accounts b WHERE a.client\_id = b.client\_id;

### Producto cartesiano

Cuando dos tablas en una consulta no se relacionan a través de un campo se obtiene lo que se llama, producto cartesiano, la cantidad de filas que obtenemos en la consulta es el producto de la cantidad de filas de cada una de las tablas.

Sintaxis:

SELECT \* FROM Tabla1, Tabla2 Ejemplo:

SELECT \* FROM Tabla1, Tabla2 Tabla1 = 500 registros.

Tabla2 = 10 registros.

Cantidad de filas resultantes = 5000 registros.

### El outer join

En el ejemplo anterior, la cantidad de filas que se obtienen en la consulta es la cantidad de filas que tiene la tabla accounts, esto es debido a que la relación entre las tablas en el modelo de datos es que un cliente tiene muchas cuentas. En el caso de que un cliente no tenga ninguna cuenta, dicho cliente no va a ser devuelto en la consulta, para esto utilizamos el outer join. El outer join se indica con el signo mas entre paréntesis (+), se debe especificar en la columna de la tabla donde no existen filas.

Sintaxis:

SELECT columnas FROM tabla1 a, tabla2 b WHERE a.columna1 = b.columna1(+)

Ejemplo:

SELECT \* FROM clients a, accounts b WHERE a.client\_id = b.client\_id(+);

Esta consulta nos devuelve todos los clientes de la tabla clients con las cuentas asociadas a cada uno de los clientes por mas que no existan cuentas para el cliente las columnas de la tabla accounts devolverán valores nulos.

### El operador UNION

El operador UNION combina el resultado de dos sentencias SELECT en un único resultado. Este resultado se compone de todos los registros devueltos de cada una de las sentencias, por defecto los registros repetidos se omiten en la consulta, para que no se omitan estos registros se debe utilizar la palabra ALL.

Sintaxis:

SELECT columnas FROM tabla1 UNION

SELECT columnas FROM tabla2

Ejemplo:

SELECT surname, name FROM clients WHERE surname like ‘MAR%’ UNION ALL

SELECT surname, name FROM clients WHERE surname like ‘BAR%’

Esta consulta nos devuelve todos los clientes con apellido que comienzan con ‘MAR’ y le unimos todos los clientes con apellido que comienzan con ‘bar’, si en la consulta hay filas repetidas y las queremos eliminar usamos el operador UNION en vez de UNION ALL.

Tener en cuenta que la cantidad de columnas de ambas consultas deben ser iguales.

### La cláusula GROUP BY

La cláusula GROUP BY se utiliza para hacer una consulta sumarizada, en vez de devolver una fila por cada fila de datos de la base de datos, una consulta sumarizada agrupa las columnas que especificamos en la consulta obteniendo como resultado 1 fila por cada uno de los grupos. En estas consultas podemos utilizar las funciones de agregación para obtener totales por cada uno de los grupos.

Las columnas de agrupamiento deben coincidir con las columnas de la sentencia SELECT sin incluir las funciones de agregación.

Sintaxis:

SELECT columnas, count(\*) FROM tabla1 GROUP BY columnas

Ejemplo:

La siguiente consulta nos devuelve la cantidad de clientes con el mismo nombre.

SELECT name, count(\*) FROM clients a GROUP BY name;

La siguiente consulta nos devuelve el importe vendido, la factura con menor importe y la factura de mayor importe de los clientes

SELECT b.name “Nombre Cliente”, sum(a.importe) “Cantidad Vendida”, min(a.importe) “Minima Factura”, max(a.importe) “Máxima Factura”

FROM ventas a, clients b WHERE a.client\_id = b.client\_id GROUP BY b.name;

Si a la consulta anterior le agregamos una columna para obtener las cantidades pero por año y mes deberíamos hacer:

SELECT b.name “Nombre Cliente”, to\_char(fecha\_factura,’yyyymm’) “Año Mes”, sum(a.importe) “Cantidad Vendida”, min(a.importe) “Minima Factura”, max(a.importe) “Máxima Factura”

FROM ventas a, clients b WHERE a.client\_id = b.client\_id

GROUP BY b.name, to\_char(fecha\_factura,’yyyymm’);

### La cláusula HAVING

La cláusula HAVING nos permite incluir solo ciertos grupos producidos por la cláusula GROUP BY, en otras palabras es aplicar una condición a un agrupamiento.

Sintaxis:

SELECT columnas, count(\*) FROM tabla1 GROUP BY columnas

HAVING expresion1 operador expresion2

Ejemplo:

La siguiente consulta nos devuelve la cantidad de clientes con el mismo nombre cuya cantidad de nombres es mayor e igual a 10.

SELECT name, count(\*) FROM clients a GROUP BY name

HAVING count(\*) > 9;

La siguiente consulta nos devuelve el importe vendido de todos los clientes que facturaron más de 10000. SELECT b.name “Nombre Cliente”,

sum(a.importe) “Cantidad Vendida” FROM ventas a, clients b

WHERE a.client\_id = b.client\_id

GROUP BY b.name

HAVING sum(a.importe) > 10000;

# Subconsultas

Nos permite encontrar un criterio de selección para otra consulta Cuando el criterio de búsqueda no es conocido usamos una Subconsulta

Pueden ser muy útiles cuando se necesita seleccionar filas de una tabla con una condición que depende de los datos que están en la misma tabla o en alguna otra.

Se pueden colocar en diferentes cláusulas, como por ejemplo: SELECT (siempre y cuando devuelvan un valor).

##### FROM WHERE

### Subconsultas en Cláusula WHERE

Sintaxis:

SELECT lista\_select FROM tabla

WHERE expr operador

(SELECT lista\_select FROM tabla)

Ejemplo: empleados que ganan más que el empleado 110.

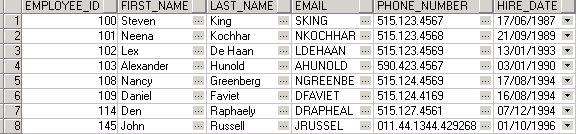
#### SELECT \*

**FROM** employees

**WHERE** salary > (**SELECT** salary

**FROM** employees s

**WHERE** s.employee\_id = 110)



Una subconsulta debe estar encerrada entre paréntesis.

La subconsulta debe aparecer a la derecha del operador de comparación. Se puede utilizar funciones de grupo en una subconsulta.

Ejemplo: Mostrar los empleados cuyo salario es igual al mínimo. SELECT \*

FROM employees

WHERE salary = (SELECT MIN (salary)

FROM employees) Algunos errores comunes:

##### SELECT \*

FROM employees

WHERE salary = (SELECT MIN (salary)

FROM employees GROUP BY department\_id)

### Referencias Externas

La subconsulta hace referencia a una columna de la tabla principal. Se evalúa una vez para cada fila que procesa la sentencia principal.

Mostrar todos los empleados que ganan más que el salario medio de su departamento.

SELECT last\_name, first\_name, department\_id FROM employees e

WHERE salary > (SELECT AVG (salary)

FROM employees ea

WHERE ea.department\_id = e.department\_id)

### Operador EXISTS

Comprueba la existencia de filas en el juego de resultados de la subconsulta. Es una alternativa del Operador IN.

Ejemplo: Buscar todos los departamentos que tengan empleados SELECT \*

FROM departments d

WHERE EXISTS (SELECT ‘a’

FROM employees e

WHERE e.department\_id = d.department\_id)

### Subconsultas en Cláusula FROM

Define un origen de datos sólo para esa sentencia. Ventaja: No creo un objeto en la Base de Datos.

La subconsulta en la cláusula FROM debe estar encerrada entre paréntesis y además es posible asignarle un alias. Las columnas seleccionadas en la subconsulta pueden ser referenciadas en la consulta principal, tal como se referencian las columnas de una tabla normal.

SELECT a.last\_name, a.salary

FROM employees a, (SELECT department\_id,

AVG (salary) sala FROM employees

GROUP BY department\_id) b WHERE a.department\_id = b.department\_id

AND a.salary >b.sala

Empleados que ganan más que el salario medio de su departamento.

### Subconsultas en Cláusula SELECT

La Subconsulta debe devolver un único valor.

Mostrar el apellido, número de departamento y el salario más alto del departamento de los empleados que el apellido comience con una H.

SELECT last\_name, department\_id,

(SELECT MAX (salary) FROM employees sq

WHERE sq.department\_id = e.department\_id ) NSAL FROM employees

WHERE last\_name LIKE ‘H%’;

# DML – Lenguaje de manipulación de datos.

## SELECT

Selecciona todos las columnas de la tabla EMPLEADO. SELECT \*

##### FROM EMPLEADO

Selecciona todas las columnas de la tabla EMPLEADO donde el código del empleado es igual a 1. SELECT \*

##### FROM EMPLEADO WHERE EMP\_CODIGO = 1

Selecciona todas las columnas de la tabla EMPLEADO donde el código del empleado es igual a 1 o igual a 2.

##### SELECT \*

##### FROM EMPLEADO

##### WHERE EMP\_CODIGO IN (1,2)

Selecciona todas las columnas de la tabla EMPLEADO donde el código del empleado es igual a 1 o igual a 2.

##### SELECT \*

##### FROM EMPLEADO

##### WHERE EMP\_CODIGO = 1 OR EMP\_CODIGO = 2

Selecciona todas las columnas de la tabla EMPLEADO y la tabla LOCALIDAD. SELECT \*

##### FROM EMPLEADO EMP, LOCALIDAD LOC WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO

Selecciona la provincia y la cantidad de clientes por provincia de las tablas LOCALIDAD y PROVINCIA. SELECT LOC\_PROVINCIA, COUNT(\*) CANTIDAD

##### FROM EMPLEADO EMP, LOCALIDAD LOC WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO GROUP BY LOC\_PROVINCIA

Selecciona todas las provincias y la cantidad de clientes por provincia de las tablas LOCALIDAD y PROVINCIA.

##### SELECT LOC\_PROVINCIA, COUNT(\*) CANTIDAD FROM EMPLEADO EMP, LOCALIDAD LOC

##### WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO(+) GROUP BY LOC\_PROVINCIA

Selecciona todas las provincias y la cantidad de clientes por provincia de las tablas LOCALIDAD y PROVINCIA, para las provincias donde la localidad es nula muestra SIN DATOS el nombre de la provincia.

##### SELECT NVL(LOC\_PROVINCIA,’SIN DATOS’) PROVINCIA , COUNT(\*) CANTIDAD FROM EMPLEADO EMP, LOCALIDAD LOC

##### WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO(+) GROUP BY LOC\_PROVINCIA

Selecciona las provincias y la cantidad de clientes por provincia de las tablas LOCALIDAD y PROVINCIA, para aquellas provincias donde la cantidad de clientes es mayor a 10, para las provincias donde la localidad es nula muestra SIN DATOS el nombre de la provincia.

##### SELECT NVL(LOC\_PROVINCIA,’SIN DATOS’) PROVINCIA , COUNT(\*) CANTIDAD FROM EMPLEADO EMP, LOCALIDAD LOC

##### WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO GROUP BY LOC\_PROVINCIA

##### HAVING COUNT(\*) > 10

Selecciona todas las columnas de la tabla EMPLEADO donde existe al menos una fila en la tabla LOCALIDAD.

##### SELECT \*

##### FROM EMPLEADO EMP WHERE EXISTS (SELECT \*

##### FROM LOCALIDAD LOC

##### WHERE LOC.LOC\_CODIGO = EMP.LOC\_CODIGO)

## INSERT

Inserta en la tabla empleado el código, apellido y nombre.

##### INSERT INTO EMPLEADO(EMP\_CODIGO, EMP\_APELLIDO, EMP\_NOMBRE) VALUES(1, ‘PEREZ’, ‘JUAN’)

Inserta en la tabla EMPLEADO el código, apellido, nombre y código de localidad.

##### INSERT INTO EMPLEADO(EMP\_CODIGO, EMP\_APELLIDO, EMP\_NOMBRE, LOC\_CODIGO) VALUES(1, ‘PEREZ’, ‘JUAN’, 1)

*ERROR en línea 1:*

*ORA-00001: restricción única (EMPLEADOS\_PK) violada*

Como existe el código 1 nos da el error de clave primaria.

Inserta en la tabla LOCALIDAD el código, localidad y provincia.

##### INSERT INTO LOCALIDAD(LOC\_CODIGO, LOC\_LOCALIDAD, LOC\_PROVINCIA) VALUES(1, ‘CORDOBA’, ‘CORDOBA’)

Si insertamos una nueva localidad con el código 1 nos va a dar error de clave primaria al igual que en el ejemplo anterior.

##### INSERT INTO LOCALIDAD(LOC\_CODIGO, LOC\_LOCALIDAD, LOC\_PROVINCIA) VALUES(1, ‘VILLA MARIA’, ‘CORDOBA’)

*ERROR en línea 1:*

*ORA-00001: restricción única (LOCALIDAD\_PK) violada*

Para esto agregamos la misma localidad con el código 2.

##### INSERT INTO LOCALIDAD(LOC\_CODIGO, LOC\_LOCALIDAD, LOC\_PROVINCIA) VALUES(2, ‘VILLA MARIA’, ‘CORDOBA’)

Insertamos en la tabla EMPLEADO el empleado Eduardo Perez cuyo domicilio esta en la localidad CORDOBA.

##### INSERT INTO EMPLEADO(EMP\_CODIGO, EMP\_APELLIDO, EMP\_NOMBRE, LOC\_CODIGO) VALUES(2, ‘PEREZ’, ‘EDUARDO’, 1)

El código de localidad 1 pertenece a la localidad CORDOBA.

Insertamos en la tabla EMPLEADO 10000 registros de la tabla CLIENTS donde la columna SURNAME es no nula y como código del empleado asignamos el valor determinado por la secuencia SEQ\_EMPLEADO.

##### INSERT INTO EMPLEADO(EMP\_CODIGO, EMP\_APELLIDO, EMP\_NOMBRE)

SELECT seq\_empleado.NEXTVAL, SURNAME, NAME FROM CLIENTS

##### WHERE SURNAME IS NOT NULL AND ROWNUM < 10001

## UPDATE

Actualizamos en la tabla EMPLEADO el nombre Eduardo por el nombre Esteban para el empleado con código igual a 3.

UPDATE empleado

SET emp\_nombre = ‘Esteban’ WHERE emp\_codigo = 3

Actualizamos en la tabla EMPLEADO el nombre por el mismo nombre asignando la primera letra de cada nombre en mayúscula para aquellos empleados con código de localidad igual a 5.

UPDATE empleado

SET emp\_nombre = initcap(emp\_nombre) WHERE loc\_codigo = 5

Actualizamos en la tabla EMPLEADO el nombre y apellido de 1 empleado, si el empleado no existe insertarlo en la misma tabla.

##### BEGIIN

UPDATE empleado

SET emp\_nombre = ‘Hernan’, emp\_apellido = ‘Martinez’ WHERE emp\_codigo = 983;

##### IF SQL%ROWCOUNT = 0 THEN

INSERT INTO empleado (emp\_codigo, emp\_apellido, emp\_nombre) VALUES(983, ‘Martinez’, ‘Hernan’);

##### END;

##### END IF; COMMIT;

## MERGE

En el caso que tenemos dos tablas y necesitamos actualizar datos en una e insertar filas si es que no existen podemos utilizar el comando MERGE.

Por ejemplo:

MERGE INTO usuarios a USING usuarios\_stl b

ON (a.usu\_codigo = b.usu\_codigo) WHEN MATCHED THEN

##### UPDATE SET

a.usu\_nombre = b.usu\_nombre, a.usu\_apellido = b.usu\_apellido

##### WHEN NOT MATCHED THEN

INSERT (usu\_codigo, usu\_nombre, usu\_apellido) VALUES(b.usu.codigo, b.usu\_nombre, b.usu\_apellido);

## DELETE

Borramos todos los registros de la tabla EMPLEADO. DELETE FROM EMPLEADO

Si queremos eliminar todos los registros de la tabla podemos utilizar el comando TRUNCATE, la diferencia con DELETE es que con el delete tenemos que confirmar el borrado con COMMIT y nos queda la posibilidad de volver los cambios con ROLLBACK. El comando TRUNCATE es siguiente:

TRUNCATE TABLE empleado

Borramos todos los registros de la tabla EMPLEADO donde el código de empleado es igual a 1 DELETE FROM empleado

WHERE emp\_codigo = 1

Borramos todos los registros de la tabla EMPLEADO donde el código de empleado es igual a 1 DELETE FROM empleado

WHERE emp\_codigo = 1

Borramos todos los registros de la tabla EMPLEADO de las localidades cuyo nombre comienzan con LAB.

DELETE FROM empleado

WHERE loc\_codigo in (SELECT loc\_codigo from LOCALIDAD WHERE loc\_localidad LIKE ‘LAB%’)

# Funciones SQL predefinidas

SQL proporciona varias funciones predefinidas a las que se puede llamar desde una orden SQL o desde órdenes procedimentales de PL/SQL.

Algunas de las funciones más interesantes son:

## TO\_DATE

La función TO\_DATE convierte un valor de tipo carácter o numérico a tipo date. Ejemplo:

##### SELECT TO\_DATE(’18-05-2004’,DD-MM-YYYY’) FECHA FROM DUAL

##### FECHA

18-05-2004

Convertimos una cadena de caracteres a fecha. Algunos formatos específicos

YYY, YY, Y Last three, two or one digits of the year

Y,YYY (Four Y's with comma) put a comma in the year (1,995)

RR Last two digits of year in another century (allows for year 2000) B.C., A.D. BC or AD indicators with periods

Q Numeric quarter of the year (1-4 with Jan-Mar=1)

MM 2 number month (Jan = 01)

RM Roman numeral month

MONTH Name of month spelled out (upper case - month is upper case) MON abbreviated name of month (upper case - month is upper case) WW Week of the year (1-53)

IW Week of the year (1-52, 1-53) based on ISO standard W Week of month (1-5)

DDD day of year (1-366) (Don't forget leap years) DD day of month (1-31)

D day of week (1-7)

DAY Name of day (upper case, day is upper case) DY Abbreviated name of day

J Julian day (Number of days since January 1, 4712 BC) A.M., P.M. Meridian indicator with periods.

HH, HH12 Hour of day (0-12)

HH24 Use 24 hour clock for hours (1-24)

MI Minute (0-59)

SS Second (0-60)

SSSSS (five S's) seconds past midnight. (0-86399)

## TO\_CHAR

La función TO\_CHAR convierte un valor de tipo numérico o fecha a tipo caracter. Ejemplo:

##### SELECT TO\_CHAR(SYSDATE,’DDMMYYYY’) FECHA FROM DUAL FECHA

18052004

Convertimos la fecha actual del sistema a caracter.

Algunos formatos específicos date

CC, SCC Century (S prefixes BC dates with a minus sign) YYYY, SYYYY Year (S prefixes BC dates with a minus sign) IYYY Year based on ISO standard

YYY, YY, Y Last three, two or one digits of the year IYY, IY, I Last three, two or one digits of the ISO year

Y,YYY (Four Y's with comma) put a comma in the year (1,995) YEAR, SYEAR Year spelled out (S prefixes BC dates with a minus sign)

RR Last two digits of year in another century (allows for year 2000) Q Numeric quarter of the year (1-4 with Jan-Mar=1)

MM 2 number month (Jan = 01)

RM Roman numeral month

MONTH Name of month spelled out (upper case - month is upper case) MON abbreviated name of month (upper case - month is upper case) WW Week of the year (1-53)

IW Week of the year (1-52, 1-53) based on ISO standard W Week of month (1-5)

DDD day of year (1-366) (Don't forget leap years) DD day of month (1-31)

D day of week (1-7)

DAY Name of day (upper case, day is upper case) DY Abbreviated name of day

J Julian day (Number of days since January 1, 4712 BC)

AM,PM Meridian indicator

A.M., P.M. Meridian indicator with periods. HH, HH12 Hour of day (0-12)

HH24 Use 24 hour clock for hours (1-24)

MI Minute (0-59)

SS Second (0-60)

SSSSS (five S's) seconds past midnight. (0-86399) None Date must be in the format 'DD-MON-YY';

Algunos formatos específicos numéricos

|  |  |  |
| --- | --- | --- |
| 9 | 9999 | significant digit |
| 0 | 0999 | leading zero instead of a blank |
| $ | $999 | leading dollar sign |
| B | B999 | leading blank instead of a zero |
| MI | 999MI | trailing minus sign |
| S | S999 | leading plus or minus sign |
| PR | 999PR | angle brackets for negative values |
| D | 99D99 | decimal character |
| G | 9G99 | group separator |
| C | C999 | ISO currency symbol |
| L | L999 | local currency symbol |
| , | 9,999 | comma |

. 99.99 period

V 999V99 10n multiplier; n is the number of 9's after the V RN, rn RN Upper or lower case roman numeral

## LAST\_DAY

La función LAST\_DAY toma como argumento una fecha y retorna una fecha con el último día del mes. Ejemplo:

##### SELECT LAST\_DAY(SYSDATE) ULTIMO FROM DUAL

##### ULTIMO

##### JUL-04

## ADD\_MONTHS

El procedimiento ADD\_MONTHS retorna la fecha pasada por parámetro con mas o menos meses especificados en el parámetro 2.

Ejemplo:

##### SELECT ADD\_MONTHS( TO\_DATE(’01-01-2004’,’DD-MM-YYYY’), 1) FECHA FROM DUAL

##### FECHA

01-02-2004

## INITCAP

La función INITCAP toma como argumento una cadena de caracteres y devuelve la misma cadena convirtiendo la primera letra de cada palabra a mayúsculas y las demás letras las convierte a minúsculas.

Ejemplo:

SELECT INITCAP('perez juan ') NOMBRE FROM DUAL

##### NOMBRE

Perez Juan

## LOWER

La función LOWER toma como argumento una cadena de caracteres y devuelve la cadena en minúsculas.

Ejemplo:

##### SELECT LOWER('PEREZ JUAN') FROM DUAL

##### LOWER('PEREZ JUAN')

perez juan

## UPPER

La función UPPER toma como argumento una cadena de caracteres y devuelve la cadena en mayúsculas. Ejemplo:

SELECT UPPER('perez juan') FROM DUAL

UPPER('perez juan')

##### PEREZ JUAN

## DECODE

DECODE(expresión base, compara1,valor1,

compara2,valor2,

...

predeterminado);

Funciona similar a una serie de órdenes IF-THEN-ELSE anidadas.

La expresión base se compara sucesivamente con las comparativas y cuando encuentra una coincidencia, las comparaciones restantes no se realizan. Si no encuentra nada devuelve el valor predeterminado.

Es importante aclarar que solo puede ser usada en órdenes SQL. SELECT DECODE(null,'a',1,null,2)

devuelve 2

## CASE

##### SELECT CASE WHEN TO\_NUMBER(TO\_CHAR(SYSDATE,’MM’)) <= 6 THEN ‘PRIMER SEMESTRE’

##### ELSE ‘SEGUNDO SEMESTRE’ END SEMESTRE

##### FROM DUAL

Funciona similar a una serie de ordenes IF-THEN-ELSE anidadas o al DECODE, la diferencia con decode es que podemos utilizar cualquier operador lógico, por ejemplo mayor, menor, like, etc.

## NVL

NVL(exp1,exp2);

Esta función resulta útil para asegurarse que los valores de una consulta no posean valores NULL. Puede ser usada en ordenes SQL y procedimentales.

SELECT NVL('NO NULO',7)||' - '||NVL(null,'NULAZO') FROM dual;

Devuelve: NO NULO – NULAZO

## CEIL

La función CEIL toma como argumento un valor con decimales y redondea hacia el valor entero inmediato superior.

Ejemplo:

##### SELECT CEIL(119.2) VALOR FROM DUAL VALOR

120

## FLOOR

La función FLOOR toma como argumento un valor con decimales y redondea hacia el valor entero inmediato inferior.

Ejemplo:

##### SELECT FLOOR(119.2) VALOR FROM DUAL VALOR

119

## LPAD

La función LPAD nos permite agregar caracteres a la izquierda de una cadena, pudiendo especificar la longitud final de la cadena y el carácter de relleno, tiene tres argumentos, el primero es la cadena de caracteres a rellenar, el segundo es la longitud con la que debe quedar la cadena y la tercera es el carácter de relleno, toma como argumento una fecha y retorna una fecha con el último día del mes.

Ejemplo:

##### SELECT LPAD(‘987’,8,’0’) VALOR FROM DUAL VALOR

00000987

## RPAD

La función RPAD nos permite agregar caracteres a la derecha de una cadena, pudiendo especificar la longitud final de la cadena y el carácter de relleno, tiene tres argumentos, el primero es la cadena de caracteres a rellenar, el segundo es la longitud con la que debe quedar la cadena y la tercera es el carácter de relleno, toma como argumento una fecha y retorna una fecha con el último día del mes.

Ejemplo:

##### SELECT RPAD(‘987’,8,’0’) VALOR FROM DUAL VALOR

98700000

## LTRIM

La función LTRIM tiene como argumento una cadena de caracteres y nos devuelve la misma cadena quitando los espacios en blanco que tiene a la izquierda de la misma.

Ejemplo:

##### SELECT LTRIM(‘ PEREZ’) VALOR FROM DUAL VALOR

##### PEREZ

## RTRIM

La función RTRIM tiene como argumento una cadena de caracteres y nos devuelve la misma cadena quitando los espacios en blanco que tiene a la derecha de la misma.

Ejemplo:

##### SELECT RTRIM(‘PEREZ ’) VALOR FROM DUAL VALOR

##### PEREZ

## TRIM

La función LTRIM tiene como argumento una cadena de caracteres y nos devuelve la misma cadena quitando los espacios en blanco que tiene tanto a la izquierda como a la derecha de la misma.

Ejemplo:

##### SELECT TRIM(‘ PEREZ ’) VALOR FROM DUAL VALOR

##### PEREZ

## SUBSTR

La función SUBSTR sirve para tomar una porción de una cadena de caracteres, toma como argumento una cadena de caracteres, la posición inicial desde donde queremos obtener la cadena y la cantidad de caracteres que deseamos obtener,

Ejemplo:

##### SELECT SUBSTR('PEREZ JUAN',7,5) NOMBRE FROM DUAL NOMBRE

##### JUAN

## LENGTH

La función LENGTH toma como argumento una cadena de caracteres y devuelve un número que indica la cantidad de caracteres de la cadena.

Ejemplo:

SELECT LENGTH('perez juan') FROM DUAL LENGTH('PEREZJUAN')

9

## ASCII

La función ASCII toma como argumento un carácter y nos devuelve el valor ASCII, esta función es la inversa a la función CHR.

Ejemplo:

##### SELECT ASCII('Q') CHR FROM DUAL CHR

81

## CHR

La función CHR toma como argumento un valor numérico y nos retorna el carácter ASCII, esta función es la inversa a la función CHR.

Ejemplo:

##### SELECT CHR(81) ASCII FROM DUAL ASCII

##### Q

## ROWNUM

Devuelve un valor de tipo NUMBER y se usa principalmente para limitar el número de filas. SELECT \* FROM empleado WHERE ROWNUM < 5;

Devuelve las primeras 4 filas de la tabla empleado

El valor de ROWNUM se asigna a las filas antes de que se realice cualquier tipo de ordenamiento. Como consecuencia, no se lo puede utilizar para recuperar las n primeras filas (de acuerdo al ROWNUM) según el orden especificado.

Por ejemplo

SELECT emp\_nombre FROM empleado WHERE ROWNUM < 5

ORDER BY emp\_nombre;

Esta consulta nos devuelve las 4 primeras filas pero no tiene en cuenta el orden especificado por el número de fila. Para realizar esta consulta habría que utilizar un CURSOR.

El ROWNUM se puede utilizar para crear una tabla con la misma estructura de otra que ya tengamos. Por ejemplo

CREATE TABLE nueva\_tabla AS SELECT \* FROM empleado WHERE ROWNUM < 1; Crea la tabla nueva\_tabla con la misma estructura de la empleado.

# Ejercicios

* 1. Mostrar todos los datos de los empleados (tabla: employees) ordenados por apellido (last\_name), cuyo nro de empleado (employee\_id) > 10.
  2. Mostrar el apellido (last\_name), nombre (first\_name) y salario (salario) del empleado (tabla: employees) con apellido (last\_name) llamado ‘Whalen’.
  3. Mostrar el nro de empleado (employee\_id), apellido (last\_name) y nombre (first\_name) de todos los empleados (tabla: employees) utilizando el alias *Apellido* para la columna last\_name y *nombre* para first\_name (deben mostrarse de la misma manera, respetando mayúsculas y minúsculas) ordenando los datos por nro de empleado (employee\_id) en forma descendente).
  4. Mostrar el apellido (last\_name) y nombre (first\_name) en una misma columna (asignar como alias *Apellido y Nombre*, además de la fecha de ingreso (hire\_date) de los empleados (tabla: employees) ordenando los datos por salario (salary), que se encuentran en el departamento (department\_id) 100 0 120.
  5. Mostrar los empleados cuyo apellido comience con A.
  6. Mostrar los apellidos (last\_name) de los empleados (tabla: employees), en el caso de que haya 2 empleados con el mismo apellido (last\_name) mostrar sólo una vez el mismo.
  7. Mostrar los datos del departamento (tabla: departments) denominado (department\_name) IT
  8. Mostrar el nro. De empleado (employee\_id), apellido (last\_name) y nombre de los empleados cuyo salario (salary) no sea igual a $2500.
  9. Mostrar los nombres (department\_name) de los departamentos (tabla: departments) que poseen un nro de departamento (department\_id) comprendido entre 40 y 100.
  10. Mostrar los empleados (tabla: employees) que no poseen departamento (department\_id).
  11. Mostrar el nro. De empleado (employee\_id), apellido (last\_name) y nombre de los empleados cuyo nro de empleado sea 100 o 101.
  12. Mostrar el nro de empleado (employee\_id), apellido (last\_name), nombre (first\_name), y el nombre de departamento (department\_name) de los empleados (tabla: employees) que se encuentran en le departamento (department\_id) 50
  13. Mostrar la cantidad de empleados (tabla: employees).
  14. Mostrar el apellido y nombre del empleado, además el nombre del departamento cuyos empleados tengan una o como segunda letra de su apellido.
  15. Mostrar el salario máximo (salary) de los empleados (tabla: employees) del departamento 20.
  16. Mostrar la diferencia entre el salario mínimo y máximo.
  17. Mostrar la cantidad de departamentos (department\_id) diferentes que poseen los empleados (tabla: employees).
  18. Mostrar la cantidad de empleados por departamento (employee\_id), si la cantidad es mayor a 5 (tabla: employees).
  19. Mostrar los empleados que ganan el salario mínimo.
  20. Mostrar el salario (salary) promedio de cada uno de los departamentos (asignarle el alias Salario Promedio.), además del nro de departamento (department\_id) y nombre (department\_name), (tablas: employees/departments)
  21. . Buscar todos los departamentos que no tengan empleados
  22. Mostrar el salario (salary) mínimo por departamento (department\_id) y cargo (job\_id). (tabla: employees).
  23. Mostrar los empleados (tabla: employees) que ganan más (salary) que el empleado 101.
  24. Mostrar los empleados (tabla: employees) que se encuentren en el departamento (department\_id) 50 y que ganan (salary) más que el empleado (employee\_id) 120
  25. Crear la tabla denominada emple con los mismos datos y estructura de la tabla employees.
  26. Crear la tabla dep con los siguientes campos:

Id, de tipo number(5) y nombre de tipo varchar2(50)

* 1. Crear la columna cantidad\_empleados en la tabla dep.La columna debe ser del tipo number.
  2. Modificar la tabla dep para que el nombre tenga una longitud de 100.
  3. Agregar una clave primaria a la tabla de emple (employee\_id)
  4. Insertar en la tabla dep el siguiente registro: Id 10

Nombre: nuevo departamento Cantidad\_empleados: 25

* 1. Modificar el nombre del departamento 10 de la tabla dep por *Recursos Humanos*
  2. Eliminar de la tabla emp el empleado 110.
  3. Eliminar la tabla dep.
  4. Mostrar el nro. De empleado, la fecha de ingreso (hire\_date) de los empleados (tabla:employees) y la misma fecha más un año.
  5. Mostrar el apellido (last\_name) en mayúsculas, nombre (first\_name) en minúsculas y nro de departamento (department\_id) de los empleados (tabla: employees) qua poseen de salario (salary) 7000. En el caso de que el empleado no posea departamento (department\_id) mostrar el nro 99 en lugar del valor nulo.
  6. Mostrar el último día del mes actual
  7. Mostrar el apellido (last\_name) de los empleados (tabla: employees) y la cantidad de letras de los mismos.
  8. Mostrar las 5 primeras filas de la tabla de empleados (tabla: employees).
  9. Mostrar el apellido, la cantidad de letras que posee y las 5 primeras letras de los apellidos.

# Ejercicios adicionales

#### Consultas básicas

1. Muestre el apellido y el salario para todos los empleados cuyos salarios no están comprendidos entre

$5.000 y $12.000.

1. Muestre el apellido del empleado, el identificador de cargo y la fecha de inicio de los empleados contratados entre el 20 de febrero de 1998 y el 1 de mayo de 1998. Ordene la consulta en orden ascendente por fecha de inicio.
2. Muestre el apellido y el cargo de todos los empleados que no tienen director.
3. Muestre el apellido de todos los empleados que tengan una a y una e en el apellido.

#### Join de tablas

1. Cree un listado único de todos los cargos que haya en el departamento 80. Incluya la ubicación del departamento en el resultado.
2. Escriba una consulta para mostrar el apellido del empleado, el nombre de departamento, el identificador de ubicación y la ciudad de todos los empleados que perciben comisión.
3. Escriba una consulta para mostrar el apellido, el cargo, el número de departamento y el nombre de departamento para todos los empleados que trabajan en Toronto.
4. Muestre el apellido y el número de empleado, junto con id y nombre de departamento. Etiquete las columnas Employee, Emp#, Cód Dep, Descrip Dep, respectivamente. Traiga todos los datos de empleados, más allá que no pertenezca a ningún departamento.

#### Funciones agregadas

1. Determine el número de directores sin enumerarlos. Etiquete la columna Number of Managers. Indicación: Utilice la columna MANAGER\_ID para determinar el número de directores.
2. Escriba una consulta que muestre la diferencia entre el salario más alto y el más bajo. Etiquete la columna DIFFERENCE.
3. Muestre el número de director y el salario del empleado peor pagado para ese director. Excluya los empleados de los que no se conozca el director. Excluya los grupos en los que el salario mínimo sea

$6.000 o menos. Ordene el resultado por salario en orden descendente.

#### Operadores de conjunto

1. Enumere los identificadores de departamento para departamentos que no contienen el ID de cargo ST\_CLERK, utilizando operadores de conjunto.
2. Escriba una consulta compuesta que enumere lo siguiente:
   * Id de empleados e identificadores de departamento de todos los empleados de la tabla EMPLOYEES, independientemente de si pertenecen o no a algún departamento.
   * Identificador y nombre de departamento de todos los departamentos de la tabla DEPARTMENTS, independientemente de si tienen o no empleados trabajando en ellos.

#### Subconsultas

1. Escriba una consulta para mostrar el apellido y la fecha de contratación de cualquier empleado del mismo departamento que Zlotkey. Excluya a Zlotkey.
2. Cree una consulta para mostrar los números de empleado y los apellidos de todos los empleados que ganen más del salario medio. Ordene los resultados por salario en orden ascendente.
3. Escriba una consulta para mostrar el apellido, el número de departamento y el salario de cualquier empleado cuyo número de departamento y salario coinciden con el número de departamento y salario de cualquier empleado que gane comisión.
4. Cree una consulta para mostrar los empleados que ganen un salario más alto que el de todos los directores de ventas (JOB\_ID = 'SA\_MAN'). Ordene los resultados por salario de más alto a más bajo.
5. Escriba una consulta para encontrar todos los empleados que ganen más que el salario medio en sus departamentos. Muestre el apellido, el salario, el identificador de departamento y el salario medio para el departamento. Ordene por salario medio. Utilice alias para las columnas recuperadas por la consulta.

#### Comandos DDL

1. Cree una tabla EMP basándose en el siguiente cuadro. Confirme que se ha creado la tabla.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Column Name** | ID | LAST\_NAME | FIRST\_NAME | DEPT\_ID |
| **Key Type** |  |  |  |  |
| **Nulls/Unique** |  |  |  |  |
| **FK Table** |  |  |  |  |
| **FK Column** |  |  |  |  |
| **Data type** | Number | Varchar2 | Varchar2 | Number |
| **Length** | 7 | 25 | 25 | 7 |

1. Modifique la tabla EMP para permitir apellidos de empleado más largos. Confirme la modificación.
2. Cree la tabla EMPLOYEES2 basándose en la estructura de la tabla EMPLOYEES. Incluya sólo las columnas EMPLOYEE\_ID, FIRST\_NAME, LAST\_NAME, SALARY y DEPARTMENT\_ID. Llame a las columnas de la nueva tabla ID, FIRST\_NAME, LAST\_NAME, SALARY y DEPT\_ID, respectivamente.
3. Borre la tabla EMP.
4. Cambie el nombre de la tabla EMPLOYEES2 a EMP.
5. Borre la columna FIRST\_NAME de la tabla EMP. Confirme la modificación comprobando la descripción de la tabla.
6. Agregue una restricción de PRIMARY KEY de nivel de tabla a la tabla EMP en la columna ID. Se debe poner nombre a la restricción en el momento de su creación. Llame a la restricción my\_emp\_id\_pk.
7. Modifique la tabla EMP. Agregue una columna COMMISSION del tipo de dato NUMBER, precisión 2, escala 2. Agregue una restricción a la columna COMMISSION que asegure que un valor de comisión sea mayor que cero.

#### Manipulación de datos

1. Crear la tabla MY\_EMPLOYEE que se utilizará para la práctica.

#### CREATE TABLE my\_employee

#### (id NUMBER(4) CONSTRAINT my\_employee\_id\_nn NOT NULL, last\_name VARCHAR2(25),

#### first\_name VARCHAR2(25), userid VARCHAR2(8), salary NUMBER(9,2));

1. Agregue la primera fila de datos a la tabla MY\_EMPLOYEE desde los siguientes datos de ejemplo. No enumere las columnas en la cláusula INSERT.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **LAST\_NAME** | **FIRST\_NAME** | **USERID** | **SALARY** |
| **1** | Patel | Ralph | Rpatel | 895 |
| **2** | Dancs | Betty | Bdancs | 860 |
| **3** | Biri | Ben | Bbiri | 1100 |
| **4** | Newman | Chad | Cnewman | 750 |
| **5** | Ropeburn | Audrey | Aropebur | 1550 |

1. Rellene la tabla MY\_EMPLOYEE con la segunda fila de los datos de ejemplo de la lista anterior. Esta vez, enumere las columnas explícitamente en la cláusula INSERT.
2. Haga que las adiciones de datos sean permanentes.
3. Cambie el apellido del empleado 3 por Drexler.
4. Cambie el salario a 1000 para todos los empleados con un salario menor que 900.
5. Suprima a Betty Dancs de la tabla MY\_EMPLOYEE.
6. Valide todos los cambios pendientes.

#### Funciones

1. Utilizando la función DECODE, escriba una consulta que muestre el grado de todos los empleados basándose en el valor de la columna JOB\_ID, según los siguientes datos:

*JOB GRADE*

##### AD\_PRES A

##### ST\_MAN B

##### IT\_PROG C

##### SA\_REP D

##### ST\_CLERK E

None of the above 0

1. Vuelva a escribir la sentencia de la pregunta anterior utilizando la sintaxis CASE.
2. Para cada empleado, muestre el número de empleado, el apellido, el salario y el salario con un aumento del 15 % y expresado como número entero. Etiquete la columna New Salary.
3. Cree una consulta que muestre el apellido y las comisiones de los empleados. Si un empleado no percibe comisión, ponga “No Commission”. Etiquete la columna COMM.
4. Muestre el salario más alto, el más bajo, el de suma y el medio para todos los empleados. Etiquete las columnas Maximum, Minimum, Sum y Average, respectivamente. Redondee los resultados hasta el número entero más cercano.
5. Modifique la consulta para mostrar los salarios mínimo, máximo, de suma y medio para cada tipo de cargo.

# Bibliografía utilizada y recomendada

Oracle 10g: Programación con PL/SQL Autor: LONEY KEVIN

Editorial: McGRAW-HILL