# Resumen SQL

Elementos:

* Comando: instrucciones que se pueden crear. Tres tipos:
  + CREATE
  + DROP
  + ALTER
* Manipulación de datos:
  + SELECT: consultar filas que cumplan un criterio determinado.
  + INSERT: cargar datos en una única operación.
  + UPDATE: modificar valores de campos y filas determinados.
  + DELETE: eliminar filas.
* De control y seguridad:
  + GRANT: conceder privilegios.
  + REVOKE: eliminar privilegios.
* Clausulas:
  + FROM: determinar la tabla de la que se van a selecciona las filas.
  + WHERE: especificar las condiciones (sueldo >2000).
  + ORDER BY: ordenar las filas seleccionadas de una manera específica (ASC|DESC).
  + GROUP BY: agrupar filas con mismos valores (Agrupar los empleados que viven en el mismo pueblo).
  + HAVING: establecer la condición que debe satisfacer cada grupo.

**\*SI HAY HAVING TIENE QUE HABER GROUP BY SÍ O SÍ\***

* Operadores:
  + AND: dos condiciones (o más), solo si se cumplen ambas.
  + OR
  + NOT: valor contrario a la expresión.
  + BETWEEN
  + LIKE: comparar con un valor entre comillas simples.
  + IN

\*\*\*\*% => para especificar que hay algo en la posición la que este colocada (J%O, hay texto entre la primera J (primera porque no hay % delante de ella) y la última O (última porque no hay % después de ella)

* Funciones:
  + AVG: promedio.
  + COUNT: nº filas sección.
  + SUM: suma de valores de una sección.

Crear bases de datos

Crear una base de datos consiste en crear las tablas que la componen.

**CREATE DATABASE nombreBD;**

Crear tablas

**CREATE TABLE**[esquema.]nombreTabla (

Columna1 tipoDato [**DEFAULT** valor][restricciones],

...

ColumnaN tipoDato [**DEFAULT** valor][restricciones]);

DEFAULT valor, será un valor por defecto.

Los elemnetos que esten entre corchetes son opcionales, no e sobligatorio que esten.

Ejemplo:

**CREATE TABLE Proveedores (nombre varchar2(50),**

**localidad DEFAULT ‘Palencia’);**

Tipos de datos

* Textos: (se indican los tamaños entre paréntesis tras el del tipo)
  + VARCHAR2: para textos de longitud variable, máximo 4000.
  + CHAR: para textos de longitud fija, hasta 2000.
  + NCHAR: para el almacenamiento de caracteres nacionales de longitud fija.
  + NVARCHAR2: para el almacenamiento de caracteres nacionales de longitud variable.
* Números:
  + NUMBER
  + NUMBER(p,s): número decimal dónde, p es la cantidad de números y s es el número de decimales a la derecha de la coma. NUMBER (8,3), permite representar hasta números de 8 cifras de las cuales 3 serían decimales.
  + NUMBER(p): número entero con p dígitos.
* Fechas y horas:
  + DATE(‘día/mes/año’)
  + TIMESTAMP(‘día/mes/año hora:minutos:segundos’)

Restricciones

* CONSTRAINT

**CREATE TABLE** USUARIOS(

Login varchar2(15) **CONSTRAINT** usu\_log\_pk PRIMARY KEY,

password varchar2(8) NOT NULL,

fechaIngreso DATE DEFAULT SYSDATE));

* NOT NULL: asociado a una columna y se obliga a tener un valor.

password varchar2(8) **NOT NULL**, ó

password varchar2(8) **CONSTRAINT** usu\_pas\_nn **NOT NULL**

* UNIQUE: asociada a una columna y se obliga a que no se puedan repetir valores.

**CREATE TABLE** USUARIOS (

login varchar2(15) **CONSTRAINT** usu\_log\_uk **UNIQUE**);

**CREATE TABLE** USUARIOS (

login varchar2(15) **UNIQUE**);

Si queremos que la restricción englobe a varios campos lo ponemos de la siguiente forma:

**CREATE TABLE** USUARIOS (

login varchar2(25),

correo varchar2(25),

**CONSTRAINT** usu\_uk **UNIQUE**(login, correo));

* PRIMARY KEY:

La clave primaria por defecto ya es NOT NULL y UNIQUE.

**CREATE TABLE** USUARIOS (

login varchar2(15) **CONSTRAINT** usu\_log\_pk **PRIMARY KEY**);

Ó

**CREATE TABLE** USUARIOS (

login varchar2(15) **PRIMARY KEY**);

Si la clave primaria está formada por más de un campo:

**CREATE TABLE** USUARIOS (

Nombre varchar2(25),

apellidos varchar2(25),

fecha\_nacimiento DATE,

**CONSTRAINT** usu\_uk **PRIMARY KEY**(nombre, apellidos, fecha\_nacimiento));

* REFERENCES. FOREIGN KEY:

Ejemplo: creamos una tabla que contiene información de alquileres de películas, el cliente que la alquila y la película alquilada, estos dos atributos son FK de otras dos tablas:

**CREATE TABLE** alquiler (

dni varchar2(9) **CONSTRAINT** alq\_dni\_fk **REFERENCES** clientes(dni),

cod\_pelicula NUMBER(5) **CONSTRAINT** alq\_pel\_fk **REFERENCES** peliculas(cod),

CONSTRAINT alq\_pel\_pk **PRIMARY** **KEY** (dni, cod\_pelicula);

Si el campo al que se hace referencia es la clave primaria, se puede omitir el nombre de este campo:

**CREATE TABLE** alquiler (

dni varchar2(9) **CONSTRAINT** alq\_dni\_fk **REFERENCES** clientes,

cod\_pelicula NUMBER(5) **CONSTRAINT** alq\_pel\_fk **REFERENCES** peliculas,

CONSTRAINT alq\_pel\_pk **PRIMARY** **KEY** (dni, cod\_pelicula);

Cuando las claves ajenas están formadas por más de un campo se pueden definir a nivel de tabla y en este caso será necesario utilizar la expresión FOREIGN KEY:

**CREATE TABLE** existencias (

Tipo char2(9),

Modelo number(3),

N\_almacen number(1),

Cantidad number(7),

**CONSTRAINT** exi\_tm\_fk **FOREIGN** **KEY** (tipo, modelo) **REFERENCES** piezas,

**CONSTRAINT** exi\_nal\_fk **FOREIGN** **KEY** (n\_almacen) **REFERENCES** almacenes,

**CONSTRAINT** exi\_pk **PRIMARY** KEY (tipo,modelo, n\_almacen)

);

* CHECK: permite comprobar que los valores introduces cumplen las condiciones asociadas a esa columna.

Supongamos la tabla USUARIOS, en su campo crédito sólo puede tomar valores entre 0 y 2000, se especificaría:

**CREATE TABLE** USUARIOS (

credito NUMBER(4) **CHECK** (credito BETWEEN 0 AND 2000));

Una misma columna puede tener varios CHECK asociados, para ello especificamos un CONSTRAINT para cada una de la siguiente forma:

**CREATE TABLE** ingresos (

cod NUMBER(5) **PRIMARY** **KEY**,

concepto VARCHAR2(50) **NOT** **NULL**,

importe NUMBER (11,2) **CONSTRAINT** importe\_min **CHECK** (importe > 0)

**CONSTRAINT** importe\_max **CHECK** (importe <8000));

En este ejemplo, se prohibe añadir datos cuyo importe no esté entre 0 y 8000.

Si queremos hacer referencia en una validación a otras columnas, hay que incluir la restricción a nivel de tabla, independiente de la columna:

**CREATE TABLE** ingresos (

cod NUMBER(5) **PRIMARY** **KEY**,

concepto VARCHAR2(50) **NOT** **NULL**,

importe\_max NUMBER (11,2),

importe NUMBER (11,2),

**CONSTRAINT** importe\_max **CHECK** (importe < importe\_max));

Eliminar tablas

**DROP TABLE nombreTabla [CASCADE CONSTRAINTS];**

También podemos eliminar las filas sin eliminar la estructura de la tabla:

**TRUNCATE TABLE nombreTabla;**

Modificar tablas

* Cambiar nombre:

**RENAME nombreViejo TO nombreNuevo;**

* Añadir columnas:

**ALTER TABLE nombreTabla ADD**

**(columnaNueva1 tipoDatos [propiedades],**

**[, columnaNueva2 tipoDatos [propiedades]] … );**

* Eliminar columnas:

**ALTER TABLE nombreTabla DROP COLUMN columna1 [, columna2, …];**

* Modificar columnas:

**ALTER TABLE nombreTabla MODIFY (columna1 tipoDato [propiedades] [, columna2 …]);**

* Renombrar columnas:

**ALTER TABLE nombreTabla RENAME COLUMN nombreAntiguo TO nombreNuevo;**

* Borrar restricciones:

**ALTER TABLE nombreTabla DROP CONSTRAINT nombreRestricción;**

* Modificar el nombre de una restricción:

**ALTER TABLE nombreTabla RENAME CONSTRAINT nombreViejo TO nombreNuevo;**

* Activar o desactivar restricciones:

**ALTER TABLE nombreTabla DISABLE CONSTRAINT nombreRestricción [CASCADE];**

La opción CASCADE desactiva las restricciones que dependan de ésta.

Para activar de nuevo la restricción:

**ALTER TABLE nombreTabla ENABLE CONSTRAINT nombreRestricción [CASCADE];**

Lenguaje de control de datos

* Crear usuarios:

**CREATE USER nombreUsuario**

**IDENTIFIED BY contraseña**

**QUOTA nº(K/M) on tablespace(users/system); //Limite de espacio que tiene el usuario para hacer tablas**

**QUOTA UNLIMITED ON tablespace(users/system); //Espacio ilimitado**

* Modificar usuarios:

**ALTER USER nombreUsuario**

**IDENTIFIED BY contraseña**

**[DEFAULT TABLESPACE tablespace]**

**[TEMPORARY TABLESPACE tablespace]**

**[QUOTA int {k | M} ON tablespace]**

**[QUOTA UNLIMITED ON tablespace]**

* Eliminar usuarios:

**DROP USER nombreUsuario [CASCADE];**

Permisos

**GRANT {privilegioObjeto [,privilegioObjeto]… |ALL [PRIVILEGES]}**

**ON [usuario.]objeto**

**TO {usuario1 | rol1 | PUBLIC} [,usuario2 | rol2 | PUBLIC] …**

**[WITH GRANT OPTION]**

**[WITH ADMIN OPTION];**

Donde:

* ON , especifica la tabla sobre la que se conceden privilegios (verduritas)
* TO, señala a los usuarios o roles a los que se conceden
* ALL, concede todos los privilegios sobre el objeto especificado
* [WITH GRANT OPTION], permite que el receptor del privilegio se lo asigne a otro
* [WITH ADMIN OPTION], se usa cuando se conceden privilegios de sistema y este puedo concederselos a otros usuarios
* PUBLIC, hace que el privilegio esté disponible para todos los usuarios

\*Privilegios sobre tablas:

* Select
* Insert
* Update

\*Privilegios de sistema:

* Create session
* Create table
* Create user
* Drop user

\*Privilegios sobre una columna:

**GRANT select/update/insert (nombreColumna) on nombreTabla**

**TO usuario/public;**

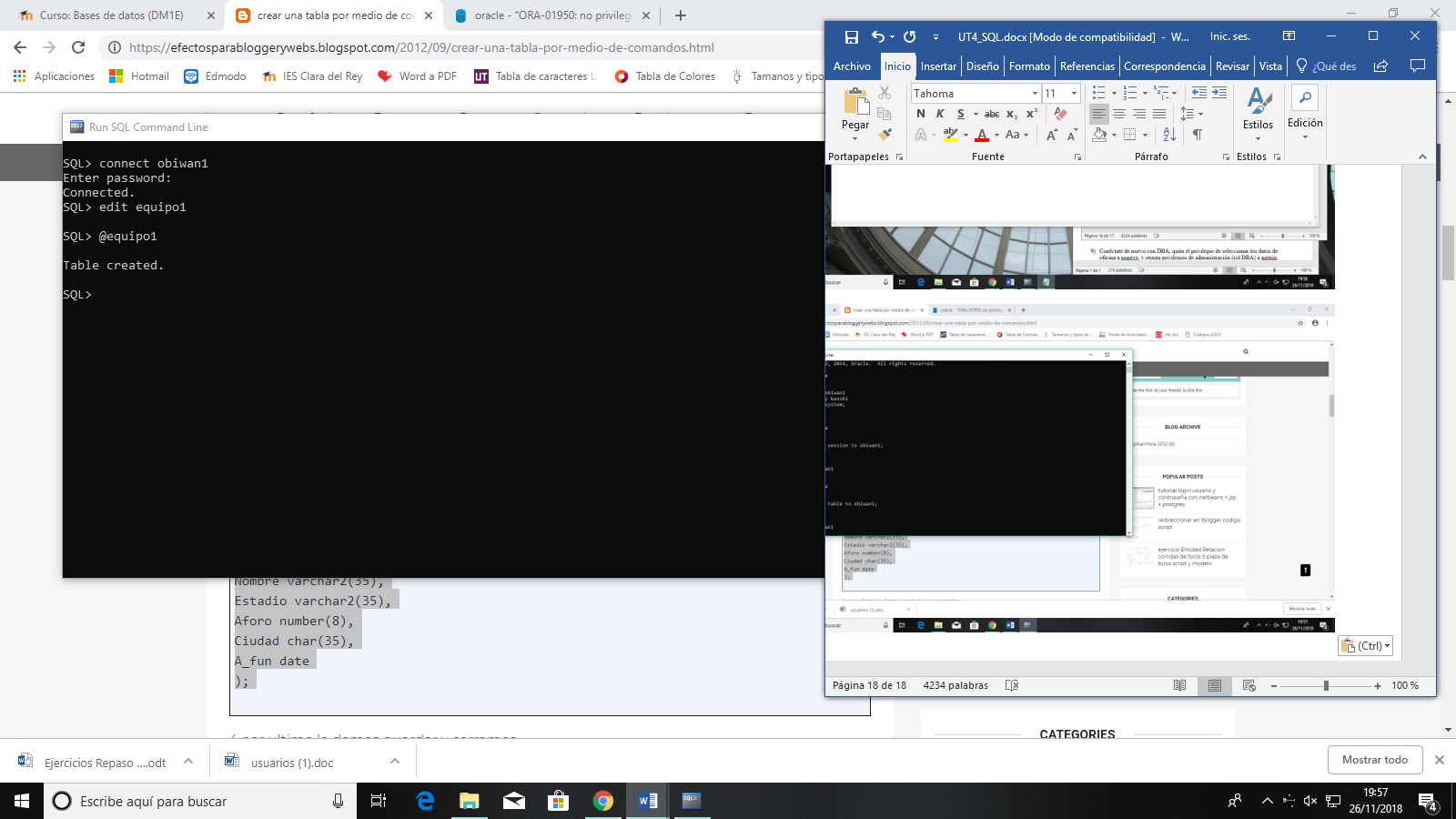
\*Quitar privilegios:

Solo se pueden quitar los privilegios que se han dado.

**REVOKE nombrePrivilegio/ALL privileges**

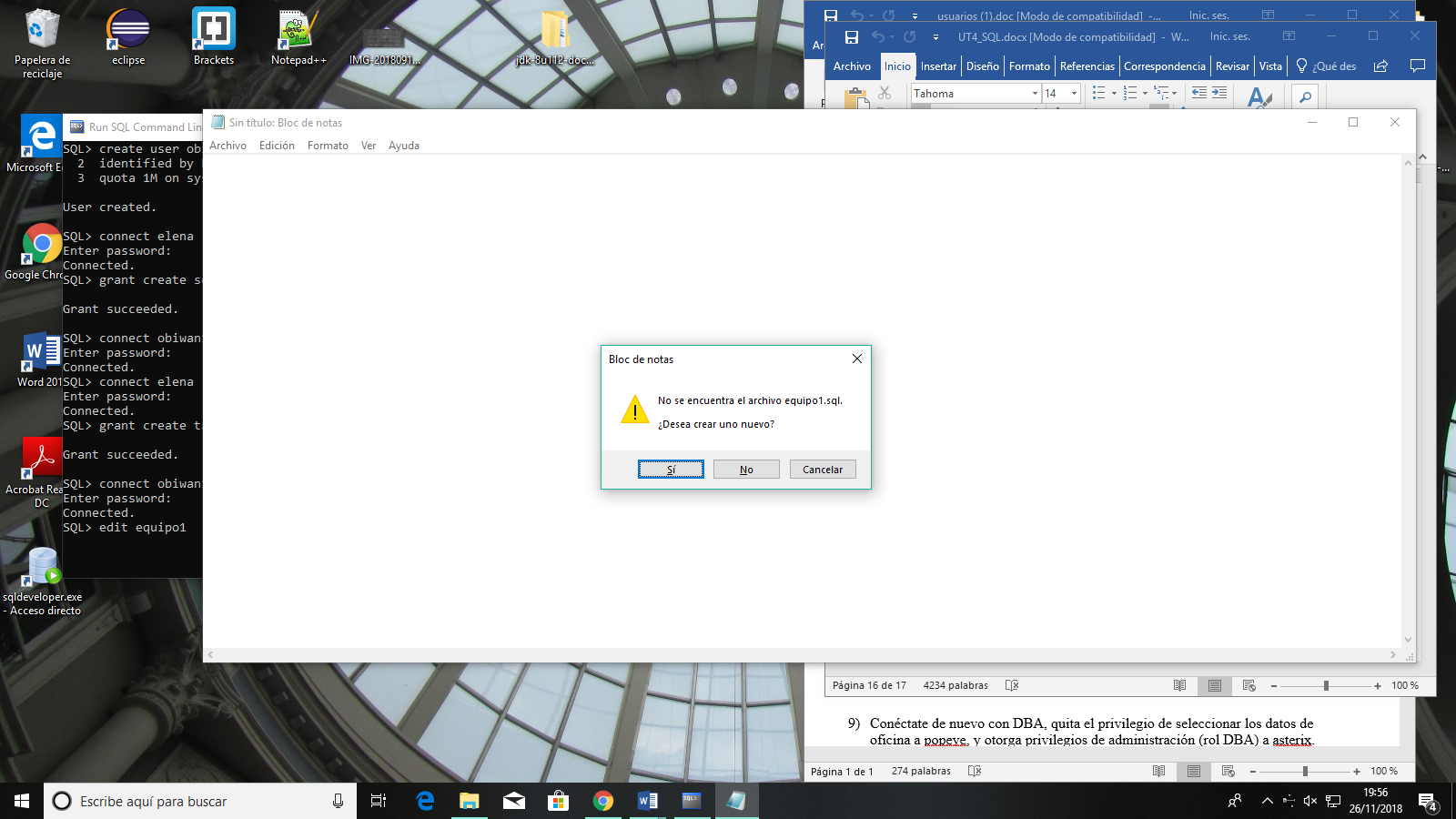
**FROM usuario;**

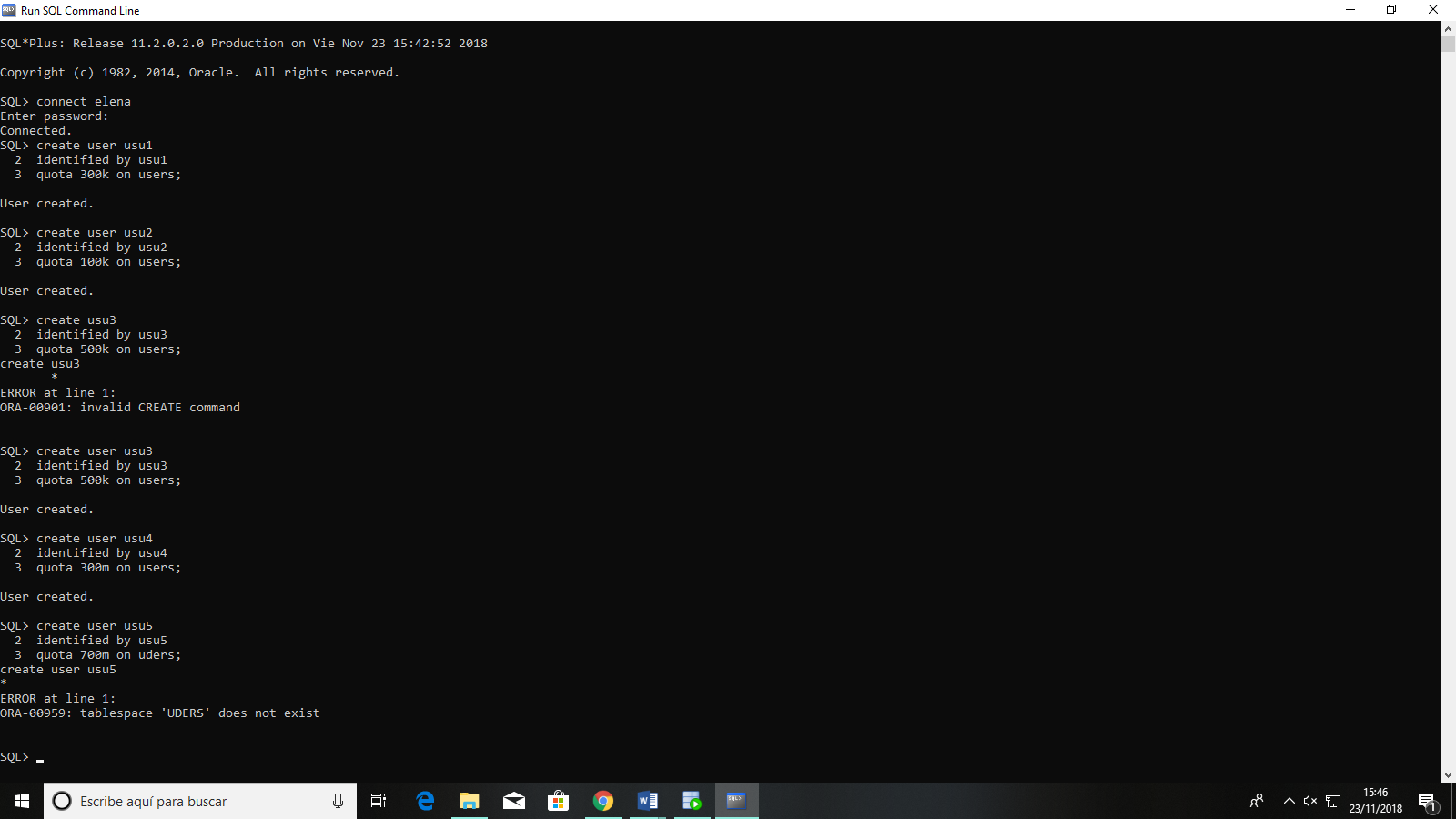
\*Crear tablas:

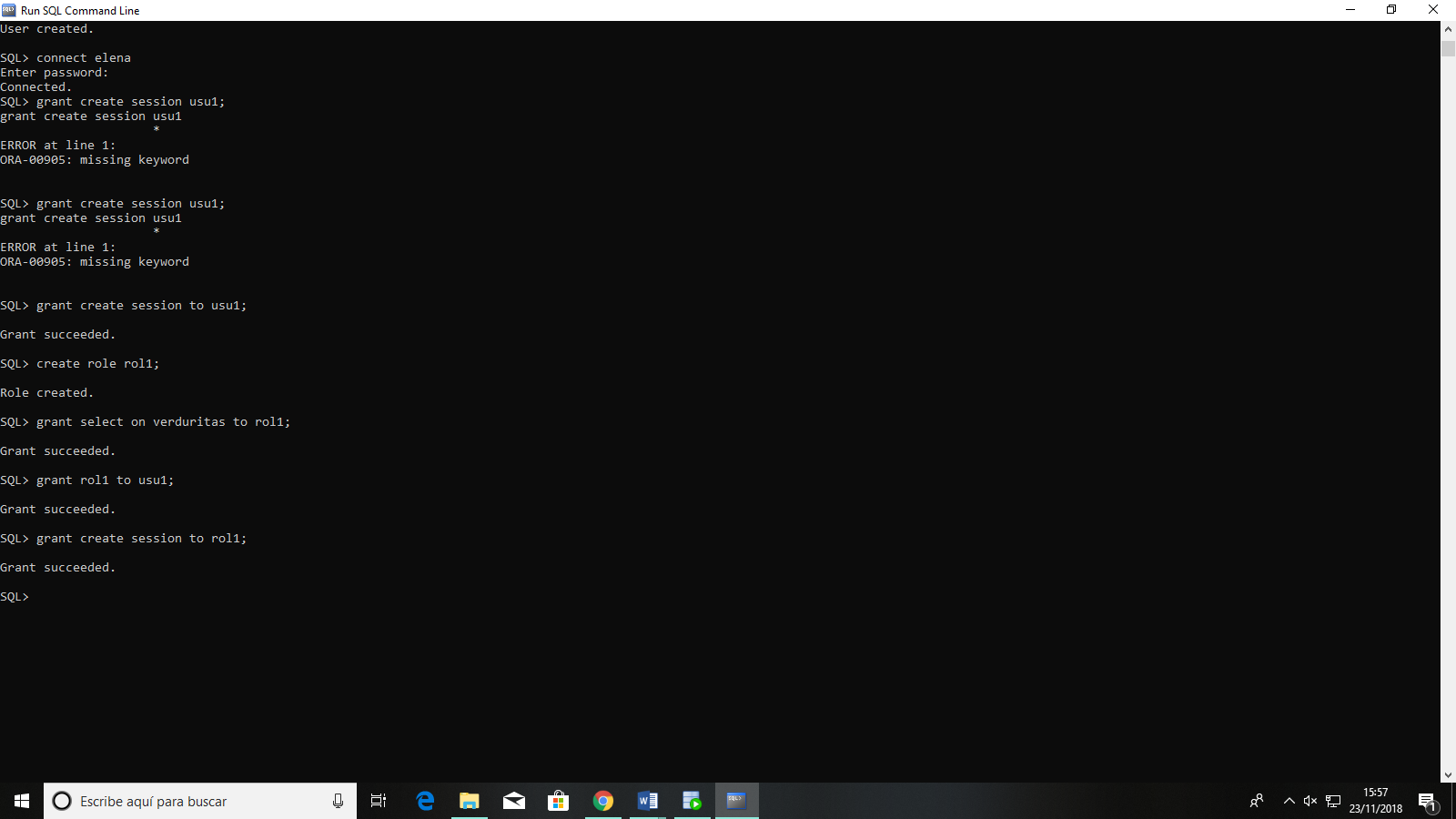


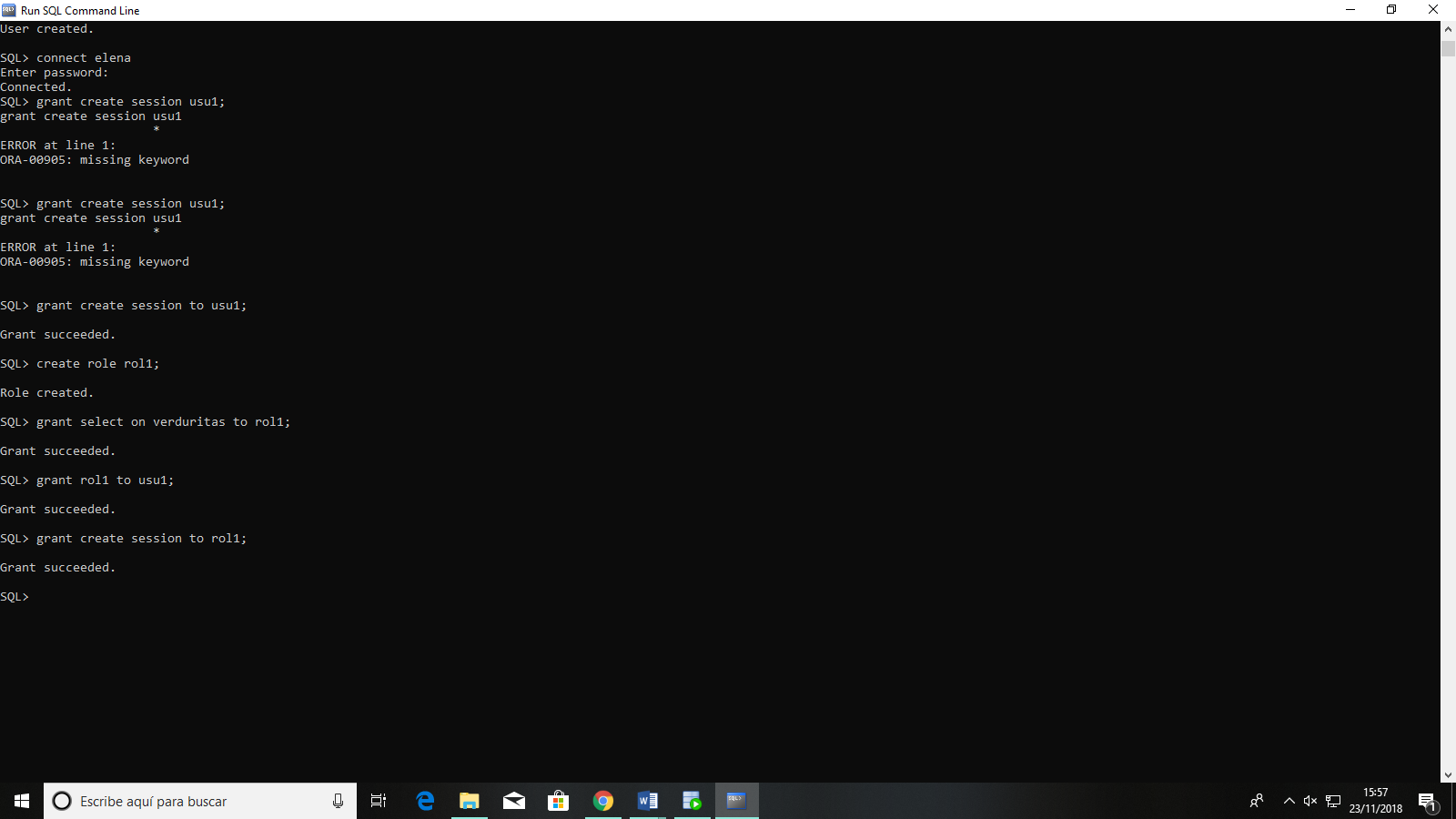
Intento editar con obiwan1 la tabla equipo1, como no existe me sale esta advertencia (imagen de abajo) le doy a Si, creo la tabla, guardo y cierro.

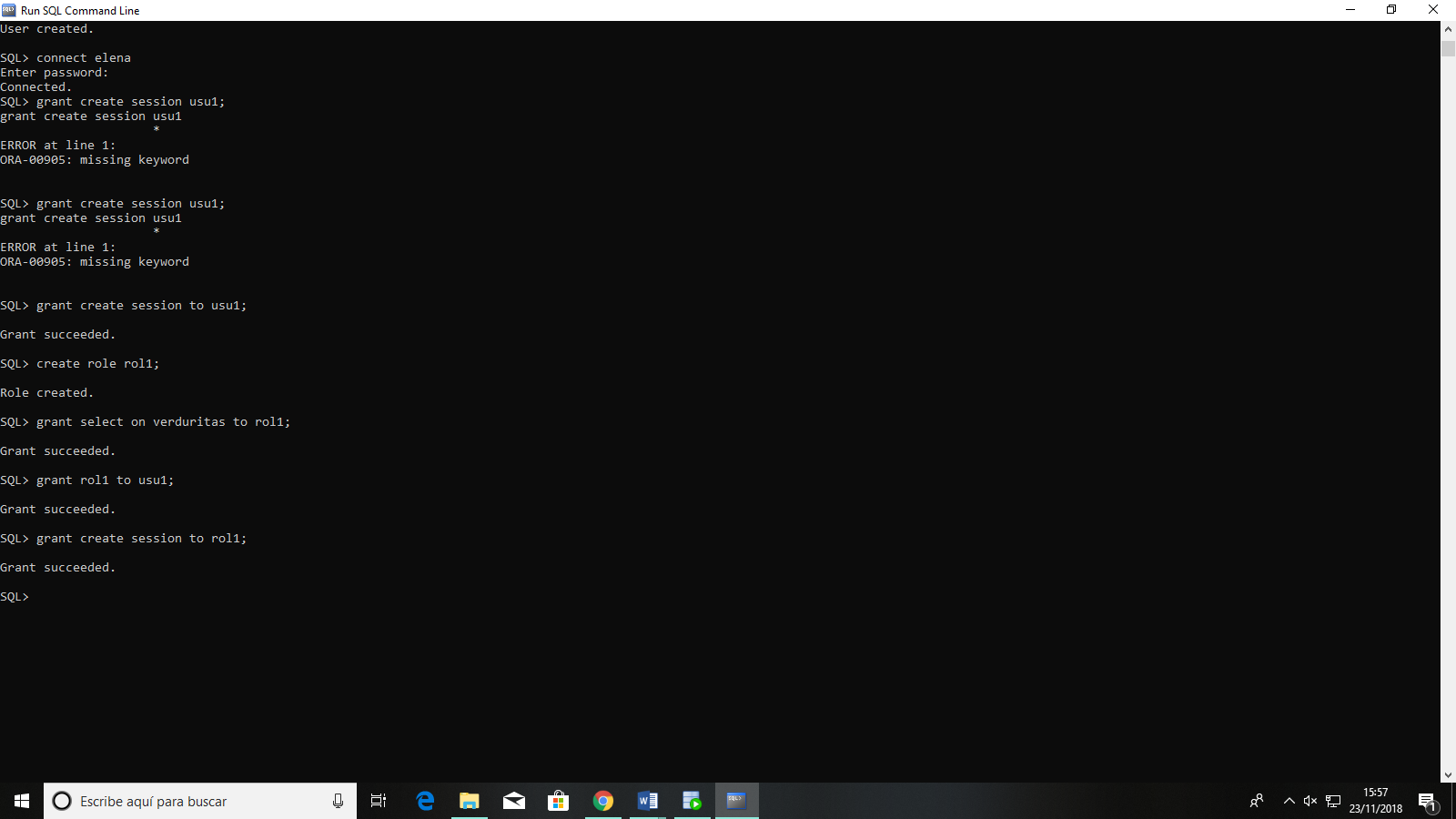
Luego pongo @equipo y ya esta

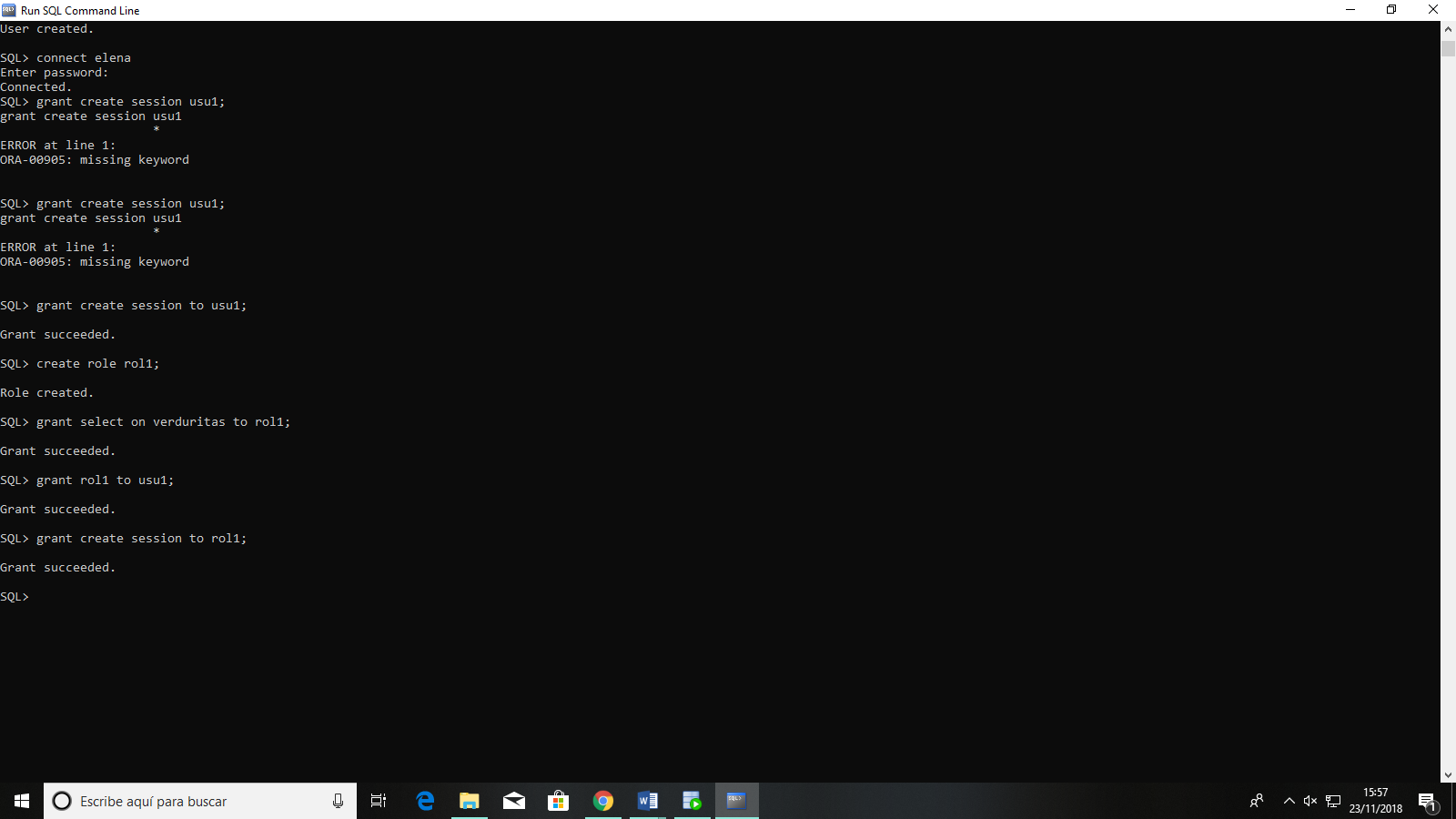


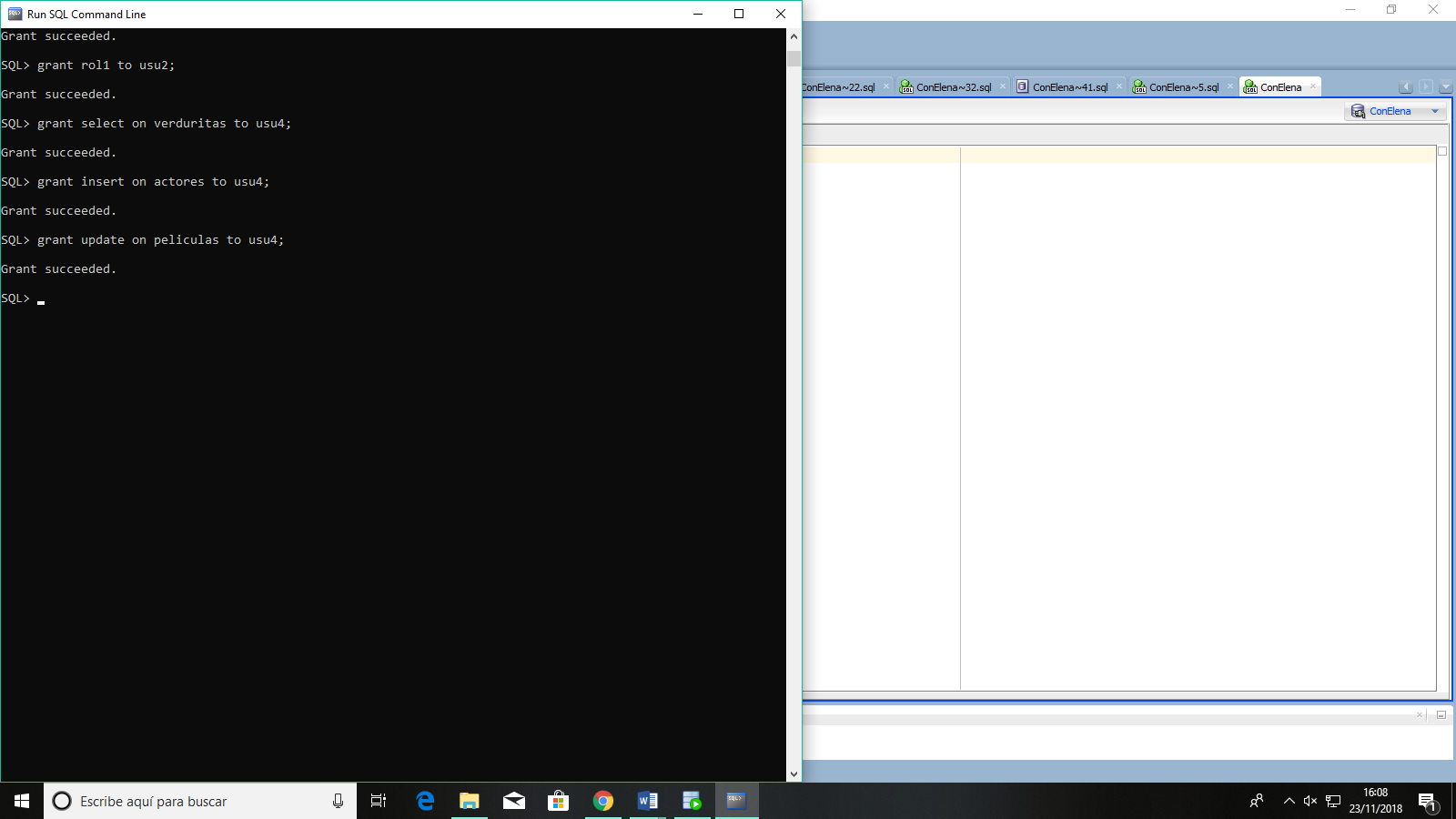


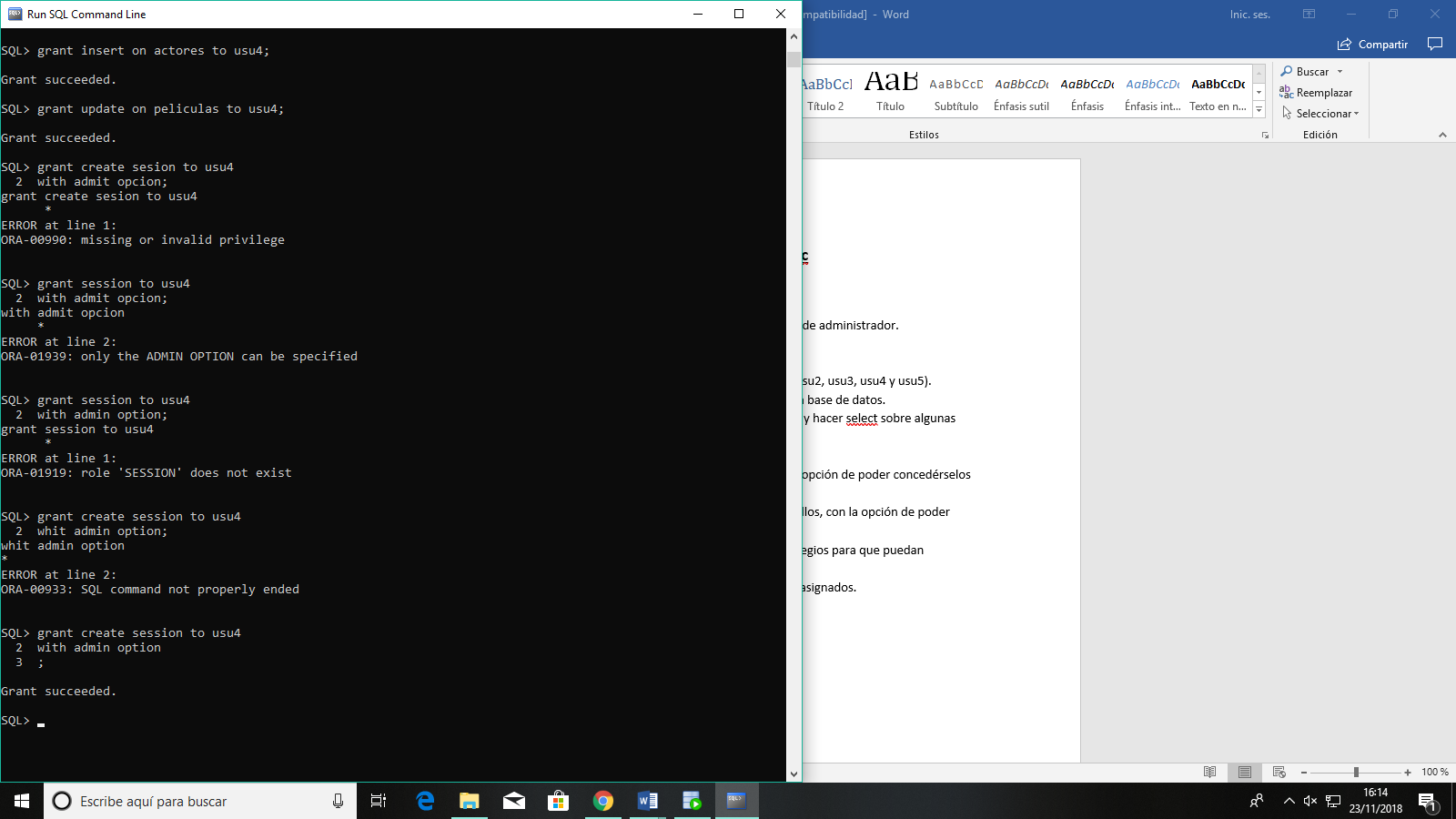












CREAR USUARIOS

CREAR TABLAS

MODIFICAR TABLAS

DAR ROLES Y PERMISOS

UNIDAD 5

# Consultas

## CADENAS

INITCAP(cad) -> devuelve la cadena cad con la primera letra en mayúsculas.

LPAD(cad1, n, cad2) -> devuelve la cadena cad1 de longitud n rellenada por la izquierda con cad2. LPAD(‘M’, 5, ‘\*’) : \*\*\*\*M

RPAD(cad1, n, cad2) -> devuelve la cadena cad1 de longitud n rellenada por la derecha con cad2. RPAD(‘M’, 5, ‘\*’) : M\*\*\*\*

REPLACE(cad, ant, nue) -> reemplaza en cad las cadenas ant por las nue.

LENGTH(cad) -> longitud de cadena.

## FECHAS

SYSDATE

SYSTIMSTAMP -> fecha y hora actuales

MONTHS\_BETWEEN(fecha1, fecha2)

LAST\_DAY(fecha) -> último día del mes al que pertenece fecha

EXTRACT(valor FROM fecha)

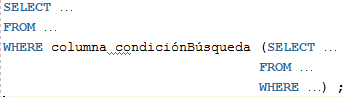
## OTRAS

DECODE (expr1, cond1, valor1 [,cond2 ,valor2, …] default ) -> Esta función evalúa una expresión **expr1**, si se cumple la primera condición **cond1** devuelve el valor **valor1,**  en caso contrario se pasa a la siguiente condición y así hasta que se cumpla una de ellas o se devuelve un valor por defecto expresado por **default** si no se cumple ninguna. Es como el if en programación.

UNIDAD 6

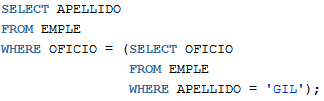
# Subconsultas

Primero se realiza la subconsulta, y con el resultado de esta, se realiza la consulta inicial.



Subconsulta

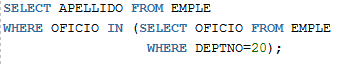
Ejemplo: obtener el APELLIDO de los empleados con el mismo oficio que ‘GIL’.



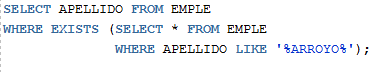
## CONDICIONES DE BÚSQUEDA

**Test de comparación en subconsultas (>, <, <>, <=, >=, =)**. Compara el valor de una expresión con un valor único producido por una subconsulta.

**Test de pertenencia a un conjunto devuelto por una subconsulta (IN)**. Ejemplo: obtener los apellidos de los empleados cuyo oficio sea alguno de los oficios que hay en el departamento 20.



**Test de existencia (EXISTS, NOT EXISTS).** Examina si una subconsulta produce alguna fila de resultados. El test será TRUE si devuelve filas y FALSE en caso contrario. Visualizar los APELLIDOS de los empleados si existe alguno llamado ‘ARROYO’. Se utilizará la claúsula NOT EXISTS si queremos comprobar si la subconsulta no devuelve ninguna fila.

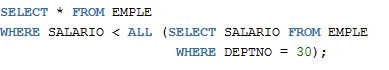


**Test de comparación cuantificada (ANY y ALL).** Se utilizan junto a los operadores de comparación (>, <, <>, <=, >=, =).

**ANY** compara el valor de una expresión con cada uno del conjunto de valores devuelto por la subconsulta, si alguna de las comparaciones individuales da resultado true, ANY devolverá true, si la sulbconsulta devuelve false, ANY devolverá false. Ejemplo, obtener los datos de los empleados cuyo salario sea igual a algún salario de los empleados del departamento 30.

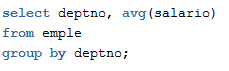


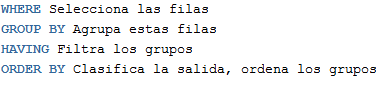
**ALL** compara el valor de una expresión con cada uno del conjunto de valores devuelto por la subconsulta, si todas las comparaciones individuales da como resultado true, ALL devuelve true, en caso contrario devuelve false. Ejemplo: obtener los datos de los empleados cuyo salario sea menor a cualquier salario de los empleados del departamento 30.



## GROUP BY

Para averiguar el salario medio de cada departamento necesitaremos realizar un agrupamiento por departamento y utilizaremos la cláusula **GROUP BY**. La consulta quedaría de la siguiente forma:





**\*Los datos seleccionados en la sentencia SELECT que lleva el GROUP BY deben ser: una constante, una función de grupo (SUM, AVG, COUNT, …), una columna expresada en el GROUP BY.\***

# Unión, Intersección y Diferencia

## OPERADOR UNION

Este operador combina resultados de dos consultas. Permite añadir el resultado de un SELECT a otro SELECT. Las filas duplicadas que aparecen se reducen a una única fila.

Ejemplo:



El resultado será una tabla que contendrá nombres de provincias y de comunidades. Se crea una sola tabla con registros incluidos en cualquiera de las consultas. Se pueden mostrar los registros duplicados especificando **UNION ALL.**

## OPERADOR INTERSECT

Devuelve las filas que son iguales en ambas consultas. Todas las filas duplicadas serán eliminadas antes de la generación del resultado final.

Ejemplo: visualizar los tipos y modelos de piezas que se encuentran en los almacenes 1 y 2.



## OPERADOR MINUS

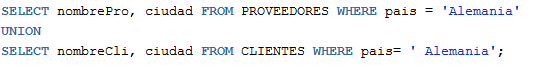
Devuelve las filas que estén en la primera SELECT y no en la segunda. Las filas duplicadas del primer conjunto se reducirán a una única fila antes de que empiece la comparación con el otro conjunto.

Ejemplo: tipos y modelos de piezas que se encuentren en el almacén 1 y no en el 2.

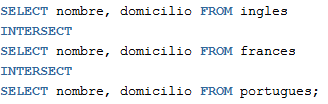


**Ejemplos:**

Obtener los nombres y ciudades de todos los proveedores y clientes de Alemania



Una academia de idiomas da clases de inglés, francés y portugués y guarda los datos de los alumnos en tres tablas cuyo nombre coincide con el idioma. Averiguar el nombre y domicilio de todos los alumnos que cursan los tres idiomas para enviarles información sobre los exámenes.



Averiguar los nombres y domicilios de los alumnos que cursan inglés y no portugués.



UNIDAD 7

# Insert, Update y Delete

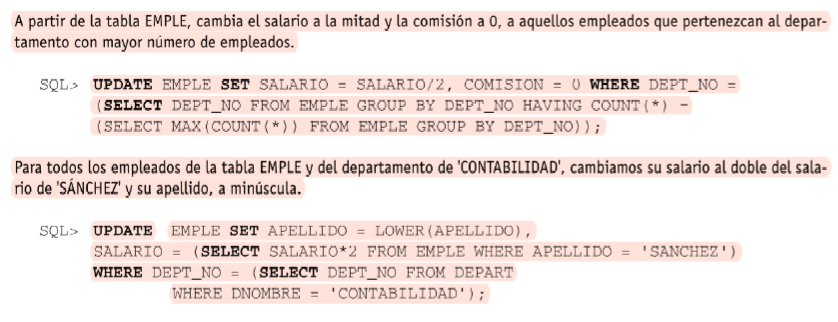
## ORDEN INSERT

­­­­

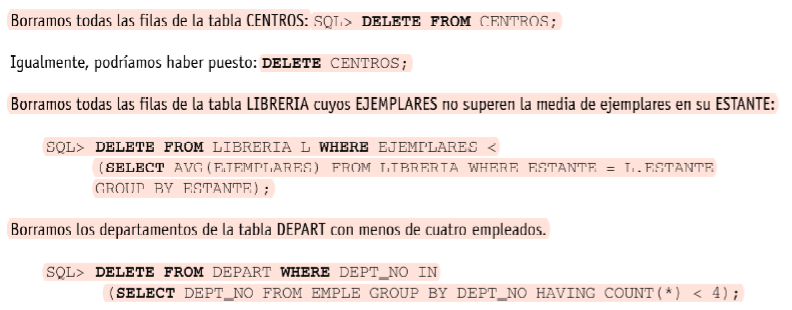


## ORDEN UPDATE





## ORDEN DELETE



# Commit, Autocommit y Rollback

## ORDEN COMMIT

Validar cambios realizados en la base de datos:



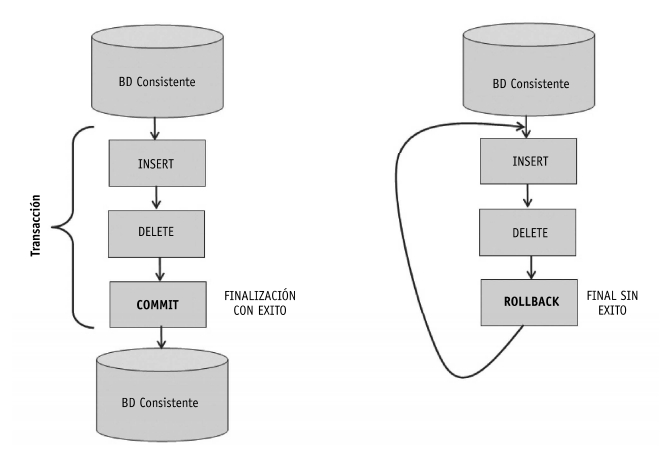
## ORDEN AUTOCOMMIT

Validar automáticamente sin necesidad de poner commit todo el rato. Para ver el estado de auto commit, pondremos “SHOW AUTOCOMMIT;” y para activarlo, ponderemos “SET AUTOCOMMIT ON;”

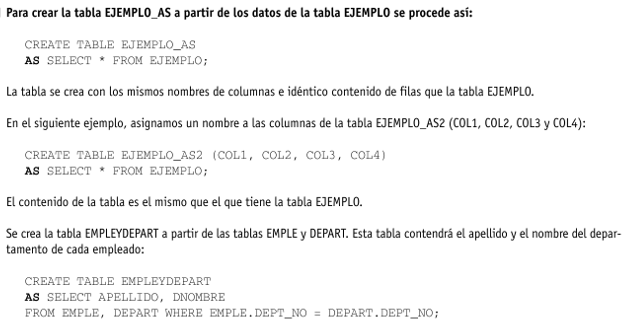
## ORDEN ROLLBACK

Ignora los cambios hechos después del último commit:





# Creación de tablas a partir de otras

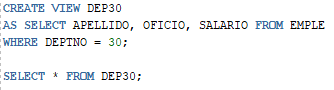




# Creación de vistas

Sirven para obtener los datos de una consulta compleja. Si se suprime una tabla la vista se invalida.

\*ES ACONSEJABLE PONER SIEMPRE CREATE OR REPLACE VIEW\*



Visualizamos la vista



Ponemos nombre nuevo a las columnas

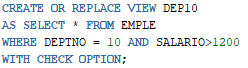
## INSERTAR VALORES EN VISTAS



## OPCIONES DE CREACIÓN DE VISTA

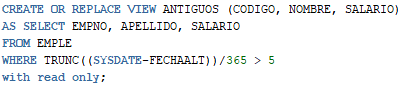
### WITH CHECK OPTION

Antes de insertar una nueva fila o de actualizar la vista, se chequea que se cumplen las condiciones de la vista:



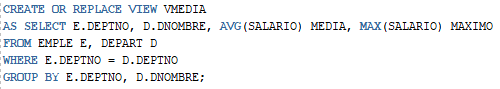
### WITH READ ONLY

Solo nos permite hacer SELECT, no podremos modificarla:



## VISTAS COMPLEJAS

Son las que se crean a partir de mas de una tabla



Estas vistas no se podrán modificar.

## ELIMINAR VISTAS

Para eliminar vistas utilizaremos el siguiente comando:



# Creación de secuencias

Sirven para generar números automáticamente.

