

TEMA - V:

TRATAMIENTO DE LOS DATOS

BD - (1º DAM)

ÍNDICE

1. Lenguaje de Manipulación de Datos (LMD).....	5
1.1. INSERCIÓN DE VALORES EN UNA TABLA.....	5
1.1.1. REPLACE.....	8
1.2. BORRADO DE VALORES DE UNA TABLA.....	9
1.3. MODIFICACIÓN DE VALORES DE UNA TABLA.....	11
1.3.1. MODIFICACIÓN ESPECIAL A TRAVÉS DE CONSULTA.....	13
1.4. CLAUSULA AUTO_INCREMENT	14
1.5. EJEMPLOS	15
2. UTILIZACIÓN DE VARIABLES DE USUARIO.....	18
3. COMANDOS TRANSACCIONALES Y DE BLOQUEO DE TABLAS	19
3.1. Sintaxis de START TRANSACTION, COMMIT y ROLLBACK	19
3.2. Sentencias que no se pueden deshacer.....	19
3.3. Sentencias que causan una ejecución (commit) implícita.....	19
3.4. Sintaxis de SAVEPOINT y ROLLBACK TO SAVEPOINT.....	21
3.5. BLOQUEO de Tablas	22
3.5.1. Sintaxis de LOCK TABLES	22
3.5.1. Sintaxis de UNLOCK TABLES	23
3.6. Sintaxis de SET TRANSACTION	23
4. Las VISTAS	24
4.1. Sintaxis de CREATE VIEW	24
4.1.1. La cláusula WITH CHECK OPTION.....	25
4.2. Sintaxis de ALTER VIEW.....	28
4.3. Sintaxis de DROP VIEW.....	28
4.4. Sintaxis de SHOW CREATE VIEW.....	28
4.5. TIPOS de VISTAS.....	29
4.6. OPERACIONES sobre VISTAS	29
5. SEGURIDAD en Oracle.....	30
5.1. USUARIOS.....	30
5.1.1. CREACIÓN DE USUARIOS.....	30
5.1.2. MODIFICACIÓN DE USUARIOS.....	31
5.1.3. BORRADO DE USUARIOS.....	32

5.2. PRIVILEGIOS.....	32
5.2.1. PRIVILEGIOS SOBRE LOS OBJETOS.....	34
5.2.2. PRIVILEGIOS DEL SISTEMA.....	35
5.2.3. RETIRADA DE PRIVILEGIOS.....	36
5.2.4. VISTAS CON INFORMACIÓN DE PRIVILEGIOS.....	37
5.3. ROLES	37
5.3.1. LÍMITES EN PRIVILEGIOS SOBRE ROLES.....	38
5.3.2. SUPRESIÓN DE PRIVILEGIOS EN LOS ROLES	39
5.3.3. SUPRESIÓN DE UN ROL.....	39
5.3.4. ESTABLECER UN ROL POR DEFECTO	39
5.3.5. INFORMACIÓN SOBRE ROLES EN EL DICCIONARIO DE DATOS	39
5.4. PERFILES	40
5.4.1. MODIFICACIÓN DE UN PERFIL.....	40
5.4.2. BORRADO DE UN PERFIL.....	40
5.5. El DICCIONARIO de Datos	41
6. EJERCICIOS y PRACTICAS	42
6.1. Ejercicios LMD	42
6.1.1. Ejercicios RESUELtos	43
6.1.2. Ejercicios PROPUESTos	44
6.2. PRACTICAS con VISTAS (MySQL).....	45
6.2.1. Prácticas RESUeltas	45
6.2.2. Prácticas PROPUESTAS	45
6.2.2.1. Práctica - A	45
6.2.2.2. Práctica - B	46
6.2.2.3. Práctica - C	46
6.2.2.4. Práctica - D	46
6.3. PRACTICAS de SEGURIDAD con Oracle.....	47
6.3.1. Prácticas RESUeltas	47
6.3.1.1. Práctica – A	47
6.3.1.2. Prácticas – B.....	49
6.3.2. Prácticas PROPUESTAS	75
6.3.2.1. Práctica – A	75
6.3.2.2. Práctica – B	75
6.3.2.3. Práctica – C	75
6.3.2.4. Práctica – D	76
6.3.2.5. Práctica – E	76

1. Lenguaje de Manipulación de Datos (LMD).

1.1. INserción de Valores en una Tabla.

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
       [INTO] tbl_name [(col_name,...)]
       VALUES ({expression | DEFAULT},...),(...),...
       [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

O

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
       [INTO] tbl_name
       SET col_name={expression | DEFAULT}, ...
       [ ON DUPLICATE KEY UPDATE col_name=expression, ... ]
```

O

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
       [INTO] tbl_name [(col_name,...)]
       SELECT ...
```

INSERT inserta nuevas filas en una tabla existente. Los formatos **INSERT ... VALUES** e **INSERT ... SET**, insertas filas basándose en los valores especificados explícitamente. El formato **The INSERT ... SELECT** inserta filas seleccionadas de otra tabla o tablas. El formato **INSERT ... VALUES** con una lista de múltiples valores está soportada por MySQL desde la versión 3.22.5. La sintaxis **INSERT ... SET** está soportada por MySQL desde la versión 3.22.10.

tbl_name es la tabla donde se insertarán las filas. Las columnas para las que la sentencia proporciona valores se pueden especificar de las siguientes formas:

- Las lista de nombres de columnas o la cláusula **SET** indican las columnas explícitamente.
- Si no se especifica una lista de columnas para **INSERT ... VALUES** o **INSERT ... SELECT**, se deben proporcionar **valores para todas las columnas** en la lista **VALUES()** o por **SELECT**. Si no se conoce el orden de las columnas dentro de la tabla, usar **DESCRIBE** *tbl_name* para encontrarlo.

Los valores de columnas se pueden proporcionar de varias formas:

- Si se está ejecutando MySQL en el modo estricto (strict mode), a cualquier columna para la que **no se proporcione un valor** explícitamente se le asignará su valor por defecto (explicito o隐式). Por ejemplo, si se especifica una lista de columnas que no contiene el nombre de todas las columnas de la tabla, a las columnas sin nombre le serán asignados sus **valores por defecto**.
- Se puede usar la palabra clave **DEFAULT** para poner una columna a su valor por defecto de forma explícita. (Nuevo en MySQL 4.0.3.) Esto hace más sencillo escribir sentencias **INSERT** que asignan valores a casi todas las columnas, porque permite la escritura de una **lista VALUES incompleta**, que no incluye un valor para cada columna de la tabla. En otro caso, se debería escribir la lista de nombres de columnas correspondiente a cada valor en la lista **VALUES**. A partir de MySQL 4.1.0, se puede usar **DEFAULT(*col_name*)** como una forma más general que puede ser usada en expresiones para obtener valores por defecto de columnas.
- Si tanto la lista de columnas como la de **VALUES** están vacías, **INSERT** crea una fila en la que cada columna tendrá su valor por defecto:

```
mysql> INSERT INTO tbl_name () VALUES();
```

- Se puede especificar una **expresión expr** para proporcionar un valor de columna. Esto forzará complejas conversiones de tipo si el de la expresión no coincide con el tipo de la columna, y la conversión de un valor dado puede provocar diferentes valores insertados dependiendo del tipo de la columna. Por ejemplo, insertar la cadena '1999.0e-2' en una columna INT, FLOAT, DECIMAL(10,6) o YEAR producirá los valores 1999, 19.9921, 19.992100 y 1999. El motivo es que el valor almacenado en una columna INT y YEAR sea 1999 es que la conversión de cadena a entero mira sólo la parte inicial de la cadena que se pueda considerar un valor entero o un año válido. Para columnas en punto flotante o punto fijo, la conversión de cadena a punto flotante tiene en cuenta la cadena completa como un valor válido en coma flotante. Una expresión **expr** se puede referir a cualquier columna que se haya asignado previamente en la lista de valores. Por ejemplo, se puede hacer esto, ya que el valor de col2 se refiere a col1, que ya ha sido asignado:

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

Pero no se puede hacer esto, porque el valor para col1 se refiere a col2, que se asigna después de col1:

```
mysql> INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

La excepción es para columnas que contengan valores autoincrementados. Esto es debido a que el valor **AUTO_INCREMENT** se genera después de la asignación de cualquier otro valor, así que cualquier referencia a una columna **AUTO_INCREMENT** devolverá un 0.

La sentencia **INSERT** soporta los **modificadores** siguientes:

- Si se especifica la palabra clave **DELAYED**, el servidor coloca la fila o filas a insertar en un búfer, y el cliente lanza la sentencia **INSERT DELAYED** y puede continuar. Si la tabla está ocupada, el servidor guarda las filas. **Cuando la tabla queda libre, empieza a insertar filas**, verificando periódicamente para ver si hay nuevas peticiones de lectura para la tabla. Si las hay, la cola de inserción retardada se suspende hasta que la tabla quede libre de nuevo. **DELAYED** se añadió en MySQL 3.22.5.
- Si se especifica la palabra clave **LOW_PRIORITY**, la ejecución de la sentencia **INSERT** se retrasa hasta que no haya clientes leyendo de la tabla. Esto incluye a otros clientes que empiecen a leer mientras existan clientes leyendo, y mientras la sentencia **INSERT LOW_PRIORITY** está esperando. Por lo tanto, es posible que un cliente que lance una sentencia **INSERT LOW_PRIORITY** permanezca esperando por mucho tiempo (eventualmente para siempre) en un entorno con muchas lecturas. (Esto contrasta con lo que sucede con **INSERT DELAYED**, que permite al cliente continuar inmediatamente). **LOW_PRIORITY** no debe ser usado con tablas **MyISAM** que no permiten inserciones concurrentes.
- Si se especifica la palabra clave **HIGH_PRIORITY**, se anula el efecto de la opción **--low-priority-updates** si el servidor fue arrancado con esa opción. Esto también hace que no se puedan usar inserciones concurrentes. **HIGH_PRIORITY** fue añadido en MySQL 3.23.11.
- Si se especifica la palabra clave **IGNORE** en una sentencia **INSERT**, los errores que se produzcan mientras se ejecuta la sentencia se tratan con avisos. Por ejemplo, sin **IGNORE** una fila que duplique un valor de clave **PRIMARY** o **UNIQUE** existente en la tabla provocará un error y la sentencia será abortada. **Con IGNORE, el error se ignora y la fila no será insertada**. Las conversiones de datos que produzcan errores aborta la sentencia si no se especifica **IGNORE**. Con **IGNORE**, los valores inválidos se ajustarán al valor más próximos y se insertarán; se producirán avisos pero la sentencia no se

aborta. Se puede determinar cuantas filas fueron insertadas mediante la función del API [mysql_info](#).

Si se especifica la cláusula *ON DUPLICATE KEY UPDATE* (nueva en MySQL 4.1.0), y se inserta una fila que puede provocar un valor duplicado en una clave *PRIMARY* o *UNIQUE*, se realiza un *UPDATE* (actualización) de la fila antigua. Por ejemplo, si se declara una columna 'a' como *UNIQUE* y ya contiene el valor 1, las dos sentencias siguientes tienen el mismo efecto:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
      -> ON DUPLICATE KEY UPDATE c=c+1;
mysql> UPDATE table SET c=c+1 WHERE a=1;
```

El valor de filas afectadas es 1 si la fila es insertada como un nuevo registro y 2 si se actualiza un registro ya existente.

Nota: si la columna 'b' es única también, la sentencia **INSERT** puede ser equivalente a esta sentencia [UPDATE](#):

```
mysql> UPDATE table SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

Si a=1 OR b=2 se cumple para varias filas, sólo una será actualizada. En general, se debe intentar evitar el uso de la cláusula *ON DUPLICATE KEY* en tablas con múltiples claves *UNIQUE*.

Desde MySQL 4.1.1 es posible usar la función **VALUES(*col_name*)** en una cláusula *UPDATE* para referirse a los valores de columna en la parte *INSERT* de una sentencia *INSERT ... UPDATE*. En otras palabras, **VALUES(*col_name*)** en una cláusula *UPDATE* se refiere al valor *col_name* que será insertado si no existe un conflicto de clave duplicada. Esta función es especialmente corriente en inserciones de varias filas. La función **VALUES** sólo tiene sentido en sentencias *INSERT ... UPDATE* y devuelve *NULL* en otro caso.

Ejemplo:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3), (4,5,6)
      -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

El comando anterior es idéntico a las dos sentencias siguientes:

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3)
      -> ON DUPLICATE KEY UPDATE c=3;
mysql> INSERT INTO table (a,b,c) VALUES (4,5,6)
      -> ON DUPLICATE KEY UPDATE c=9;
```

Cuando se usa *ON DUPLICATE KEY UPDATE*, la opción *DELAYED* se ignora.

Se puede encontrar el valor usado para una columna *AUTO_INCREMENT* usando la función [LAST_INSERT_ID\(\)](#). Desde el API C, usar la función [mysql_insert_id](#). Sin embargo, notar que las dos funciones no se comportan de forma idéntica en todas las circunstancias.

Si se usa una sentencia *INSERT ... VALUES* con una lista de múltiples valores o *INSERT ... SELECT*, la sentencia devuelve una cadena de información con este formato:

```
Records: 100 Duplicates: 0 Warnings: 0
```

Records indica el número de filas procesadas por la sentencia. (No es necesariamente el número de filas insertadas. "Duplicates" puede ser distinto de cero.) **Duplicates** indica el número de filas que no pudieron ser insertadas porque contienen algún valor para un índice único ya existente. **Warnings** indica el número de intentos de inserción de valores de columnas que han causado algún tipo de problemas. Se pueden producir "Warnings" bajo cualquiera de las siguientes condiciones:

- **Inserción de *NULL* en una columna declarada como *NOT NULL*.** Para sentencias **INSERT** de varias filas o en sentencias **INSERT ... SELECT**, se asigna el valor por defecto apropiado al tipo de columna. Ese valor es 0 para tipos numéricos, la cadena vacía ("") para tipos cadena, y el valor "cero" para tipos de fecha y tiempo.
- Asignación de un **valor fuera de rango** a una columna numérica. El valor se recorta al extremo apropiado del rango.
- Asignación de un valor como '**10.34 a' a un campo numérico**. La parte añadida se ignora y **se inserta sólo la parte numérica** inicial. Si el valor no tiene sentido como un número, el valor asignado es 0.
- Inserción de un valor a una columna de cadena (**CHAR, VARCHAR, TEXT o BLOB**) que exceda la longitud máxima para la columna. **El valor se trunca a la longitud máxima** de la columna.
- Inserción de un valor dentro de una columna de fecha o tiempo que sea ilegal para el tipo de columna. Se asigna a la columna el valor apropiado de cero, según su tipo.

1.1.1. REPLACE.

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
VALUES ({expr | DEFAULT},...),(...),...
```

O:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name
SET col_name={expr | DEFAULT}, ...
```

O:

```
REPLACE [LOW_PRIORITY | DELAYED]
[INTO] tbl_name [(col_name,...)]
SELECT ...
```

REPLACE trabaja exactamente igual que **INSERT**, excepto que si existe algún registro viejo en la tabla que tenga el mismo valor que uno nuevo para un índice **PRIMARY KEY** o **UNIQUE**, el viejo se borra antes de que el nuevo sea insertado.

Hay que tener en cuenta que salvo que la tabla tenga una **PRIMARY KEY** o un índice **UNIQUE**, usar una sentencia **REPLACE** no tiene sentido. En ese caso es equivalente usar una sentencia **INSERT**, ya que no hay ningún índice que se pueda usar para determinar si una nueva fila duplica a otra.

Para poder usar **REPLACE**, se deben poseer los privilegios **INSERT** y **DELETE** para la tabla.

La sentencia **REPLACE** devuelve un **contador para indicar el número de filas afectadas**. Ese número es la suma de filas borradas e insertadas. Si el contador es 1 para un **REPLACE** de una única fila, la fila fue insertada y no se borró ninguna fila. Si el contador es mayor de 1, una o más de las viejas filas fue borrada antes de que la nueva fila fuese insertada. Es posible que una única fila reemplace a más de una fila vieja si la tabla contiene varios índices únicos y la nueva fila duplica valores de diferentes filas viejas en diferentes índices únicos.

1.2. BORRADO DE VALORES DE UNA TABLA.

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM table_name
      [WHERE where_definition]
      [ORDER BY ...]
      [LIMIT row_count]
```

Sintaxis multitable:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      table_name[.*] [, table_name[.*] ...]
      FROM table-references
      [WHERE where_definition]
```

O:

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
      FROM table_name[.*] [, table_name[.*] ...]
      USING table-references
      [WHERE where_definition]
```

DELETE elimina columnas desde "table_name" que satisfagan la condición dada por la "where_definition", y devuelve el número de registros borrados.

Si se usa una sentencia **DELETE** sin cláusula *WHERE*, todas las filas serán borradas. Una forma más rápida de hacer esto, cuando no se necesita conocer el número de filas eliminadas, es usar [TRUNCATE TABLE](#).

En MySQL 3.23, **DELETE** sin la cláusula *WHERE* retorna cero como número de filas afectadas.

En esta versión, si realmente se quiere saber cuántas filas fueron eliminadas cuando se borran todas, y se está dispuesto a sufrir una pérdida de velocidad, se puede usar una sentencia **DELETE** con una cláusula *WHERE* que siempre se cumpla. Por ejemplo:

```
mysql> DELETE FROM table_name WHERE 1>0;
```

Esto es mucho más lento que **DELETE FROM table_name** sin cláusula *WHERE*, porque borra filas una a una.

Si se borra la fila que contiene el valor máximo para una columna *AUTO_INCREMENT*, ese valor podrá ser usado por una tabla **ISAM** o **BDB**, pero no por una tabla **MyISAM** o **InnoDB**. Si se borran todas las filas de una tabla con **DELETE FROM tbl_name** (sin un WHERE) en modo *AUTOCOMMIT*, la secuencia comenzará de nuevo para todos los motores de almacenamiento, excepto para **InnoDB** y (desde MySQL 4.0) **MyISAM**. Hay algunas excepciones a este comportamiento para tablas **InnoDB**.

Para tablas **MyISAM** y **BDB**, se puede especificar una columna secundaria *AUTO_INCREMENT* en una clave multicolumna. En ese caso, la reutilización de los valores mayores de la secuencia borrados ocurre para tablas **MyISAM**.

La sentencia **DELETE** soporta los siguientes modificadores:

- Si se especifica la palabra **LOW_PRIORITY**, la ejecución de **DELETE** se retrasa hasta que no existan clientes leyendo de la tabla.
- Para tablas **MyISAM**, si se especifica la palabra **QUICK** entonces el motor de almacenamiento no mezcla los permisos de índices durante el borrado, esto puede mejorar la velocidad en ciertos tipos de borrado.

- La opción **IGNORE** hace que MySQL ignore todos los errores durante el proceso de borrado. (Los errores encontrados durante en análisis de la sentencia se procesan normalmente.) Los errores que son ignorados por el uso de esta opción se devuelven como avisos. Esta opción apareció en la versión 4.1.1.

La velocidad de las operaciones de borrado pueden verse afectadas por otros factores, como el número de índices o el tamaño del caché para índices.

El modificador **QUICK** afecta a si las páginas de índice se funden para operaciones de borrado. **DELETE QUICK** es más práctico para aplicaciones donde los valores de índice para las filas borradas serán reemplazadas por valores de índice similares para filas insertadas más adelante. En ese caso, los huecos dejados por los valores borrados serán reutilizados.

DELETE QUICK no es práctico cuando los valores borrados conducen a bloques de indices que no se llenan dejando huecos en valores de índice para nuevas inserciones. En ese caso, usar **QUICK** puede dejar espacios desperdiciados en el índice que permanecerán sin reclamar. Veamos un ejemplo de este tipo de problema:

1. Crear una tabla que contiene un índice en una columna **AUTO_INCREMENT**.
2. Insertar muchos registros en esa tabla. Cada inserción produce un valor de índice que es añadido al extremo mayor del índice.
3. Borrar un bloque de registros en el extremo inferior del rango de valores de la columna usando **DELETE QUICK**.

En este caso, los bloques de índices asociados con los valores de índice borrados quedan vacíos por debajo pero no se mezclan con otros bloques de índices debido al uso de **QUICK**. Esos bloques permanecen sin llenar cuando se insertan nuevas filas, ya que los nuevos registros no tienen valores de índice en el rango borrado. Además, permanecerán sin llenar aunque después se use **DELETE** sin **QUICK**, a no ser que alguno de los valores de índice borrados hagan que afecten a bloques de índices dentro o adyacentes a los bloques vacíos por debajo. Para liberar el espacio de índices no usado bajo estas circunstancias, se puede usar **OPTIMIZE TABLE**.

Si se van a borrar muchas filas de una tabla, puede ser mucho más rápido usar **DELETE QUICK** seguido de **OPTIMIZE TABLE**. Esto reconstruye el índice en lugar de relazar muchas operaciones de mezcla de bloques de índice.

La opción **LIMIT row count**, específica de MySQL para **DELETE** indica al servidor el máximo número de filas a borrar antes de que el control se devuelva al cliente. Esto se puede usar para asegurar que una sentencia **DELETE** específica no tomará demasiado tiempo. Se puede repetir la sentencia **DELETE** hasta que el número de filas afectadas sea menor que el valor de **LIMIT**.

Si se usa una cláusula **ORDER BY** las filas serán borradas en el orden especificado por la cláusula. Esto sólo será práctico si se usa en conjunción con **LIMIT**. Por ejemplo, la siguiente sentencia encuentra filas que satisfagan la cláusula **WHERE**, las ordena por tiempos, y borra la primera (la más antigua):

```
DELETE FROM somelog
  WHERE user = 'jcole'
    ORDER BY timestamp
      LIMIT 1
```

ORDER BY se puede usar con **DELETE** desde MySQL 4.0.0.

Desde MySQL 4.0, se pueden **especificar múltiples tablas en la sentencia DELETE** para eliminar filas de una o más tablas dependiendo de una condición particular en múltiples tablas. Sin embargo, **no es posible usar ORDER BY o LIMIT en un DELETE multitabla**.

La primera sintaxis de **DELETE** para tablas múltiples está soportada a partir de MySQL 4.0.0. La segunda desde MySQL 4.0.2. La parte 'table_references' lista las tablas involucradas en la fusión. Su sintaxis se describe en [JOIN](#).

Para la primera sintaxis, sólo se borran las filas coincidentes de las tablas listadas antes de la cláusula *FROM*. Para la segunda, sólo se borran las filas coincidentes de las tablas listadas en la cláusula *FROM* (antes de la cláusula *USING*). El efecto es que se pueden borrar filas de muchas tablas al mismo tiempo y además tener tablas adicionales que se usan para búsquedas:

```
DELETE t1,t2 FROM t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

O

```
DELETE FROM t1,t2 USING t1,t2,t3 WHERE t1.id=t2.id AND t2.id=t3.id
```

Estas sentencias usan los tres ficheros cuando buscan filas para borrar, pero borran las filas coincidentes sólo de las tablas t1 y t2.

Los ejemplos muestran fusiones internas usando el operador coma, pero las sentencias **DELETE** multitabla pueden usar cualquier tipo de fusión (join) permitidas en sentencias [SELECT](#), como *LEFT JOIN*.

El * después de los nombres de tabla aparece sólo por compatibilidad con **Access**:

Si se usa una sentencia **DELETE** multitabla que afecte a tablas **InnoDB** para las que haya definiciones de claves foráneas, el optimizador MySQL procesará las tablas en un orden diferente del de la relación padre/hijo. En ese caso, la sentencia puede fallar y deshará los cambios (roll back). En su lugar, se debe borrar de una sola tabla y confiar en las capacidades de *ON DELETE* que proporciona **InnoDB** que harán que las otras tablas se modifiquen del modo adecuado.

1.3. MODIFICACIÓN DE VALORES DE UNA TABLA.

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
    [ORDER BY ...]
    [LIMIT row_count]
```

O:

```
UPDATE [LOW_PRIORITY] [IGNORE] tbl_name [,tbl_name ...]
    SET col_name1=expr1 [, col_name2=expr2 ...]
    [WHERE where_definition]
```

UPDATE actualiza columnas de filas existentes de una tabla con nuevos valores. La cláusula *SET* indica las columnas a modificar y los valores que deben tomar. La cláusula *WHERE*, si se da, especifica qué filas deben ser actualizadas. Si no se especifica, serán actualizadas todas ellas. Si se especifica la cláusula *ORDER BY*, las filas se modificarán en el orden especificado. La cláusula *LIMIT* establece un límite al número de filas que se pueden actualizar.

La sentencia **UPDATE** soporta los modificadores siguientes:

- Si se usa la palabra **LOW_PRIORITY**, la ejecución de **UPDATE** se retrasará hasta que no haya otros clientes haciendo lecturas de la tabla.

- Si se especifica **IGNORE**, la sentencia **UPDATE** no se abortará si se producen errores durante la actualización. Las filas con conflictos de claves duplicadas no se actualizarán. Las filas para las que la actualización de columnas se puedan producir errores de conversión se actualizarán con los valores válidos más próximos.

Si se accede a una columna de "tbl_name" en una expresión, **UPDATE** usa el valor actual de la columna. Por ejemplo, la siguiente sentencia asigna a la columna "edad" su valor actual más uno:

```
mysql> UPDATE persondata SET edad=edad+1;
```

Las asignaciones **UPDATE** se evalúan de izquierda a derecha. Por ejemplo, las siguientes sentencias doblan el valor de la columna "edad", y después la incrementan:

```
mysql> UPDATE persondata SET edad=edad*2, edad=edad+1;
```

Si se asigna a una columna el valor que tiene actualmente, MySQL lo notifica y no la actualiza.

Si se actualiza una columna que ha sido declarada como *NOT NULL* con el valor *NULL*, se asigna el valor por defecto apropiado para el tipo de la columna y se incrementa en contador de avisos. El valor por defecto es 0 para tipos numéricos, la cadena vacía ("") para tipos de cadena, y el valor "cero" para tipos de fecha y tiempo.

UPDATE devuelve el número de filas que se han modificado. A partir de la versión 3.22 de MySQL, la función de API C [mysql_info](#) devuelve el número de filas que han coincidido y actualizado, y el número de avisos que se han obtenido durante la actualización.

Desde la versión 3.23 de MySQL, se puede usar **LIMIT row_count** para restringir el rango de actualización. La cláusula **LIMIT** trabaja del modo siguiente:

- Antes de MySQL 4.0.13, **LIMIT** restringía el número de filas afectadas. La sentencia se detiene tan pronto como se modifican "row_count" filas que satisfagan la cláusula **WHERE**.
- Desde 4.0.13, **LIMIT** se restringe al número de filas coincidentes. La sentencia se detiene tan pronto como se encuentran "row_count" filas que satisfagan la cláusula **WHERE**, tanto si se han modificado como si no.

Si se usa una cláusula **ORDER BY**, las filas serán actualizadas en el orden especificado. **ORDER BY** está disponible desde MySQL 4.0.0.

Desde la versión 4.0.4 de MySQL, también es posible realizar operaciones **UPDATE** que cubran múltiples tablas:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

El ejemplo muestra una fusión interna usando el operador coma, pero un **UPDATE** multitable puede usar cualquier tipo de fusión (join) permitido en sentencias [SELECT](#), como un **LEFT JOIN**.

Nota: no es posible usar **ORDER BY** o **LIMIT** con **UPDATE** multitable.

1.3.1. MODIFICACIÓN ESPECIAL A TRAVÉS DE CONSULTA.

```
UPDATE nombre_tabla SET
    columna= <consulta>
WHERE <condición>
```

Actualmente, no se puede actualizar una tabla y seleccionar desde la misma en una subconsulta.

Ejemplos:

- Cambiamos la dirección del COD_CENTRO 22 a 'C/Pilón 13' y el número de plazas a 295:

```
UPDATE Centros SET
    DIRECCION = 'C/Pilón 13',
    NUM_plazas = 295           => cuidadito con la última coma
    WHERE cod_centro = 22;
```

- En la tabla centros, igualar la dirección y el número de plazas del código de centro 10 a los valores de las columnas correspondientes que están almacenadas para el código de centro 50

```
UPDATE Centros SET
    (direccion, num_plazas) = (SELECT direccion, num_plazas
        FROM Centros
        WHERE cod_centro = 50)
WHERE cod_centro =10;
```

- Modificar en la tabla EMP todos los empleados que pertenezcan al departamento con mayor número de empleados, incrementando su sueldo en un 15% y poniendo 0 de comisión:

```
UPDATE EMP SET
    sal = sal * 1.15,
    comm = 0
    WHERE deptno = (SELECT deptno
        FROM EMP
        GROUP BY deptno
        HAVING COUNT (*) = (SELECT MAX(COUNT(*))
            FROM EMP
            GROUP BY deptno));
```

- Para los empleados de la tabla EMP del departamento 'ACCOUNTING', ponerles el salario del empleado 'SCOTT' reducido en un 20%.

```
UPDATE EM0 SET
    sal = (SELECT sal*0.8
        FROM EMP
        WHERE ename = 'SCOTT')
    WHERE deptno = (SELECT deptno
        FROM DEPT
        WHERE dname = 'ACCOUNTING'); => Clark, King, Miller
```

1.4. CLAUSULA AUTO_INCREMENT.

```
DROP TABLE IF EXISTS Auto_Inc;  
CREATE TABLE Auto_Inc  
    (COD INT UNIQUE AUTO_INCREMENT,  
     valor CHAR(1));  
INSERT INTO Auto_Inc (valor) VALUES ('A'), ('B');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc VALUES (3,'T');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc VALUES (7,'V');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc (valor) VALUES ('C');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc (valor) VALUES ('D');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc VALUES (7,'X');  
DELETE FROM Auto_Inc  
    WHERE COD=7;  
INSERT INTO Auto_Inc (valor) VALUES ('Y');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc VALUES (7,'Z');  
SELECT * FROM Auto_Inc;  
INSERT INTO Auto_Inc VALUES ('nada','_');  
INSERT INTO Auto_Inc VALUES(NULL,'_');  
SELECT * FROM Auto_Inc;
```

1.5. EJEMPLOS

INserción de Valores en una Tabla

- Inserción de una tupla completa :

```
INSERT INTO Referencia
VALUES ('A1','Pepe','Prieto','Perez')
```

- Inserción haciendo uso del valor por defecto :

```
INSERT INTO Referencia
(CLAVE,NOMBRE,APELLIDO2)
VALUES ('A2','Salustio','Centeno')
```

- Inserción de una tupla dejando una columna a nulo :

```
INSERT INTO Referencia (CLAVE,NOMBRE,APELLIDO1)
VALUES ('A3','Salustio','Centeno');
```

1 fila creada.

```
SQL> SELECT *
  2  FROM Referencia;
CL NOMBRE    APELLIDO1 APELLIDO2
-----+
A2 Salustio  Pelaez   Centeno
A3 Salustio  Centeno
A1 Pepe      Prieto   Perez
```

- Casos erróneos :

SQL>

```
1 INSERT INTO Referencia (CLAVE,NOMBRE,APELLIDO2)
2* VALUES (null,'Salustio','Centeno')
INSERT INTO Referencia (CLAVE,NOMBRE,APELLIDO2)
*
```

ERROR en línea 1:

ORA-01400: falta columna obligatoria (NOT NULL) o NULL durante inserción

***** No se permiten valores nulos en las claves *****

SQL>

```
1 INSERT INTO Referencia (CLAVE,APELLIDO1,APELLIDO2)
2* VALUES ('A4','Ferreiro','Zapatero')
INSERT INTO Referencia (CLAVE,APELLIDO1,APELLIDO2)
*
```

ERROR en línea 1:

ORA-01400: falta columna obligatoria (NOT NULL) o NULL durante inserción

*** No se permiten valores nulos en campos definidos como NOT NULL

SQL>

```

1 INSERT INTO Referencia
(CLAVE,NOMBRE,APELLIDO1,APELLIDO2)
2* VALUES ('A4','Saturnino','Ferreiro','Roldan')
INSERT INTO Referencia
(CLAVE,NOMBRE,APELLIDO1,APELLIDO2)
*
ERROR en línea 1:
ORA-02290: restricción de control (PEDRO.SYS_C00409) violada
*** No se permite el valor Roldan como segundo apellido ***
INSERT INTO Referencia
VALUES ('A2','AVION')
INSERT INTO Referencia
VALUES ('A3','ALADELTA')
```

BORRADO DE VALORES DE UNA TABLA

Tabla PROVEEDORES

CL	NOMBRE	APELLIDO1	APELLIDO2
A2	Salustio	Pelaez	Centeno
A3	Salustio	Centeno	
A1	Pepe	Prieto	Perez

- Borrado de todos los proveedores que se llamen *Pepe*

```

SQL> DELETE
2  FROM Referencia
3  WHERE UPPER(NOMBRE)='PEPE';
1 fila borrada.
```

*** *UPPER* es una función que transforma una cadena a mayúsculas

- Borrado de todos los proveedores que tengan un valor nulo (NULL) como segundo apellido :

```

SQL>
1 DELETE
2  FROM Referencia
3* WHERE APELLIDO2 IS NULL
1 fila borrada.
```

- RESULTADO :

```

SQL> SELECT *
2  FROM Referencia;
CL NOMBRE APELLIDO1 APELLIDO2
-----
A2 Salustio Pelaez Centeno
```

MODIFICACIÓN DE LOS VALORES DE UNA TABLA

- El primer apellido del proveedor 'A2' , no es 'Pelaez', sino 'Paez' :

```
SQL> UPDATE Referencia
2   SET APELLIDO1='Paez'
3   WHERE CLAVE='A2';
```

1 fila actualizada.

```
SQL> SELECT *
2   FROM Referencia;
```

CL	NOMBRE	APELLIDO1	APELLIDO2
A2	Salustio	Paez	Centeno
A1	Pepe	Prieto	Perez
A3	Roque	Martin	Suarez
A4	Roque	Martin	
A5	Tome	Pelaez	Soro
A6	Siloe	Pelaez	Sito

(Para mayor claridad en las operaciones insertamos más valores en la tabla) :

CL	NOMBRE	APELLIDO1	APELLIDO2
A2	Salustio	Paez	Centeno
A1	Pepe	Prieto	Perez
A3	Roque	Martin	Suarez
A4	Roque	Martin	
A5	Tome	Pelaez	Soro
A6	Siloe	Pelaez	Sito

- A todos aquellos proveedores que tengan un valor nulo en el segundo apellido, ponerle el primer apellido del proveedor 'A1' :

```
SQL> UPDATE Referencia
2   SET APELLIDO2=(SELECT APELLIDO1
3   FROM Referencia
4   WHERE CLAVE='A1')
5   WHERE APELLIDO2 IS NULL;
```

1 fila actualizada.

```
SQL> SELECT *
2   FROM Referencia;
```

CL	NOMBRE	APELLIDO1	APELLIDO2
A2	Salustio	Paez	Centeno
A1	Pepe	Prieto	Perez
A3	Roque	Martin	Suarez
A4	Roque	Martin	Prieto
A5	Tome	Pelaez	Soro
A6	Siloe	Pelaez	Sito

- Al proveedor 'A3', ponerle '*Miguel Porlan Chendo*' si existe otro con un primer apellido igual:

```
SQL>
1 UPDATE Referencia R1
2 SET R1.NOMBRE='Miguel',
3     R1.APELLIDO1='Porlan',
4     R1.APELLIDO2='Chendo'
5 WHERE R1.CLAVE='A3' AND EXISTS (SELECT *
6                               FROM REFERENCIA R2
7*                             WHERE R1.APELLIDO1=R2.APELLIDO1)
1 fila actualizada.
```

```
SQL> SELECT *
2  FROM Referencia;
```

CL	NOMBRE	APELLIDO1	APELLIDO2
A2	Salustio	Paez	Centeno
A1	Pepe	Prieto	Perez
A3	Miguel	Porlan	Chendo
A4	Roque	Martin	Prieto
A5	Tome	Pelaez	Soro
A6	Siloe	Pelaez	Sito

2. UTILIZACIÓN DE VARIABLES DE USUARIO

Se pueden emplear variables de usuario de MySQL para retener resultados sin necesidad de almacenarlos en variables del lado del cliente.

Por ejemplo, para encontrar los artículos con el precio más alto y más bajo se puede hacer lo siguiente:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+-----+-----+-----+
| article | dealer | price |
+-----+-----+-----+
| 0003 | D | 1.25 |
| 0004 | D | 19.95 |
+-----+-----+-----+
          SELECT nomp INTO @nombre FROM Pro
          SET @nombre = 'Pepe'
```

3. COMANDOS TRANSACCIONALES Y DE BLOQUEO DE TABLAS

3.1. Sintaxis de START TRANSACTION, COMMIT y ROLLBACK

Por defecto, MySQL se ejecuta con el modo **autocommit activado**. Esto significa que en cuando ejecute un comando que actualice (modifique) una tabla, MySQL almacena la actualización **en disco**.

Si usa tablas transaccionales (como InnoDB o BDB), puede desactivar el modo autocommit con el siguiente comando: **SET AUTOCOMMIT=0;**

Tras deshabilitar el modo autocommit poniendo la variable **AUTOCOMMIT** a cero, debe usar **COMMIT** para almacenar los cambios en disco o **ROLLBACK** si quiere ignorar los cambios hechos desde el comienzo de la transacción.

Si quiere deshabilitar el modo autocommit para una serie única de comandos, puede usar el comando **START TRANSACTION**:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

Con **START TRANSACTION**, autocommit permanece deshabilitado hasta el final de la transacción con **COMMIT** o **ROLLBACK**. El modo autocommit vuelve a su estado previo.

Comenzar una transacción provoca que se realice un **UNLOCK TABLES** implícito.

Cada transacción se almacena en el log binario en un trozo, hasta **COMMIT**. Las transacciones que se deshacen no se loguean. (Excepción: Las modificaciones a tablas no transaccionales no pueden deshacerse. Si una transacción que se deshace incluye modificaciones a tablas no transaccionales, la transacción entera se loguea con un comando **ROLLBACK** al final para asegurar que las modificaciones a estas tablas se replican.)

Deshacer puede ser una operación lenta que puede ocurrir sin que el usuario lo haya pedido explícitamente (por ejemplo, cuando ocurre un error). Debido a ello, **SHOW PROCESSLIST** en MySQL 5.0 muestra **Rolling back** en la columna **State** para la conexión durante rollbacks implícitos y explícitos (comando SQL **ROLLBACK**).

3.2. Sentencias que no se pueden deshacer

Algunos comandos no pueden deshacerse. En general, esto incluye **comandos del lenguaje de definición de datos (DDL)**, tales como los que crean y borran bases de datos, los que crean, borran o alteran tablas o rutinas almacenadas.

3.3. Sentencias que causan una ejecución (commit) implícita

Cada uno de los comandos siguientes (y cualquier sinónimo de los mismos) terminan una transacción implícitamente, como si hubiera realizado un **COMMIT** antes de ejecutar el comando:

ALTER TABLE	BEGIN	CREATE INDEX
CREATE TABLE		CREATE DATABASE
DROP DATABASE	DROP INDEX	DROP TABLE
LOAD MASTER DATA	LOCK TABLES	RENAME TABLE
SET AUTOCOMMIT=1	START TRANSACTION	TRUNCATE TABLE

UNLOCK TABLES también realiza un **commit** de una transacción si hay cualquier tabla bloqueada.

3.4. Sintaxis de SAVEPOINT y ROLLBACK TO SAVEPOINT

SAVEPOINT identifier
ROLLBACK TO SAVEPOINT identifier

En MySQL 5.0, InnoDB soporta los comandos SQL **SAVEPOINT** y **ROLLBACK TO SAVEPOINT**.

El comando **SAVEPOINT** crea un punto dentro de una transacción con un nombre **identifier**.

Si la transacción actual tiene un punto con el mismo nombre, el antiguo se borra y se crea el nuevo.

El comando **ROLLBACK TO SAVEPOINT** deshace una transacción hasta el punto nombrado. Las modificaciones que la transacción actual hace al registro tras el punto se deshacen en el rollback.

Si un comando retorna el siguiente error, significa que no existe ningún punto con el nombre especificado: **ERROR 1181: Got error 153 during ROLLBACK**

Todos los puntos de la transacción actual se borran si ejecuta un **COMMIT**, o un **ROLLBACK** que no nombre ningún punto.

EJEMPLO: Indicar en los recuadros el resultado de cada SELECT.

```

/*TRANSACCIONES*/
DROP TABLE IF EXISTS Tran;
CREATE TABLE Tran
    (valor INT);
SET AUTOCOMMIT=0;
INSERT INTO Tran VALUES (1);
SELECT * FROM Tran;

ROLLBACK;
SELECT * FROM Tran;

INSERT INTO Tran VALUES (2);
SELECT * FROM Tran;

COMMIT;
SELECT * FROM Tran;

INSERT INTO Tran VALUES (3);
SELECT * FROM Tran;

SAVEPOINT UNO;
INSERT INTO Tran VALUES (4);
SELECT * FROM Tran;

ROLLBACK TO SAVEPOINT UNO;
SELECT * FROM Tran;

ROLLBACK;
SELECT * FROM Tran;

```

3.5. BLOQUEO de Tablas

3.5.1. Sintaxis de LOCK TABLES

LOCK TABLES

```
tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}
[,tbl_name [AS alias] {READ [LOCAL] | [LOW_PRIORITY] WRITE}] ...
```

LOCK TABLES bloquea tablas para el flujo actual. Si alguna de las tablas la bloquea otro flujo, bloquea hasta que pueden adquirirse todos los bloqueos.

Un bloqueo de tabla protege sólo contra lecturas inapropiadas o escrituras de otros clientes. El cliente que tenga el bloqueo, incluso un bloqueo de lectura, puede realizar operaciones a nivel de tabla tales como **DROP TABLE**.

LOCK TABLES no es una operación transaccional y hace un commit implícito de cualquier transacción activa antes de tratar de bloquear las tablas.

Para usar **LOCK TABLES** en MySQL 5.0, debe tener el permiso **LOCK TABLES** y el permiso **SELECT** para las tablas involucradas.

WRITE bloquea normalmente teniendo una prioridad superior que **READ** al bloquear para asegurar que las actualizaciones se procesan en cuanto se puede. Esto significa que si un flujo obtiene un bloqueo **READ** y luego otro flujo pide un bloqueo **WRITE**, las peticiones de bloqueo **READ** posteriores esperan hasta que el flujo **WRITE** quita el bloqueo.

Reglas para la Adquisición de bloqueos

READ:

- La sesión que mantiene el bloqueo puede leer la tabla (pero no escribirlo).
- Múltiples sesiones pueden adquirir un bloqueo **READ** para una tabla al mismo tiempo.
- Otras sesiones pueden leer la tabla sin adquirir explícitamente una **READ** bloqueo.

WRITE:

- La sesión que mantiene el bloqueo puede leer y escribir la tabla.
- Sólo la sesión que mantiene el bloqueo puede acceder a la tabla. Ninguna otra sesión puede acceder a él hasta que se libere el bloqueo.
- Solicitudes de actividad de otras sesiones quedan bloqueadas mientras que el **WRITE** bloqueo se mantiene.

Una declaración de bloqueo de tablas debe esperar debido a bloqueos mantenidos por otras sesiones en cualquiera de las tablas, ya que se bloquea hasta que todos los bloqueos pueden ser adquiridos.

Una sesión que requiere bloqueos debe adquirir todos los bloqueos que necesita en una sola declaración **LOCK TABLES**. Mientras que las cerraduras así obtenidos se llevan a cabo, la sesión sólo puede acceder a las tablas bloqueadas. Por ejemplo, en la siguiente secuencia de instrucciones, se produce un error durante el intento de acceso t2, ya que no estaba cerrada con llave en la declaración **LOCK TABLES**:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+-----+
| COUNT (*) |
+-----+
| 3 |
+-----+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Tabla 't2' no estaba cerrada con LOCK TABLES
```

3.5.1. Sintaxis de UNLOCK TABLES

UNLOCK TABLES

UNLOCK TABLES libera cualquier bloqueo realizado por el flujo actual. Todas las tablas bloqueadas por el flujo actual se liberan implícitamente cuando el flujo realiza otro **LOCK TABLES**, o cuando la conexión con el servidor se cierra.

Reglas para la liberación de bloqueos

Cuando los bloqueos de tabla en poder de una sesión son liberados, todos ellos son liberados al mismo tiempo. Una sesión puede liberar sus bloqueos de forma explícita, o puede ser lanzado de forma implícita en ciertas condiciones.

- Una sesión puede liberar sus bloqueos explícitamente **UNLOCK TABLES**.
- Si una sesión emite una **LOCK TABLES**, sus cerraduras existentes se liberan implícitamente antes de conceder las nuevas.
- Si una sesión inicia una transacción (por ejemplo, con **START TRANSACTION**), entonces se realiza un **UNLOCK TABLES**.
- Si la conexión para una sesión de cliente termina, ya sea normal o anormal, el servidor libera de forma implícita todos los bloqueos de tabla en poder de la sesión (transaccionales y no transaccionales). Si el cliente vuelve a conectarse, los bloqueos ya no estarán en vigor.

Puede usar **KILL** para terminar un flujo que está esperando para un bloqueo de tabla.

Nota: Si usa **ALTER TABLE** en una tabla bloqueada, puede desbloquearse.

3.6. Sintaxis de SET TRANSACTION

**SET [GLOBAL | SESSION] TRANSACTION ISOLATION LEVEL
{ READ UNCOMMITTED | READ COMMITTED | REPEATABLE READ | SERIALIZABLE }**

Este comando prepara el nivel de aislamiento de transacción para la siguiente transacción, globalmente, o para la sesión actual.

El comportamiento por defecto de **SET TRANSACTION** es poner el nivel de aislamiento para la siguiente transacción (que no ha comenzado todavía). Si usa la palabra clave **GLOBAL** el comando pone el nivel de aislamiento de transacción por defecto globalmente para todas las transacciones creadas desde ese momento. Las conexiones existentes no se ven afectadas. Necesita el permiso **SUPER** para hacerlo. Usar la palabra clave **SESSION** determina el nivel de transacción para todas las transacciones futuras realizadas en la conexión actual.

4. Las VISTAS

Una vista es una tabla virtual formada a través de una consulta a partir de otras tablas o vistas existentes en las bases de datos. Suelen usarse para definir esquemas externos o subesquemas.

Si se especifica una lista de nombres de columnas, tiene que tener el mismo número de elementos que el número de columnas producidas por la consulta. Si no se especifican, se toman los nombres de las columnas que se especifican en las columnas.

Es obligatorio poner nombre de columnas cuando se producen datos calculados.

4.1. Sintaxis de CREATE VIEW.

```
CREATE [OR REPLACE] [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW nombre_vista [(columnas)]
AS sentencia_select
[WITH [CASCDED | LOCAL] CHECK OPTION]
```

Esta sentencia crea una vista nueva o reemplaza una existente si se incluye la cláusula **OR REPLACE**. La *sentencia_select* es una sentencia SELECT que proporciona la definición de la vista. Puede estar dirigida a tablas de la base o a otras vistas.

Se requiere que posea el permiso CREATE VIEW para la vista, y algún privilegio en cada columna seleccionada por la sentencia SELECT. Para columnas incluidas en otra parte de la sentencia SELECT debe poseer el privilegio SELECT. Si está presente la cláusula OR REPLACE, también deberá tenerse el privilegio DELETE para la vista.

Toda vista pertenece a una base de datos. Por defecto, las vistas se crean en la base de datos actual. Para crear una vista en una base de datos específica, indíquela con *base_de_datos.nombre_vista* al momento de crearla.

```
mysql> CREATE VIEW test.v AS SELECT * FROM t;
```

Las tablas y las vistas comparten el mismo espacio de nombres en la base de datos, por eso, una base de datos no puede contener una tabla y una vista con el mismo nombre.

Al igual que las tablas, las vistas no pueden tener nombres de columnas duplicados. Por defecto, los nombres de las columnas devueltos por la sentencia SELECT se usan para las columnas de la vista. La cantidad de nombres indicados en *columnas* debe ser igual a la cantidad de columnas devueltas por la sentencia SELECT.

Las columnas devueltas por la sentencia SELECT pueden ser simples referencias a columnas de la tabla, pero tambien pueden ser expresiones conteniendo funciones, constantes, operadores, etc.

Los nombres de tablas o vistas sin calificar en la sentencia SELECT se interpretan como pertenecientes a la base de datos actual. Una vista puede hacer referencia a tablas o vistas en otras bases de datos precediendo el nombre de la tabla o vista con el nombre de la base de datos apropiada.

Las vistas pueden crearse a partir de varios tipos de sentencias SELECT. Pueden hacer referencia a tablas o a otras vistas. Pueden usar combinaciones, UNION, y subconsultas. En el siguiente ejemplo se define una vista que selecciona dos columnas de otra tabla, así como una expresión calculada a partir de ellas:

```
CREATE TABLE t (qty INT, price INT);
INSERT INTO t VALUES(3, 50);
CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
SELECT * FROM v;
```

La definición de una vista está sujeta a las siguientes limitaciones:

- La sentencia SELECT no puede contener una subconsulta en su cláusula FROM.
- La sentencia SELECT no puede hacer referencia a variables del sistema o del usuario.
- La sentencia SELECT no puede hacer referencia a parámetros de sentencia preparados.
- Dentro de una rutina almacenada, la definición no puede hacer referencia a parámetros de la rutina o a variables locales.
- La definición no puede hacer referencia a una tabla TEMPORARY, y tampoco se puede crear una vista TEMPORARY.
- Las tablas mencionadas en la definición de la vista deben existir siempre.
- No se puede asociar un disparador con una vista.

En la definición de una vista está **permitido** ORDER BY, pero es ignorado si se seleccionan columnas de una vista que tiene su propio ORDER BY.

4.1.1. La cláusula WITH CHECK OPTION.

Se puede utilizarse en una vista actualizable para evitar inserciones o actualizaciones excepto en los registros en que la cláusula WHERE de la sentencia_select se evalúe como true.

EJEMPLOS:

```
DROP DATABASE IF EXISTS Vistas;
CREATE DATABASE Vistas;
USE Vistas;

CREATE TABLE Alumno(
    DniA CHAR(9) PRIMARY KEY,
    NombreA VARCHAR(20),
    Localidad VARCHAR(20) NOT NULL);

INSERT INTO Alumno (DniA, NombreA, Localidad)
    VALUES ('A0', 'Felipin', 'Madrid'),
            ('A1', 'Joselín', 'Madrid'),
            ('A2', 'Marianin', 'Sevilla'),
            ('A3', 'Felisin', 'Zamora');

SELECT * FROM Alumno;
```

/ *** (1A1)- SIN Clausula Y CON Atributo_Localidad *** */*

```
DROP VIEW IF EXISTS VALu1A1;
CREATE VIEW VALu1A1(DniA, NombreA, Localidad)
    AS SELECT DniA, NombreA, Localidad
        FROM Alumno
        WHERE Localidad='Madrid';

SET AUTOCOMMIT=0;

INSERT INTO VALu1A1 (DniA, NombreA, Localidad)
    VALUES ('A4', 'Javier', 'Madrid'),
            ('A5', 'Manuel', 'Zamora'),
            ('A6', 'Antonio', 'Madrid'),
            ('A7', 'Francisco', 'Zamora');

SELECT * FROM Alumno;
SELECT * FROM VALu1A1;
ROLLBACK;
```

```
SELECT * FROM Alumno;
UPDATE VAlu1A1
    SET Localidad='Madrid'
    WHERE DniA='A2';
UPDATE VAlu1A1
    SET Localidad='Leon'
    WHERE DniA='A0';
UPDATE VAlu1A1
    SET Localidad='Madrid'
    WHERE DniA='A3';
UPDATE VAlu1A1
    SET Localidad='Salamanca'
    WHERE DniA='A1';
SELECT * FROM Alumno;
SELECT * FROM VAlu1A1;
ROLLBACK;

SELECT * FROM Alumno;
UPDATE VAlu1A1
    SET Localidad='XXXXX';
SELECT * FROM Alumno;
SELECT * FROM VAlu1A1;
ROLLBACK;

/*DELETE FROM VAlu1A1
WHERE ... */
```

/* * (1_B1)- CON Clausula Y CON Atributo_localidad *** */**

```
DROP VIEW IF EXISTS VAlu1B1;
CREATE VIEW VAlu1B1 (DniA, NombreA, Localidad)
AS SELECT DniA, NombreA, Localidad
        FROM Alumno
        WHERE Localidad='Madrid'
        WITH CHECK OPTION;
SET AUTOCOMMIT=0;
INSERT INTO VAlu1B1 (DniA, NombreA, Localidad)
    VALUES ('A4', 'Javier', 'Madrid');
INSERT INTO VAlu1B1 (DniA, NombreA, Localidad)
    VALUES ('A5', 'Manuel', 'Zamora');
INSERT INTO VAlu1B1 (DniA, NombreA, Localidad)
    VALUES ('A6', 'Antonio', 'Madrid');
INSERT INTO VAlu1B1 (DniA, NombreA, Localidad)
    VALUES ('A7', 'Francisco', 'Zamora');
SELECT * FROM Alumno;
SELECT * FROM VAlu1B1;
ROLLBACK;

SELECT * FROM Alumno;
UPDATE VAlu1B1
    SET Localidad='Madrid'
    WHERE DniA='A2';
UPDATE VAlu1B1
    SET Localidad='Leon'
    WHERE DniA='A0';
UPDATE VAlu1B1
```

```
SET Localidad='Madrid'  
WHERE DniA='A4';  
UPDATE VALu1B1  
    SET Localidad='Salamanca'  
    WHERE DniA='A4';  
SELECT * FROM Alumno;  
SELECT * FROM VALu1B1;  
ROLLBACK;  
  
SELECT * FROM Alumno;  
UPDATE VALu1B1  
    SET Localidad='XXXXX';  
SELECT * FROM Alumno;  
SELECT * FROM VALu1B1;  
ROLLBACK;
```

/* * (1_A2)- SIN Clausula Y SIN Atributo_localidad *** */**

```
DROP VIEW IF EXISTS VALu1A2;  
CREATE VIEW VALu1A2(DniA, NombreA)  
AS SELECT DniA, NombreA  
    FROM Alumno  
    WHERE Localidad='Madrid';  
SET AUTOCOMMIT=0;  
  
INSERT INTO VALu1A2 (DniA, NombreA)  
    VALUES ('A4', 'Javier');  
SELECT * FROM Alumno;  
SELECT * FROM VALu1A2;  
ROLLBACK;
```

/* * (1_B2)- CON Clausula Y SIN Atributo_localidad *** */**

```
DROP VIEW IF EXISTS VALu1B2;  
CREATE VIEW VALu1B2(DniA, NombreA)  
AS SELECT DniA, NombreA  
    FROM Alumno  
    WHERE Localidad='Madrid'  
    WITH CHECK OPTION;  
  
SET AUTOCOMMIT=0;  
  
INSERT INTO VALu1B2 (DniA, NombreA)  
    VALUES ('A4', 'Javier');  
SELECT * FROM Alumno;  
SELECT * FROM VALu1A2;  
ROLLBACK;
```

En la cláusula WITH CHECK OPTION de una vista actualizable, las palabras reservadas **LOCAL** y **CASCDED** determinan el alcance de la verificación cuando la vista está definida en términos de otras vistas. LOCAL restringe el CHECK OPTION sólo a la vista que está siendo definida. CASCDED provoca que las vistas subyacentes también sean verificadas. Si no se indica, el valor por defecto es CASCDED. Considere las siguientes definiciones de tabla y vistas:

```
mysql> CREATE TABLE t1 (a INT);
mysql> CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
-> WITH CHECK OPTION;
mysql> CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
-> WITH LOCAL CHECK OPTION;
mysql> CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
-> WITH CASCDED CHECK OPTION;
```

Las vistas v2 y v3 están definidas en términos de otra vista, v1. v2 emplea *check option* LOCAL, por lo que las inserciones sólo atraviesan la verificación de v2 . v3 emplea *check option* CASCDED de modo que las inserciones no solamente atraviesan su propia verificación sino también las de las vistas subyacentes. Las siguientes sentencias demuestran las diferencias:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

4.2. Sintaxis de ALTER VIEW.

```
ALTER [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
VIEW nombre_vista [(columnas)]
AS sentencia_select
[WITH [CASCDED | LOCAL] CHECK OPTION]
```

Esta sentencia modifica la definición de una vista existente. La sintaxis es semejante a la empleada en CREATE VIEW. Se requiere que posea los **permisos** CREATE VIEW y DELETE para la vista, y algún privilegio en cada columna seleccionada por la sentencia SELECT.

4.3. Sintaxis de DROP VIEW.

```
DROP VIEW [IF EXISTS] nombre_vista [, nombre_vista] ...
```

DROP VIEW elimina una o más vistas de la base de datos. Se debe poseer el **privilegio** DROP en cada vista a eliminar. La cláusula **IF EXISTS** se emplea para evitar que ocurra un error por intentar eliminar una vista inexistente.

4.4. Sintaxis de SHOW CREATE VIEW.

```
SHOW CREATE VIEW nombre_vista
```

Muestra la sentencia CREATE VIEW que se utilizó para crear la vista.

```
mysql> SHOW CREATE VIEW v;
```

Table	Create Table
v	CREATE VIEW `test`.`v` AS select 1 AS `a`,2 AS `b`

4.5. TIPOS de VISTAS.

1. Vista vertical: cuando la selección se hace tomando columnas.

Ejemplo: **CREATE VIEW Lupa**
AS SELECT Dnombre
FROM Dept

2. Vista horizontal: cuando se seleccionan una serie de atributos.

Ejemplo: **CREATE VIEW Lupah**
AS SELECT *
FROM Dept
WHERE Loc <> 'MADRID'

3. Vista vertical-horizontal: se cogen ambas.

Ejemplo: **CREATE VIEW Lupahv**
AS SELECT Loc
FROM dept
WHERE loc <> 'MADRID'

4.6. OPERACIONES sobre VISTAS.

- Consulta. Ej. **SELECT * FROM TAB**
- Actualización: La información puede ser actualizada en las vistas directamente o a través de las tablas sobre las que se defina. Existen restricciones:

1.- Para el borrado de filas de una tabla a través de una vista, la vista se debe crear:

- a) Con filas de una sola tabla.
- b) Sin utilizar cláusulas GROUP BY y DISTINCT.
- c) Sin utilizar las funciones de grupo (MAX, MIN...).

2.- Para las modificaciones las vistas se definen según las restricciones anteriores y además ninguna de las columnas a actualizar debe haber sido definida como una expresión. Ej.

SELECT (CANT*PREC) – RESTO

3. –Para las inserciones todas las restricciones anteriores y además todas las tuplas obligatoriamente de la tabla deben estar en la lista.

5. SEGURIDAD en Oracle

5.1. USUARIOS

La seguridad de la BBDD se puede clasificar en dos categorías:

- **Seguridad del sistema:** controlar el acceso y uso de la BBDD a nivel del sistema (comprobar si un usuario está autorizado para acceder a la BBDD)
- **Seguridad de los datos:** controlar el acceso y uso de la BBDD a nivel de objetos (cada vez que un usuario accede a un objeto como tabla, vista, etc, se comprueba si el usuario puede o no acceder al mismo y qué tipo de operación puede hacer con él → SELECT, INSERT)

Un **usuario** es un nombre definido en la BBDD que se puede conectar a ella y acceder a determinados objetos según ciertas condiciones que define el administrador.

Asociado con **cada usuario** hay un **esquema** => colección lógica de objetos (tablas, vistas, secuencias, sinónimos, índices, clusters, procedures, funciones, paquetes y database links). Por defecto tiene acceso a todos los objetos de su esquema.

5.1.1. CREACIÓN DE USUARIOS

Al instalar la BBDD se crean automáticamente dos usuarios con el privilegio administrador de la BBDD (DBA)

El **usuario SYS** es el propietario de las tablas del diccionario de datos, donde se almacena información del resto de las estructuras de la BBDD. Ningún usuario puede modificar las tablas de SYS (ni siquiera los administradores)

El **usuario SYSTEM** es creado por Oracle para realizar las tareas de administración de la BBDD. Para crear otros usuarios es preciso conectarse como usuario SYSTEM (posee el correspondiente privilegio). Lo normal suele ser que el Administrador de la BBDD se cree un usuario para sí mismo con derechos de administrador y realizar todas las tareas de administración con este nombre de usuario.

Para crear otros usuarios se necesita el privilegio CREATE USER. La sintaxis es:

```
CREATE USER Nombre_usuario
IDENTIFIED BY clave_acceso
[DEFAULT TABLESPACE espacio_tabla]
[TEMPORARY TABLESPACE espacio_tabla]
[QUOTA entero K|M ON espacio_tabla]
    UNLIMITED
[PROFILE perfil];
```

DEFAULT TABLESPACE: asigna a un usuario el tablespace donde almacenar los objetos que cree. Si no se asigna ninguno el tablespace por defecto será SYSTEM (Un tablespace es una unidad lógica de almacenamiento de datos representada físicamente por uno o más archivos de datos. Contiene los objetos creados por los usuarios).

TEMPORARY TABLESPACE: especifica el nombre de tablespace para trabajos temporales. Si no se especifica ninguno por defecto será SYSTEM.

QUOTA: asigna un espacio en megabytes o kilobytes en el tablespace asignado. Si no se especifica esta cláusula, el usuario no tiene cuota asignada y no podrá crear objetos en el tablespace.

PROFILE: asigna un perfil al usuario. Limita el número de sesiones concurrentes de usuario, el tiempo de uso de la CPU, el tiempo de sesión, desconecta al usuario si sobrepasa el tiempo, etc.

Por ejemplo:

```
CREATE USER FERNANDO  
  IDENTIFIED BY FERNANDO  
  DEFAULT TABLESPACE TRABAJO → Es distinta de la de abajo que es  
    temp.  
  QUOTA 500K ON TRABAJO  
  TEMPORARY TABLESPACE TRABAJO; => debemos conectarnos con usuario  
    SYSTEM y tener TRABAJO ya creado
```

Para obtener información sobre todos los usuarios creados en la BBDD se dispone de las vistas ALL_USERS y DBA_USERS (como SYSTEM). Hacer un DESC de las dos vistas.

```
SELECT *  
  FROM ALL_USERS;  
SELECT USERNAME, DEFAULT_TABLESPACE  
  FROM DBA_USERS;
```

5.1.2. MODIFICACIÓN DE USUARIOS

Las opciones dadas a un usuario en la orden CREATE USER se pueden modificar con la orden ALTER USER. Es posible cambiar la clave de acceso, el tablespace por defecto, el tablespace temporal, la cuota en el tablespace o el perfil

```
ALTER USER Nombre_usuario  
  IDENTIFIED BY clave_acceso  
  [DEFAULT TABLESPACE espacio_tabla]  
  [TEMPORARY TABLESPACE espacio_tabla]  
  [QUOTA entero K|M ON espacio_tabla]  
    UNLIMITED  
  [PROFILE perfil];
```

Los parámetros significan lo mismo que en la orden CREATE. Se necesita el privilegio ALTER USER para cambiar otros parámetros que no sean la clave de acceso (cada usuario puede cambiar su propia clave de acceso)

5.1.3. BORRADO DE USUARIOS

Podemos borrar un usuario de la BBDD, incluidos los objetos que contiene

DROP USER usuario [CASCADE];

La opción CASCADE suprime todos los objetos del usuario antes de borrar el usuario.

Nos conectamos como SYSTEM y vemos si JUAN tiene tablas:

```
SELECT OWNER, TABLE_NAME
  FROM DBA_TABLES
 WHERE OWNER = 'JUAN';
```

OWNER	TABLE_NAME
JUAN	DEPT30

Intentamos borrar el usuario sin usar CASCADE

DROP USER JUAN; => No deja con el siguiente error "se debe especificar CASCADE para borrar JUAN"

Será necesario un **DROP USER JUAN CASCADE;** => borra el usuario y sus tablas

5.2. PRIVILEGIOS

Un **privilegio** es la capacidad de un usuario dentro de la BBDD a realizar determinadas operaciones o a acceder a determinados objetos de otros usuarios.

PRIVILEGIO DEL SISTEMA	OPERACIONES AUTORIZADAS
AUDIT	
AUDIT ANY	Auditar un objeto de la base de datos.
CLUSTER	
CREATE CLUSTER	Crear un cluster en el propio esquema.
CREATE ANY CLUSTER	Crear un cluster en cualquier esquema.
ALTER ANY CLUSTER	Modificar cualquier cluster de la base de datos
DROP ANY CLUSTER	Borrar cualquier cluster de la base de datos.
DATABASE	
ALTER DATABASE	Modificar la base de datos, añadiéndole ficheros.
CREATE DATABASE LINK	Crear links privados para acceder a otra base.
CREATE PUBLIC DATABASE LINK	Crear links públicos para acceder a otra base de datos.
INDEX	
CREATE ANY INDEX	Crear un índice en cualquier esquema en cualquier tabla.
ALTER ANY INDEX	Modificar cualquier índice de la base de datos.
DROP ANY INDEX	Borrar cualquier índice de la base de datos.
GRANT ANY PRIVILEGE	Conceder cualquier privilegio de sistema.
PROCEDURE	
CREATE ANY PROCEDURE	Crear procedimientos almacenados, funciones y paquetes en cualquier esquema.
CREATE PROCEDURE	Crear procedimientos almacenados, funciones y paquetes en nuestro esquema.
ALTER ANY PROCEDURE	Modificar procedimientos almacenados, funciones y paquetes en cualquier esquema.
DROP ANY PROCEDURE	Borrar procedimientos almacenados, funciones y paquetes en cualquier esquema.
EXECUTE ANY PROCEDURE	Ejecutar procedimientos, funciones o referencias a paquetes públicos en cualquier esquema.
PROFILE	
CREATE PROFILE	Crear un perfil de usuario.
ALTER PROFILE	Modificar cualquier perfil.
DROP PROFILE	Borrar cualquier perfil.

PRIVILEGIO DEL SISTEMA	OPERACIONES AUTORIZADAS
ROLE	
CREATE ROLE	Crear roles.
ALTER ANY ROLE	Modificar roles.
DROP ANY ROLE	Borrar cualquier rol.
GRANT ANY ROLE	Dar permisos para cualquier rol de la base.
ROLLBACK_SEGMENT	
CREATE ROLLBACK SEGMENT	Crear segmentos de rollback.
ALTER ROLLBACK SEGMENT	Modificar segmentos de rollback.
DROP ROLLBACK SEGMENT	Eliminar segmentos de rollback.
SEQUENCE	
CREATE SEQUENCE	Crear secuencias en nuestro esquema.
ALTER ANY SEQUENCE	Modificar cualquier secuencia de la base.
DROP ANY SEQUENCE	Borrar secuencias de cualquier esquema.
SELECT ANY SEQUENCE	Referenciar secuencias de cualquier esquema.
SESSION	
CREATE SESSION	Conectarnos a la base de datos.
ALTER SESSION	Manejar la orden ALTER SESSION.
RESTRICTED SESSION	Conectarnos a la base de datos cuando se ha levantado con STARTUP RESTRICT.
SYNONYM	
CREATE SYNONYM	Crear sinónimos en nuestro esquema.
CREATE PUBLIC SYNONYM	Crear sinónimos públicos.
DROP PUBLIC SYNONYM	Borrar sinónimos públicos.
CREATE ANY SYNONYM	Crear sinónimos en cualquier esquema.
DROP ANY SYNONYM	Borrar sinónimos de cualquier esquema.
TABLE	
CREATE TABLE	Crear tablas en nuestro esquema y generar índices sobre las tablas del esquema.
CREATE ANY TABLE	Crear una tabla en cualquier esquema.
ALTER ANY TABLE	Modificar una tabla en cualquier esquema.
DROP ANY TABLE	Borrar una tabla en cualquier esquema.
LOCK ANY TABLE	Bloquear una tabla en cualquier esquema.
SELECT ANY TABLE	Hacer SELECT en cualquier tabla.
INSERT ANY TABLE	Insertar filas en cualquier tabla.
UPDATE ANY TABLE	Modificar filas en cualquier tabla.
DELETE ANY TABLE	Borrar filas de cualquier tabla.
TABLESPACES	
CREATE TABLESPACE	Crear espacios de tablas.
ALTER TABLESPACE	Modificar tablespaces.
MANAGE TABLESPACES	Poner on-line u off-line a cualquier tabla
DROP TABLESPACE	Eliminar tablespaces.
UNLIMITED TABLESPACE	Utilizar cualquier espacio de cualquier tabla
TRIGGER	
CREATE ANY TRIGGER	Crear triggers en cualquier esquema de la base
ALTER ANY TRIGGER	Activar o desactivar cualquier trigger.
DROP ANY TRIGGER	Eliminar triggers de cualquier esquema.
CREATE TRIGGER	Crear triggers en nuestro esquema.
USER	
CREATE USER	Crear usuarios y crear cuotas sobre cualquier espacio de tablas, establecer espacios de tablas por omisión y temporales.
ALTER USER	Modificar cualquier usuario. Este privilegio autoriza al que lo recibe a cambiar la contraseña de otro usuario, a cambiar cuotas cualquier espacio de tablas, a establecer espacios de tablas por omisión, etc.
DROP USER	Eliminar usuarios.
VIEW	
CREATE VIEW	Crear vistas en el esquema propio.
CREATE ANY VIEW	Crear vistas en cualquier esquema.
DROP ANY VIEW	Borrar vistas en cualquier esquema.

Cuando se crea un usuario es necesario darle privilegios para que pueda hacer algo. Para ello, Oracle ofrece varios roles o funciones:

Roles (funciones)	Privilegios
CONNECT	ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE y CREATE VIEW
RESOURCE	CREATE CLUSTER, CREATE PROCEDURE, CREATE TABLE, CREATE SEQUENCE y CREATE TRIGGER
DBA	Posee todos los privilegios del sistema
EXP_FULL_DATABASE	Derechos para exportar la BBDD completa
IMP_FULL_DATABASE	Derechos para importar una BBDD

Hay dos tipos de privilegios que podemos definir en la BBDD: privilegios sobre los objetos y privilegios del sistema

5.2.1. PRIVILEGIOS SOBRE LOS OBJETOS

Las sentencias SQL que permite cada privilegio son:

Privilegio	Sentencias SQL permitidas con cada privilegio
ALTER	ALTER objeto (tabla o secuencia)
DELETE	DELETE FROM objeto (tabla o vista)
EXECUTE	EXECUTE objeto (procedimiento)
INDEX	CREATE INDEX ON objeto (tablas)
INSERT	INSERT INTO objeto (tabla o vista)
REFERENCES	CREATE o ALTER TABLE definiendo una restricción de integridad de clave ajena sobre el objeto (sólo tablas)
SELECT	SELECT ... FROM objeto (tabla o vista) Sentencia SQL utilizando un generador de secuencias
UPDATE	UPDATE objeto (tabla o vista)

Nos permiten acceder y realizar cambios en los datos u objetos de otros usuarios (por ejemplo consultar la tabla de otro usuario). Tenemos los siguientes privilegios sobre los objetos tablas, vistas, secuencias y procedures:

Privilegio sobre los objetos	Tabla	Vista	Secuencia	Procedure
ALTER	X		X	
DELETE	X	X		
EXECUTE				X
INDEX	X			
INSERT	X	X		
REFERENCES	X			
SELECT	X	X	X	
UPDATE	X	X		

La orden para dar privilegios sobre los objetos es GRANT, con el siguiente formato:

GRANT privilegio1, [privilegio2 [(col1, col2)]]

ALL PRIVILEGES

ON [usuario.]objeto => objeto sobre el que se dan los privilegios

TO usuario => usuarios o roles a los que se conceden los privilegios

rol

PUBLIC => asigna los privilegios a todos los usuarios actuales y futuros.

Es para garantizar el acceso a determinados objetos a todos los usuarios de la BBDD

[WITH GRANT OPTION]; => el receptor del privilegio o rol pueda "pasarlo" a otros usuarios o roles

5.2.2. PRIVILEGIOS DEL SISTEMA

Son los que dan derecho a ejecutar un tipo de comando SQL o a realizar alguna acción sobre objetos de un tipo especificado (el privilegio para crear tablespaces). Existen unos 80 tipos de privilegios distintos disponibles. Algunos de ellos están en el fichero: **Privilegios_sistema.rtf**

El formato de la orden GRANT para asignar privilegios del sistema es:

GRANT privilegio

rol

ALL PRIVILEGES

TO usuario => identifica a los usuarios o roles a los que se conceden rol privilegios

(rol)

PUBLIC

[WITH ADMIN OPTION]; => permite que el receptor del privilegio o rol pueda conceder esos mismos privilegios a otros usuarios o roles

Cuando creamos un usuario tenemos que darle privilegios para que, como mínimo, pueda iniciar sesión en la BBDD:

CONNECT SYSTEM/MANAGER;

CREATE USER MILAGROS

IDENTIFIED BY MILAGROS

DEFAULT TABLESPACE SYSTEM;

GRANT CREATE SESSION

TO MILAGROS;

CONNECT MILAGROS/MILAGROS;

* Se concede a MILAGROS el rol CONNECT, lo que le permitirá tener los privilegios (ALTER SESSION, CREATE CLUSTER, CREATE DATABASE LINK, CREATE SEQUENCE, CREATE SESSION, CREATE SYNONYM, CREATE TABLE y VIEW)

CONNECT SYSTEM/MANAGER;

GRANT CONNECT

TO MILAGROS;

* Ahora concedemos a PEDRO y JUAN el privilegio de administrador (DBA):

GRANT DBA

TO PEDRO, JUAN;

* Para hacer que MILAGROS pueda borrar usuarios y, además, pueda conceder este privilegio a otros usuarios:

```
GRANT DROP USER  
TO MILAGROS  
WITH ADMIN OPTION;
```

* Para hacer que todos los usuarios puedan hacer SELECT en cualquier tabla de cualquier usuario:

```
GRANT SELECT ANY TABLE  
TO PUBLIC;
```

5.2.3. RETIRADA DE PRIVILEGIOS

Al igual que se conceden privilegios, se pueden retirar mediante la orden REVOKE:

El formato para retirar privilegios de objetos a los usuarios es:

```
REVOKE privilegio_objeto, privilegio_objeto  
ALL PRIVILEGES  
ON [usuario.]objeto  
FROM usuario  
rol  
PUBLIC;
```

Y el formato para retirar privilegios de sistema:

```
REVOKE priv_sistema  
rol  
ALL PRIVILEGES  
FROM usuario  
rol  
PUBLIC;
```

Ejemplos:

* JUAN retira los privilegios SELECT y UPDATE sobre DEPT30 a PEDRO:

```
CONNECT JUAN/JUAN;  
REVOKE SELECT, UPDATE  
ON DEPT30  
FROM PEDRO; => "revocación terminada"
```

REVOKE ALL PRIVILEGES elimina todos los privilegios concedidos anteriormente sobre algún objeto. La opción WITH GRANT OPTION desaparece con el privilegio con el cual fue asignada.

* Quitar todos los privilegios sobre DEPT30 a SCOTT y PEDRO:

```
REVOKE ALL PRIVILEGES  
ON DEPT30  
FROM SCOTT, PEDRO;
```

* Retirar el privilegio de sistema de borrar usuarios a MILAGROS:

```
CONNECT SYSTEM/MANAGER;  
REVOKE DROP USER  
FROM MILAGROS; (no teclear)
```

* Retirar el privilegio de administrador (DBA) a los usuarios JUAN y PEDRO;

```
REVOKE DBA  
FROM JUAN, PEDRO; (no teclear)
```

5.2.4. VISTAS CON INFORMACIÓN DE PRIVILEGIOS

Para conocer los privilegios que han concedido o recibido los usuarios sobre los objetos o a nivel de sistema, podemos consultar las siguientes vistas del diccionario de datos:

SESSION_PRIVS: privilegios del usuario activo

SELECT * FROM SESSION_PRIVS;

USER_SYS_PRIVS: privilegios de sistema asignados al usuario

DBA_SYS_PRIVS: privilegios de sistema asignados a los usuarios o a los roles

USER_TAB_PRIVS: concesiones sobre objetos que son propiedad del usuario, concedidos o recibidos por éste

USER_TAB_PRIVS_MADE: concesiones sobre objetos que son propiedad del usuario (asignadas)

USER_TAB_PRIVS_REC'D: concesiones sobre objetos que recibe el usuario

USER_COL_PRIVS: concesiones sobre columnas en las que el usuario es el propietario, asigna el privilegio o lo recibe.

USER_COL_PRIVS_MADE: todas las concesiones sobre columnas de objetos que son propiedad del usuario

USER_COL_PRIVS_REC'D: concesiones sobre columnas recibidas por el usuario

5.3. ROLES

Un **rol** o **función** es un conjunto de privilegios que recibe un nombre común para facilitar la tarea de asignación de éstos a los usuarios o a otros roles. Los privilegios de un rol pueden ser de sistema y a nivel de objeto.

Supongamos que un conjunto de usuarios del departamento de contabilidad requiere el mismo conjunto de privilegios para trabajar con ciertos datos. Este conjunto de privilegios se puede agrupar en un rol, de tal manera que es posible asignar el mismo rol a cada uno de los usuarios de forma más rápida que ir asignando los privilegios de uno en uno a cada usuario.

CREATE ROLE NombreRol [IDENTIFIED BY contraseña];

Si se usa contraseña, en cada sesión: **SET ROLE NombreRol IDENTIFIED BY contraseña**

Si se pone contraseña indica que el usuario que deseé usar los privilegios tiene que introducir la clave de acceso en la orden SET ROLE para activar el rol.

El usuario que crea el rol ha de ser administrador o tener el privilegio CREATE ROLE (GRANT CREATE ROLE TO NOMBREUSUARIO)

Ejemplos:

CREATE ROLE ACCESO;

A continuación hemos de conceder privilegios al rol usando la orden GRANT.

Asignamos los privilegios al rol ACCESO: SELECT e INSERT sobre la tabla EMP, INSERT en la tabla DEPT y CREATE SESSION para poder iniciar sesión en Oracle:

```
CONNECT SYSTEM/manager;
GRANT SELECT, INSERT
  ON SCOTT.EMP
  TO ACCESO;
GRANT INSERT
  ON SCOTT.DEPT
  TO ACCESO;
GRANT CREATE SESSION
```

TO ACCESO;

Creamos un usuario para concederle el rol creado:

```
CREATE USER MIGUEL  
IDENTIFIED BY MIGUEL  
QUOTA 100K ON SYSTEM;  
GRANT ACCESO  
TO MIGUEL;
```

Le damos el privilegio CREATE TABLE:

```
GRANT CREATE TABLE  
TO ACCESO;
```

Ahora nos conectamos como MIGUEL y hacemos SELECT e INSERT en EMP y DEPT:

```
CONNECT MIGUEL/MIGUEL;  
SELECT ENAME, SAL  
FROM SCOTT.EMP  
WHERE DEPTNO=10; => 3 empleados  
INSERT INTO SCOTT.DEPT  
VALUES (60,'INFORMATICA','MADRID');
```

Miguel no puede hacer SELECT en DEPT:

```
SELECT *  
FROM SCOTT.DEPT; => "privilegios insuficientes"  
ROLLBACK;
```

5.3.1. LÍMITES EN PRIVILEGIOS SOBRE ROLES

Un rol puede decidir el acceso de un usuario a un objeto, pero no puede permitir la creación de objetos. Veámoslo con un ejemplo.

MIGUEL puede crear vistas haciendo GRANT CREATE VIEW TO MIGUEL;

También puede hacer SELECT en la tabla EMP de SCOTT por el rol ACCESO:

```
CONNECT MIGUEL/MIGUEL;  
SELECT *  
FROM SCOTT.EMP  
WHERE DEPTNO=10;
```

Podrá crear vistas por el privilegio CREATE VIEW:

```
CREATE VIEW V1 AS  
SELECT *  
FROM SESSION_PRIVS; => vista creada
```

Pero no puede crear una vista sobre la tabla EMP debido a que recibió el privilegio SELECT a través del rol ACCESO:

```
CREATE VIEW V2  
AS SELECT *  
FROM SCOTT.EMP  
WHERE DEPTNO=10; => "la tabla o vista no existe"
```

Sería necesario que el privilegio SELECT sobre la tabla EMP se le concediese directamente a MIGUEL (no al rol ACCESO) por parte del administrador:

```
GRANT SELECT, INSERT  
ON SCOTT.EMP  
TO MIGUEL; => ahora sí podrá crear la vista V2 anterior
```

5.3.2. SUPRESIÓN DE PRIVILEGIOS EN LOS ROLES

La orden REVOKE permite suprimir los privilegios dados a los roles

Retirar del rol ACCESO el privilegio INSERT en la tabla EMP. Ahora, si MIGUEL intenta insertar una fila en EMP no podrá:

```
REVOKE INSERT
  ON SCOTT.EMP
  FROM ACCESO;
```

Retirar del rol ACCESO el privilegio CREATE TABLE:

```
REVOKE CREATE TABLE
  FROM ACCESO;
```

5.3.3. SUPRESIÓN DE UN ROL

DROP ROLE Nombrerol;

Permite eliminar un rol de la BBDD. Se retira el rol concedido a todos los usuarios y roles a los que se les concedió. Es necesario ser administrador o tener el privilegio DROP ANY ROLE

```
DROP ROLE ACCESO;
```

Si MIGUEL intenta conectarse a Oracle no podrá, ya que se ha eliminado ACCESO que le otorgaba el privilegio de iniciar sesión en Oracle.

CONNECT MIGUEL/MIGUEL => "user MIGUEL lacks CREATE SESSION privilege; logon denied"

5.3.4. ESTABLECER UN ROL POR DEFECTO

Es posible establecer un rol por defecto mediante la orden ALTER USER, cuyo formato es:

```
ALTER USER Nombreusuario DEFAULT      ROLE
  {nombre_rol [...] | ALL [ EXCEPT nombre_rol [...] ] | NONE};
```

Con CREATE USER no se puede asignar un rol por defecto

La sentencia SQL SET ROLE permite activar o desactivar un rol.

```
SET ROLE { nombre_rol [ IDENTIFIED BY la_contraseña ] [...]
  | ALL [ EXCEPT nombre_rol [...] ] | NONE};
```

5.3.5. INFORMACIÓN SOBRE ROLES EN EL DICCIONARIO DE DATOS

USER_TAB_PRIVS: concesiones sobre objetos que son propiedad del usuario, concedidos o recibidos por éste

ROLE_SYS_PRIVS: privilegios del sistema asignados a roles

ROLE_TAB_PRIVS: privilegios sobre tablas aplicados a roles

ROLE_ROLE_PRIVS: roles asignados a otros roles

SESSION_ROLES: roles activos para el usuario

USER_ROLE_PRIVS: roles asignados al usuario

DBA_SYS_PRIVS: privilegios del sistema asignados a los usuarios o roles

DBA_ROLE_PRIVS: privilegios asignados a todos los usuarios y roles

DBA_ROLES: todos los roles

5.4. PERFILES

Un **perfil** es un conjunto de límites de recursos en la BBDD. Por ejemplo, podemos crear un perfil que limite el tiempo de conexión a la BBDD. **Por omisión, un usuario recibe el perfil DEFAULT** cuando se le da de alta por primera vez. En principio este perfil define recursos ilimitados (UNLIMITED).

CREATE PROFILE NombrePerfil LIMIT Recurso [Entero | UNLIMITED | DEFAULT]

UNLIMITED significa que no hay límite sobre un recurso particular
DEFAULT coge el límite del perfil DEFAULT

RECURSO	FUNCIÓN
SESSIONS_PER_USER	Número de sesiones múltiples concurrentes permitidas por nombre de usuario
CONNECT_TIME	Límite el tiempo de conexión permitido por sesión antes de que el usuario sea desconectado (minutos)
IDLE_TIME	Límite el tiempo de inactividad permitido antes de que el usuario sea desconectado (minutos)
CPU_PER_CALL	Tiempo límite de respuesta para una llamada en centésimas de segundo
CPU_PER_SESSION	Límite el tiempo máximo de CPU por sesión (centésimas de segundo)
LOGICAL_READS_PER_CALL	Número máximo de bloques de datos leídos en una llamada.
LOGICAL_READS_PER_SESSION	Número máximo de bloques de datos leídos en una sesión.
PRIVATE_SGA	Tamaño de memoria que una sesión puede reservar del espacio privado de memoria del SGA.
COMPOSITE_LIMIT	Coste total en recursos de una sesión, expresado mediante el promedio de ciertos parámetros.

Ejemplos: Crear el perfil PERFIL1 donde limitamos a uno el número de sesiones concurrentes por usuario: **CREATE PROFILE PERFIL1 LIMIT SESSIONS_PER_USER 1;**

Los demás límites de recurso, como no se mencionan en la instrucción, se asignan los valores que tenga por defecto el sistema (normalmente UNLIMITED)

Crear un usuario PRUEBA al que se le asigna PERFIL1 y se le concede el rol CONNECT:

```
CREATE USER PRUEBA
IDENTIFIED BY PRUEBA
QUOTA 100K ON SYSTEM
PROFILE PERFIL1;
```

```
GRANT CONNECT
TO PRUEBA;
```

Para activar el uso de perfiles en el sistema, el administrador ha de ejecutar lo siguiente:

```
ALTER SYSTEM SET RESOURCE_LIMIT = TRUE;
```

Conectamos como PRUEBA/PRUEBA y ejecutamos otra sesión de SQL Plus para conectar con el mismo usuario: **ERROR: "exceeded simultaneous SESSIONS_PER_USER limit"**

5.4.1. MODIFICACIÓN DE UN PERFILE

```
ALTER PROFILE nombreperfil LIMIT ...
```

5.4.2. BORRADO DE UN PERFILE

```
DROP PROFILE nombreperfil [CASCADE];
```

La opción CASCADE será necesaria cuando algún usuario de la BBDD tenga asignado dicho perfil para que primero lo desasigne y después borre el perfil.

DROP PROFILE PERFIL1; => "PERFIL1 tiene usuarios asignados, no puede borrarlo sin CASCADE"

5.5. El DICCIONARIO de Datos

Oracle posee un diccionario de datos; es decir la manera de extraer el catálogo de objetos de una base de datos, nos referimos a: tablas, usuarios, roles, vistas, columnas de las tablas, secuencias, constraints, sinónimos, índices, triggers, funciones etc.., esta información se encuentra contenida en tablas y vistas del sistema. Dichas tablas en base a las cuales podemos obtener esta información aplicando sentencias sql.

A continuación enumeramos las tablas más importantes del diccionario de datos que nos permitirá obtener información de los objetos de la base de datos.

Los prefijos **dba_** son exclusividad para los usuarios que tengan el rol dba que sería el administrador de la base de datos y mostraría información de accesos de los usuarios que tienen asignado dicho rol.

Los prefijos **user_** indican que mostrará información en el ámbito del usuario que se encuentre conectado; [esquema].[objeto].

Los prefijos **all_** indican que mostrará información en el ámbito que tenga permisos el usuario.

- **Información sobre todos los objetos:** **tablas, vistas, funciones, procedimientos, índices, triggers, etc.** : dba_objects, user_objects, all_objects
- **Código de funciones y procedimientos:** dba_source, user_source, all_source
- **Usuarios:** dba_users, user_users, all_users
- **Roles:** dba_roles
- **Roles asignados a roles o usuarios:** dba_role_privs, user_role_privs
- **Privilegios asignados a roles o usuarios:** dba_sys_privs
- **Permisos sobre tablas asignados a roles o usuarios:** dba_tab_privs
- **Límites de recursos:** user_resource_limits
- **Perfiles y sus límites de recursos asociados:** dba_profiles
- **Límites de recursos en cuanto a restricciones en claves:** user_password_limits
- **Límites de recursos en cuanto a espacio máximo en tablespaces:** dba_ts_quotas, user_ts_quotas
- **Tablespaces:** dba_tablespaces, user_tablespaces
- **Ficheros que componen los datafiles:** dba_data_files
- **Segmentos:** dba_segments, user_segments, all_segments
- **Segmentos de Rollback:** dba_rollback_segs
- **Extensiones que forman los segmentos:** dba_extents, user_extents
- **Bloques libres:** dba_free_space, user_free_space
- **Bloques libres que podrían unirse:** dba_free_space_coalesced
- **Secuencias:** dba_sequences, user_sequences, all_sequences
- **Tablas, vistas, sinónimos y secuencias:** dba_catalog, user_catalog, all_catalog
- **Tablas :** dba_tables, user_tables, all_tables
- **Campos de tablas:** dba_cons_columns, user_cons_columns, all_cons_columns
- **Columnas de las tablas:** dba_tab_columns, user_tab_columns, all_tab_columns
- **Vistas:** dba_views, user_views, all_views
- **Sinónimos:** dba_synonyms, user_synonyms, all_synonyms
- **Restricciones de clave primaria, externa, not null, integridad referencial:** dba_constraints, user_constraints, all_constraints
- **Índices:** dba_indexes, user_indexes, all_indexes
- **Columnas de los índices:** dba_ind_columns, user_ind_columns, all_ind_columns

6. EJERCICIOS y PRACTICAS

6.1. Ejercicios LMD

Base de datos OBRAS

CONDUCTORES

Columna	Tipo	Nulo	Pred
CODC	char(3)	No	
NOMBRE	char(25)	No	
LOCALIDAD	char(20)	Sí	NULL
CATEG	int(11)	Sí	NULL

TRABAJOS

Columna	Tipo	Nulo	Pred
CODC	char(3)	No	
CODM	char(3)	No	
CODP	char(3)	No	
FECHA	date	No	
TIEMPO	int(11)	Sí	NULL

C01	José Sánchez	Arganda	18
C02	Manuel Díaz	Arganda	15
C03	Juan Pérez	Rivas	20
C04	Luis Ortiz	Arganda	18
C05	Javier Martín	Loeches	12
C06	Carmen López	Rivas	15

MAQUINAS

Columna	Tipo	Nulo	Pred
CODM	char(3)	No	
NOMBRE	char(20)	No	
PRECIOHORA	int(11)	Sí	NULL

C01	M02	P02	14/09/2002	120
C01	M02	P04	20/09/2002	NULL
C01	M03	P04	18/09/2002	180
C02	M03	P01	10/09/2002	100
C02	M03	P01	21/09/2002	NULL
C02	M03	P02	17/09/2002	NULL
C02	M03	P03	15/09/2002	30
C03	M01	P02	11/09/2002	200
C03	M01	P04	16/09/2002	300
C04	M03	P02	13/09/2002	90
C05	M03	P02	12/09/2002	150
C05	M03	P04	19/09/2002	90

M01	Excavadora	15000
M02	Hormigonera	10000
M03	Volquete	11000
M04	Apisonadora	18000

PROYECTOS

Columna	Tipo	Nulo	Pred
CODP	char(3)	No	
DESCRIP	char(25)	No	
LOCALIDAD	char(20)	Sí	NULL
CLIENTE	char(25)	Sí	NULL
TELEFONO	char(9)	Sí	NULL

P01	Garaje	Arganda	Felipe Sol	600111111
P02	Solado	Rivas	José Pérez	912222222
P03	Garaje	Arganda	Rosa López	666999666
P04	Techado	Loeches	José Pérez	913333333
P05	Buhardilla	Rivas	Ana Botijo	NULL

6.1.1. Ejercicios RESUELTOS

1. Subir el precio por hora en un 10% del precio por hora más bajo para todas las máquinas excepto para aquella que tenga el valor más alto.

```
UPDATE maquinas
SET preciohora = preciohora + (SELECT MIN(preciohora)*0.1
                                FROM maquinas)
WHERE preciohora NOT IN (SELECT MAX(preciohora)
                           FROM maquinas);
```

2. Subir la categoría un 15% a los conductores que no hayan trabajado con Volquete y hayan trabajado en más de un proyecto distinto.

```
UPDATE conductores
SET categ = categ*1.15
WHERE CodC NOT IN (SELECT CodC
                     FROM trabajos
                     WHERE codM IN (SELECT codM
                                    FROM maquinas
                                    WHERE nombre = 'Volquete'))
AND
CodC IN (SELECT CodC
          FROM trabajos
          GROUP BY CodC
          HAVING COUNT(DISTINCT CodP) > 1);
```

3. Eliminar el proyecto Solado de José Pérez.

1.

```
DELETE
FROM trabajos
WHERE CodP IN (SELECT CodP
                  FROM proyectos
                  WHERE descrip = 'Solado' AND cliente = 'José Pérez' );
DELETE
FROM Proyectos
WHERE descrip = 'Solado' AND cliente = 'José Pérez' ;
```

2.

```
DELETE
FROM Proyectos
WHERE descrip = 'Solado' AND cliente = 'José Pérez' ;
DELETE
FROM trabajos
WHERE CodP NOT IN (SELECT CodP
                      FROM proyectos);
```

4. Insertar en la tabla trabajos la fila 'C01', 'M04','P07','19/09/02',100.

```
INSERT INTO trabajos
VALUES ('C01', 'M04','P07','19/sep/02',null);
```

Inserción errónea, por no cumplir la restricción de integridad referencial el valor 'P07'

5. Eliminar el conductor 'C01' de la tabla conductores.

```
DELETE
FROM conductores
WHERE codC = 'C01';
```

Imposible eliminar la fila porque se incumpliría la restricción de integridad referencial con la tabla trabajos.

6. Modificar el código del conductor 'C01' de la tabla conductores, por el código 'C05'.

```
UPDATE conductores
    SET codC = 'C05'
WHERE codC = 'C01'
```

Imposible modificar el valor porque crearía un valor duplicado en el índice de clave primaria.

7. Modificar el código del conductor 'C01' de la tabla conductores, por el código 'C07'.

```
UPDATE conductores
    SET codC = 'C07'
WHERE codC = 'C01'
```

Imposible modificar la fila porque se incumpliría la restricción de integridad referencial con la tabla trabajos.

6.1.2. Ejercicios PROYECTOS

- Define una tabla con todas las restricciones posibles, y prueba su funcionalidad. Prueba las distintas formas de meter valores nulos y de hacer uso de valores por defecto.
- Crea una tabla con el resultado de una consulta y a continuación insértale los valores obtenidos con otra consulta.
- Añadir restricciones (PRIMARY KEY, UNIQUE, NOT NULL, DEFAULT) a una tabla, la cual contiene datos.
- Prueba el funcionamiento de la cláusula AUTO_INCREMENT; qué pasa si se borran filas y luego se inserta, si se borra la fila con mayor valor de clave y se inserta una fila, etc.
- Crea dos tablas relacionadas y opera con ellas para poner en práctica todo el potencial y características de la **Integridad Referencial** a lo hora de hacer operaciones LMD (altas-bajas-modificaciones), usando ... ON DELETE CASCADE ON UPDATE CASCADE y sin usarlo.

F. BD: Gestión de HOSPITALES

- En la tabla EMP incrementar el salario el 10% a los empleados que tengan una comisión superior al 5% del salario.
- Buscar todos los empleados que tienen un *salario + comisión* superior a 2000 y asignarles como nuevo salario esta suma. Sólo para los que tienen comisión.

6.2. PRACTICAS con VISTAS (MySQL)

6.2.1. Prácticas RESUELTA

- Crear una vista que contenga el nombre del conductor, la descripción del proyecto y la media aritmética del tiempo trabajado.

```
CREATE VIEW ap33 (conductor, proyecto, tiempoMedio)
AS SELECT nombre, descrip, avg(tiempo)
FROM conductores, trabajos, proyectos
WHERE conductores.codC = trabajos.codC AND
trabajos.codP = proyectos.codP
GROUP BY nombre, descrip;
```

- Crear una vista sobre la tabla trabajos, para los trabajos realizados después del 15 de septiembre de 2002. Crearla sin la cláusula "With Check Option" y sin ella, comprobando su funcionamiento.

```
CREATE VIEW ap34a (conductor, maquina, proyecto, fecha, tiempo)
AS SELECT codc, codm, codp, fecha, tiempo
FROM trabajos
WHERE fecha > '15/sep/02';
```

```
INSERT INTO ap34
VALUES ('C01','M01','P01','03/sep/02',10);
```

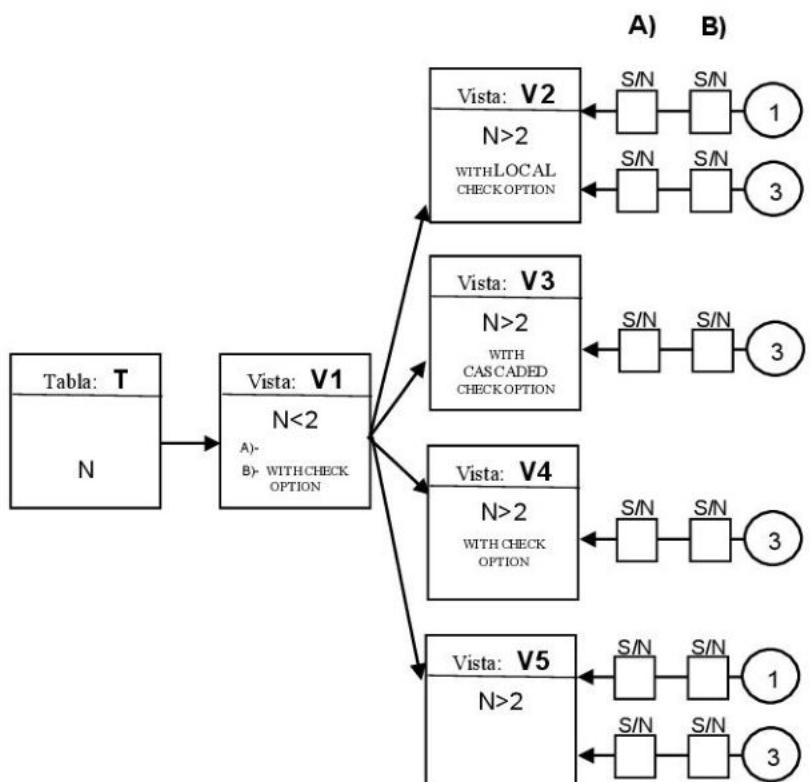
```
CREATE VIEW ap34b (conductor, maquina, proyecto, fecha, tiempo)
AS SELECT codc, codm, codp, fecha, tiempo
FROM trabajos
WHERE fecha > '15/sep/02'
WITH CHECK OPTION;
```

```
INSERT INTO ap34
VALUES ('C01','M01','P01','04/sep/02',10);
```

6.2.2. Prácticas PROPUESTAS

6.2.2.1. Práctica - A

El siguiente esquema muestra la creación de una serie de vistas sobre la tabla T o sobre otras vistas, con las distintas especificaciones de la cláusula WITH CHECK OPTION. Completar los recuadros S/N con **S** si el valor del círculo es **INSERTADO** o **N** si no lo es.



6.2.2.2. Práctica - B

Realiza una práctica donde se creen Vistas: horizontales, verticales, horizontales-verticales, sobre más de una tabla, etc.

6.2.2.3. Práctica - C

Realiza una práctica donde pongas a prueba la cláusula, tanto en la **insercción** de datos como en la **modificación**:

[WITH [CASCADED | LOCAL] CHECK OPTION]

6.2.2.4. Práctica - D

Realiza una práctica donde pongas a prueba las OPERACIONES que se pueden hacer con Vistas, y las **limitaciones** que existen según el tipo de operación (Insertar, Borrar, Modificar, Consultar).

- Sobre varias tablas.
- Subconsultas
- GROUP BY
- HAVING
- UNION o UNION ALL
- MAX – MIN ...
- DISTINCT
- ...

6.3. PRACTICAS de SEGURIDAD con Oracle

6.3.1. Prácticas RESUELTA

6.3.1.1. Práctica – A

- Crear los usuarios JUAN y PEDRO.
- Conectar con el usuario JUAN/JUAN y conceder a PEDRO el privilegio SELECT e INSERT sobre la tabla DEPT30

CONNECT JUAN/JUAN;

**GRANT SELECT, INSERT
ON DEPT30
TO PEDRO;**

PEDRO puede acceder a DEPT30 de JUAN:

**CONNECT PEDRO/PEDRO;
SELECT *
FROM JUAN.DEPT30;**

Sin embargo, SCOTT no puede operar con DEPT30 porque no tiene privilegios:

**CONNECT SCOTT/TIGER;
SELECT *
FROM JUAN.DEPT30;**

- JUAN da privilegios a SCOTT para insertar en DEPT30

CONNECT JUAN/JUAN;

**GRANT INSERT
ON DEPT30
TO SCOTT;**

CONNECT SCOTT/TIGER;

**SELECT *
FROM JUAN.DEPT30; => "privilegios insuficientes"**

**INSERT INTO JUAN.DEPT30
VALUES (50, 'VENTAS', 'MADRID');**

Si JUAN consulta su tabla verá que hay una fila más:

**CONNECT JUAN/JUAN;
SELECT *
FROM DEPT30;**

- JUAN da a PEDRO todos los privilegios sobre DEPT30 (no teclearla)

CONNECT JUAN/JUAN;

**GRANT ALL PRIVILEGES
ON DEPT30
TO PEDRO;**

Le daremos los permisos ALTER, DELETE, INDEX, INSERT, REFERENCES, SELECT, UPDATE sobre la tabla DEPT30

Con la orden GRANT se pueden conceder privilegios INSERT, UPDATE o REFERENCES sobre determinadas columnas de una tabla

- Dar privilegios a SCOTT sobre DEPT30 para que pueda modificar sólo la columna DNAME:

```
CONNECT JUAN/JUAN;
GRANT UPDATE (DNAME)
ON DEPT30
TO SCOTT;
```

- SCOTT intenta modificar DNAME y LOC pero no podrá:

```
CONNECT SCOTT/TIGER;
UPDATE JUAN.DEPT30
SET DNAME='DIRECCION', LOC='BARCELONA'
WHERE DEPTNO=10; => "privilegios insuficientes"
UPDATE JUAN.DEPT30
SET DNAME='DIRECCION'
WHERE DEPTNO=10; => 1 fila
CONNECT JUAN/JUAN;
SELECT *
FROM DEPT30; => se modificó la fila DEPTNO 10
```

Con la opción **WITH GRANT OPTION** el usuario que recibe los privilegios podrá concederlos a otros usuarios:

- SCOTT puede conceder privilegios a otros usuarios sobre sus tablas, pero no puede concederlos sobre tablas que no le pertenezcan. Intentar dar el privilegio INSERT a PEDRO sobre la tabla DEPT30 de JUAN: (Scott ya tiene ese privilegio)

```
CONNECT SCOTT/TIGER;
GRANT INSERT
ON JUAN.DEPT30
TO PEDRO; => "privilegios insuficientes"
```

- Ahora JUAN concede a SCOTT el privilegio de insertar en DEPT30 con WITH GRANT OPTION

```
CONNECT JUAN/JUAN;
GRANT INSERT
ON DEPT30
TO SCOTT
WITH GRANT OPTION;
```

```
CONNECT SCOTT/TIGER;
GRANT INSERT
ON JUAN.DEPT30
TO PEDRO;
```

Ahora PEDRO puede insertar filas en DEPT30:

```
CONNECT PEDRO/PEDRO;
INSERT INTO JUAN.DEPT30
VALUES (60, 'OTROS', 'SEVILLA'); => 1 fila insertada
```

6.3.1.2. Prácticas – B

1. Buscar en la documentación en línea y en bd el contenido de las vistas:

- **dba_profiles**
- **dba_roles**
- **dba_users**
- **dba_role_privs**
- **dba_tab_privs**
- **dba_sys_privs**

SQL> DESC DBA_PROFILES

SQL> DESC DBA_ROLES

SQL> DESC DBA_USERS

SQL> DESC DBA_ROLE_PRIVS

SQL> desc DBA_TAB_PRIVS

SQL> DESC DBA_SYS_PRIVS

2. Conectarse como usuario SYSTEM a la base y crear un usuario llamado "administrador" autenticado por la base de datos. Indicar como "tablespace" por defecto USERS y como "tablespace" temporal TEMP; asignar una cuota de 500K en el "tablespace" USERS.

```
SQL> CREATE USER ADMINISTRADOR IDENTIFIED BY ADMIN
      DEFAULT TABLESPACE USERS
      TEMPORARY TABLESPACE TEMP
      QUOTA 500K ON USERS;
```

User created.

```
SQL> SELECT USERNAME
      FROM DBA_USERS
      WHERE USERNAME='ADMINISTRADOR';

      USERNAME
      -----
      ADMINISTRADOR
```

3. Abrir una sesión sqlplus e intentar conectarse como usuario "administrador", ¿qué sucede?, ¿por qué?.

```
/u01/app/oracle/admin/CURSO01/creacion (CURSO01)> sqlplus SQL*Plus: Release 9.2.0.1.0 -
Production on Mon Nov 22 12:50:48 2004 Copyright (c) 1982, 2002, Oracle Corporation. All rights
reserved.
```

Enter user-name: administrador

Enter password: ERROR:

*ORA-01045: user ADMINISTRADOR lacks CREATE SESSION privilege;
logon denied*

4. Averiguar qué privilegios de sistema, roles y privilegios sobre objetos tiene concedidos el usuario "administrador".

```
SQL> select *
  from dba_role_privs
  where grantee='ADMINISTRADOR'; no rows selected
SQL> select *
  from dba_tab_privs
  where grantee='ADMINISTRADOR'; no rows selected
SQL> select *
  from dba_sys_privs
  where grantee='ADMINISTRADOR';
no rows selected
```

5. Otorgar el privilegio "CREATE SESSION" al usuario "administrador" e intentar de nuevo la conexión sqlplus.

```
SQL> grant create session
      to administrador;
Grant succeeded.
SQL> connect administrador          Enter password: Connected.
```

6. Conectarse como usuario "administrador" y crear un usuario llamado "prueba00" que tenga como "tablespace" por defecto USERS y como "tablespace" temporal TEMP; asignar una cuota de 0K en el "tablespace" USERS. ¿Es posible hacerlo?

```
SQL> show user
USER is "ADMINISTRADOR"
SQL> create user prueba00 identified by prueba00
      default tablespace users
      temporary tablespace temp
      quota 0k on users

create user prueba00 identified by prueba00
      *
```

ERROR at line 1:
ORA-01031: insufficient privileges

7. Conectado como usuario SYSTEM, otorgar el privilegio "create user" al usuario "administrador" y repetir el ejercicio anterior.

```
/u01/app/oracle/admin/CURSO01 (CURSO01)>sqlplus
```

SQL*Plus: Release 9.2.0.1.0 - Production on Mon Nov 22 12:55:31 2004 Copyright

(c) 1982, 2002, Oracle Corporation. All rights reserved.

Enter user-name: system

Enter password:

Connected to:

Oracle9i Enterprise Edition Release 9.2.0.1.0 - Production With the
Partitioning and Oracle Label Security options JServer Release 9.2.0.1.0 -
Production

SQL> grant create user to administrador; Grant
succeeded.

SQL> connect administrador Enter
password: Connected.

```
SQL> create user prueba00 identified by prueba00  
      default tablespace users  
      temporary tablespace temp 4  
      quota 0k on users
```

User created.

8. Averiguar que usuarios de la base de datos tienen asignado el privilegio "create user" de forma directa, ¿qué vista debe ser consultada?.

SQL> connect system

Introduzca su clave:

Connected.

SQL> desc dba_sys_privs

Nombre	¿Nulo?	Tipo
GRANTEE	NOT NULL	VARCHAR2(30)
PRIVILEGE	NOT NULL	VARCHAR2(40)
ADMIN_OPTION		VARCHAR2(3)

SQL> select * from dba_sys_privs where privilege ='CREATE USER';

GRANTEE	PRIVILEGE	
ADM		
DBA	CREATE USER	YES
ADMINISTRADOR	CREATE USER	NO
IMP_FULL_DATABASE	CREATE USER	NO

9. Hacer lo mismo para el privilegio "create session".

```
SQL> select * from dba_sys_privs where privilege ='CREATE SESSION'; GRANTEE
```

	PRIVILEGE	ADM
DBA	CREATE SESSION	YES
CONNECT	CREATE SESSION	NO
ADMINISTRADOR	CREATE SESSION	NO
RECOVERY_CATALOG_OWNER	CREATE SESSION	NO

10. Crear dos "tablespace" llamados NOMINA y ACADEMO, que contendrán datos relativos a las aplicaciones de nomina y datos académicos de los empleados de una empresa, según las siguientes características:

	ACADEMO	NOMINA
Tamaño inicial	1M	1M
Autoextensible	SI	SI
Extensión	200K	100K
Tamaño máximo	1400K	1500K
Parámetros almacenamiento	Initial	16K
	Next	16K
	Minextents	1
	Maxextents	3
Localización	/u02/oradata/<bd>	/u02/oradata/<bd>

Consulte la ayuda en línea si no recuerda la sintaxis exacta de la sentencia.

```
SQL> create tablespace academo
  datafile '/u02/oradata/CURSO01/academo01.dbf' size 1M
  autoextend on next 200k maxsize 1400K
  default storage (initial 16k next 16k
  minextents 1 maxextents 3);
```

Tablespace created.

```
SQL> create tablespace nomina
  datafile '/u02/oradata/CURSO01/nomina01.dbf' size 1M
  autoextend on next 100K maxsize 1500K
  default storage (initial 16k next 16k
  minextents 1 maxextents 3);
```

Tablespace created.

11. Crear dos "tablespace" temporales, manejados de forma local, llamados TEMP_NOMINA y TEMP_ACADEMO con las siguientes características:

	TEMP_ACADEMO	TEMP_NOMINA
Tamaño inicial	500K	600K
Autoextensible	SI	SI
Extensión	50K	50K
Tamaño máximo	600K	700K
Localización	/u04/oradata/<bd>	/u04/oradata/<bd>

```
SQL> create temporary tablespace temp_academo
      tempfile '/u04/oradata/CURSO01/temp_academo01.dbf'
      size 500k autoextend on next 50k maxsize 600k
      extent management local uniform size 100k;
```

Tablespace created.

```
SQL> create temporary tablespace temp_nomina
      tempfile '/u04/oradata/CURSO01/temp_nomina01.dbf'
      size 600k autoextend on next 50k maxsize 700k
      extent management local uniform size 100k;
```

Tablespace created.

12. Estando conectado como usuario “administrador” probar a crear un rol llamado “administrador”, ¿qué ocurre?.

```
SQL> connect administrador
Enter password: Connected.
```

```
SQL> create role administrador; create
role administrador
*
```

ERROR at line 1:
ORA-01031: insufficient privileges

13. Idem estando conectado como usuario SYSTEM, ¿qué sucede?, ¿por qué?.

```
SQL> connect system Enter
password: Connected.
```

```
SQL> create role administrador; create role administrador
* ERROR at line 1:
ORA-01921: role name 'ADMINISTRADOR' conflicts with another user or role name
```

14. Comprobar en el diccionario de datos los usuarios o roles que poseen el privilegio "CREATE ROLE".

```
SQL> select * from dba_sys_privs where privilege ='CREATE ROLE'; GRANTEE
```

	PRIVILEGE	ADM
DBA	CREATE ROLE	YES
IMP_FULL_DATABASE	CREATE ROLE	NO

15. Crear un rol llamado "ADMIN", asignarle los privilegios "create session", "create user" y "CREATE ROLE". Asignarlo al usuario administrador.

```
SQL> create role admin;
```

Role created.

```
SQL> grant create session to admin;
```

Grant succeeded.

```
SQL> c.session.user.
```

```
1* grant create user to admin
```

```
SQL> grant create user to admin
```

Grant succeeded.

```
SQL> c.user.role.
```

```
1* grant create role to admin
```

```
SQL> grant create role to admin
```

Grant succeeded.

```
SQL> grant admin to administrador;
```

Grant succeeded.

16. Consultar los privilegios de sistema que tiene asignados de forma directa el usuario "administrador", revocarlos y asignarle el rol "admin".

```
SQL> select * from dba_sys_privs where grantee ='ADMINISTRADOR'
```

GRANTEE	PRIVILEGE	ADM
---------	-----------	-----

ADMINISTRADOR	CREATE SESSION	NO
---------------	----------------	----

ADMINISTRADOR	CREATE USER	NO
---------------	-------------	----

```
SQL> revoke create session from administrador;
```

Revoke succeeded.

```
SQL> c.session.user.
```

```
1* revoke create user from administrador
```

```
SQL> revoke create user from administrador Revoke
succeeded.
```

```
SQL> grant admin to administrador;
```

Grant succeeded.

17. Crear, conectado como SYSTEM, un usuario llamado "prueba01" autenticado por base de datos al que no se le asigne "tablespace" por defecto ni temporal.

```
SQL> create user prueba01 identified by prueba01;
```

User created.

18. Consultar en las vistas correspondientes los "tablespaces" y la quota en cada uno de ellos que tiene los usuarios SYS, SYSTEM, "administrador", "prueba00" y "prueba01". ¿Qué ha ocurrido con el usuario "prueba01"?

```
SQL> select substr(username,1,15) usuario, DEFAULT_TABLESPACE ,TEMPORARY_TABLESPACE
  from dba_users
 where username in ('SYS','SYSTEM','ADMINISTRADOR','PRUEBA00','PRUEBA01');
```

USUARIO	DEFAULT_TABLESPACE	TEMPORARY_TABLESPACE
PRUEBA01	SYSTEM	SYSTEM
PRUEBA00	USERS	TEMP
ADMINISTRADOR	USERS	TEMP
SYSTEM	SYSTEM	TEMP
SYS	SYSTEM	TEMP

```
SQL> select substr(username,1,15) usuario, tablespace_name, max_bytes
  from dba_ts_quotas
 where username in ('SYS','SYSTEM','ADMINISTRADOR','PRUEBA00','PRUEBA01')
```

USUARIO	TABLESPACE_NAME	MAX_BYTES
ADMINISTRADOR	USERS	512000

19. Crear un usuario llamado "prueba02" autenticado por base de datos, asignando como "tablespace" por defecto NOMINA y como "tablespace" temporal TEMP_NOMINA (no se le asignara cuota en NOMINA).

```
SQL> create user prueba02 identified by prueba02
      default tablespace nomina
      temporary tablespace temp_nomina;
```

User created.

20. Asignar al usuario "prueba01" los "tablespace" ACADEMO y TEMP_ACADEMO como "tablespace" de trabajo y temporal respectivamente (sin especificar cuota).

```
SQL> alter user prueba01 temporary tablespace temp_academo;
```

User altered.

```
SQL> alter user prueba01 default tablespace academo;
```

User altered.

21. Consultar en las vistas correspondientes los "tablespace" y la cuota en cada uno de ellos que tiene los usuarios "prueba01" y "prueba02".

```
SQL> select *
  from dba_ts_quotas
  where username in ('PRUEBA01','PRUEBA02');
no rows selected
```

22. Crear un rol llamado "CONEXIÓN" y asignarle el permiso "CREATE SESSION".

```
SQL> create role conexion;
Role created.
SQL> grant create session to conexion; Grant
succeeded.
```

23. Asignar el rol "CONEXIÓN" a los usuarios "prueba00", "prueba01" y "prueba02".

```
SQL> grant conexion to prueba00, prueba01, prueba02; Grant
succeeded.
```

24. Comprobar en la vista correspondiente cuales son los roles asignados a los usuarios "prueba00", "prueba01" y "prueba02".

```
SQL> select *
  from dba_role_privs
  where grantee in ('PRUEBA00','PRUEBA01','PRUEBA02');
```

GRANTEE	GRANTED_ROLE	ADM	DEF
PRUEBA00	CONEXION	NO	YES
PRUEBA01	CONEXION	NO	YES
PRUEBA02	CONEXION	NO	YES

25. Conectarse como usuario "prueba01" y crear la tabla siguiente en el "tablespace" ACADEMO:

```
CREATE TABLE CODIGOS
  (CODIGO varchar2(3), DESCRIPCION
   varchar2(20))
TABLESPACE ACADEMO STORAGE
  (INITIAL 64K
   NEXT 64K MINEXTENTS 5
   MAXEXTENTS 10);
```

¿Es posible hacerlo?, ¿falta algún permiso?.

```
SQL> connect prueba01 Enter
password: Connected.
```

```
SQL> CREATE TABLE CODIGOS
  2  (CODIGO varchar2(3),
  3  DESCRIPCION varchar2(20))
  4  TABLESPACE ACADEMO
  5  STORAGE (INITIAL    64K
  6  NEXT   64K
  7  MINEXTENTS 5
  8* MAXEXTENTS 10)
CREATE TABLE CODIGOS
*
```

ERROR at line 1:
ORA-01031: insufficient privileges

26. Crear un rol llamado "DESARROLLO" y asignarle los permisos "CREATE SEQUENCE", "CREATE SESSION", "CREATE SYNONYM", "CREATE TABLE" y "CREATE VIEW". Asignar el rol "DESARROLLO" a los usuarios "prueba00", "prueba01" y "prueba02".

SQL> connect system Enter

password: Connected.

SQL> create role desarrollo; Role

created.

SQL> grant create sequence, create session, create synonym, create table, create view
to desarrollo;

Grant succeeded.

SQL> grant desarrollo to prueba00, prueba01, prueba02; Grant

succeeded.

27. Volver a conectarse como usuario "prueba01" y crear la tabla anterior en el "tablespace" ACADEMO.

SQL> connect prueba01

Introduzca su clave:

Connected.

```
SQL> CREATE TABLE CODIGOS
  2  (CODIGO varchar2(3),
  3  DESCRIPCION varchar2(20))
  4  TABLESPACE ACADEMO
  5  STORAGE (INITIAL    64K
  6  NEXT   64K
  7  MINEXTENTS 5
  8* MAXEXTENTS 10)
```

```
CREATE TABLE CODIGOS
```

```
*
```

ERROR at line 1:

ORA-01950: no privileges on tablespace 'ACADEMO'

28. Asignar cuota ilimitada al usuario "prueba01" en el "tablespace" ACADEMO.

Volver a repetir el ejercicio 26.

```
SQL> connect system
```

Enter password:

Connected.

```
SQL> alter user prueba01 quota unlimited on academo;
```

User altered.

```
SQL> connect prueba01
```

Enter password:

Connected.

```
SQL> CREATE TABLE CODIGOS
```

(CODIGO varchar2(3),

DESCRIPCION varchar2(20))

TABLESPACE ACADEMO

STORAGE (INITIAL 64K NEXT 64K

MINEXTENTS 5

MAXEXTENTS 10);

Table created.

29. Asignar cuota ilimitada al usuario "prueba02" en el "tablespace" NOMINA.

```
SQL> connect system
```

Introduzca su clave:

Connected.

```
SQL> alter user prueba02 quota unlimited on academo;
```

User altered.

30. Obtener información sobre roles, privilegios de sistema, "tablespace" y cuotas para los usuarios "prueba00", "prueba01" y "prueba02".

```
SQL> select *
```

```
  from dba_role_privs
```

```
 where grantee in ('PRUEBA00','PRUEBA01','PRUEBA02');
```

GRANTEE	GRANTED_ROLE	ADM	DEF
PRUEBA00	CONEXION	NO	YES
PRUEBA00	DESARROLLO	NO	YES
PRUEBA01	CONEXION	NO	YES
PRUEBA01	DESARROLLO	NO	YES
PRUEBA02	CONEXION	NO	YES
PRUEBA02	DESARROLLO	NO	YES

6 rows selected.

SQL> select *

```
from dba_sys_privs
where grantee in ('PRUEBA00','PRUEBA01','PRUEBA02');
```

no rows selected

SQL> select USERNAME , TABLESPACE_NAME , BYTES

```
from dba_ts_quotas
where username in ('PRUEBA00','PRUEBA01','PRUEBA02');
```

USERNAME	TABLESPACE_NAME	BYTES
PRUEBA02	ACADEMO	0
PRUEBA01	ACADEMO	327680

31. Asignar cuota cero en el "tablespace" por defecto para el usuario "prueba01", ¿siguen estando sus objetos?, ¿es posible crear algún otro? (probad a crear un tabla).

SQL> alter user prueba01 quota 0k on academo; User altered.

SQL> select owner, table_name from dba_tables where owner='PRUEBA01';

OWNER	TABLE_NAME
PRUEBA01	CODIGOS

SQL> connect prueba01

Enter password:

Connected.

```
SQL> CREATE TABLE CODIGOS2(
    CODIGO varchar2(3),
    DESCRIPCION varchar2(20))
TABLESPACE ACADEMO
STORAGE (INITIAL 64K NEXT      64K
         MINEXTENTS 5
         MAXEXTENTS 10);
```

CREATE TABLE CODIGOS2(CODIGO varchar2(3),

ERROR at line 1:

ORA-01536: space quota exceeded for tablespace 'ACADEMO'

- 32.** Conectarse como usuario "prueba01" e intentar modificar su cuota en el "tablespace" ACADEMO, ¿es posible?

SQL> connect prueba01

Introduzca su clave:

Connected.

SQL> alter user prueba01 quota unlimited on academo;
alter user prueba01 quota unlimited on academo

ERROR at line 1:

ORA-01031: insufficient privileges

- 33.** Conectarse como usuario "prueba01" y modificar su clave, ¿es posible?.

SQL> alter user prueba01 identified by probando01;
User altered.

- 34.** Averiguar que usuarios o roles de base de datos tienen asignado el privilegio ALTER USER.

SQL> connect system

Introduzca su clave:

Conectado.

*SQL> select * from dba_sys_privs where privilege='ALTER USER';*

GRANTEE	PRIVILEGE	ADM
DBA	ALTER USER	YES

- 35.** Abrir una sesión con el usuario "administrador" y otra con el usuario "prueba02". Siendo el usuario "administrador", intentar borrar el usuario "prueba02".

SQL> show user

USER es "SYSTEM"

SQL> drop user prueba02; drop
user prueba02

ERROR en línea 1:

ORA-01940: no se puede borrar un usuario conectado actualmente

- 36.** Asignar el permiso DROP USER al rol ADMIN.

SQL> grant drop user to admin;
Grant succeeded.

37. Averiguar que usuarios o roles de base de datos tienen asignado el privilegio DROP USER.

```
SQL> select * from dba_sys_privs where privilege='DROP USER';
GRANTEE          PRIVILEGE      ADM
-----
ADMIN            DROP USER      NO
DBA              DROP USER      YES
IMP_FULL_DATABASE  DROP USER    NO
```

38. Conectado como usuario "administrador", crear el usuario "prueba03" autentificado por base de datos y asignando cuotas en el "tablespace" ACACDEMO (500K) y NOMINA (200K). Su "tablespace" temporal será TEMP.

SQL> connect administrador

Enter password:

Connected.

SQL> create user prueba03 identified by prueba03

```
default tablespace academo
temporary tablespace temp
quota 500k on academo quota
200k on nomina
```

User created.

39. Comprobar en el fichero de inicialización si está activado el modo de limitación de recursos.

Editar con el editor vi, por ejemplo, el fichero de inicializacion.

40. Averiguar que usuarios de base de datos o que roles tienen asignado el privilegio "CREATE PROFILE".

SQL> connect system Enter

password: Connected.

*SQL> select * from dba_sys_privs where privilege='CREATE PROFILE';*

GRANTEE	PRIVILEGE	ADM
DBA	CREATE PROFILE	YES
IMP_FULL_DATABASE	CREATE PROFILE	NO

41. Asignar el permiso "CREATE PROFILE" al rol ADMIN.

SQL> grant create profile to admin;

Grant succeeded.

42. Averiguar que perfiles están definidos en la base de datos y que límites de recursos fija cada uno de ellos.

```
SQL> select substr(profile,1,12) perfil, substr(resource_name,1,25) recurso,
resource_type, substr(limit,1,10) limite
from dba_profiles
order by profile, resource_name;
```

PERFIL	RECURSO	RESOURCE	LIMITE
DEFAULT	COMPOSITE_LIMIT	KERNEL	UNLIMITED
DEFAULT	CONNECT_TIME	KERNEL	UNLIMITED
DEFAULT	CPU_PER_CALL	KERNEL	UNLIMITED
DEFAULT	CPU_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	FAILED_LOGIN_ATTEMPTS	PASSWORD	UNLIMITED
DEFAULT	IDLE_TIME	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_CALL	KERNEL	UNLIMITED
DEFAULT	LOGICAL_READS_PER_SESSION	KERNEL	UNLIMITED
DEFAULT	PASSWORD_GRACE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LIFE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_LOCK_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_MAX	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_REUSE_TIME	PASSWORD	UNLIMITED
DEFAULT	PASSWORD_VERIFY_FUNCTION	PASSWORD	NULL
DEFAULT	PRIVATE_SGA	KERNEL	UNLIMITED
DEFAULT	SESSIONS_PER_USER	KERNEL	UNLIMITED

16 rows selected.

43. Consultar que perfiles tiene asignados cada usuario de la base de datos.

```
SQL> select username, profile from dba_users order by username;
USERNAME           PROFILE
```

ADMINISTRADOR	DEFAULT
DBSNMP	DEFAULT
OUTLN	DEFAULT
PRUEBA00	DEFAULT
PRUEBA01	DEFAULT
PRUEBA03	DEFAULT
SCOTT	DEFAULT
SYS	DEFAULT
SYSTEM	DEFAULT

9 rows selected.

44. Crear un perfil llamado "DESARROLLO" con las siguientes especificaciones:

Sessions_per_user	2
Cpu_per_session	unlimited
Cpu_per_call	6000
Connect_time	480
Idle_time	2
Failed_login_attempts	2
Password_life_time	120

```
SQL>create profile desarrollo limit sessions_per_user 2 cpu_per_session unlimited cpu_per_call
6000 connect_time 480 idle_time 2 failed_login_attempts 2
password_life_time 120; Profile
created.
```

45. Asignar el perfil anterior a los usuarios "prueba00", "prueba01", "prueba02" y "prueba03".

```
SQL>alter user prueba00 profile desarrollo;
User altered.
SQL>c.00.01
1* alter user prueba01 profile desarrollo
SQL>r
1* alter user prueba01 profile desarrollo
User altered.

SQL>c.01.02
1* alter user prueba02 profile desarrollo
SQL>r
1* alter user prueba02 profile desarrollo User
altered.

SQL>c.02.03
1* alter user prueba03 profile desarrollo
SQL>r
1* alter user prueba03 profile desarrollo
User altered.
```

46. Intentar la conexión dos veces como usuario "prueba01" fallando la contraseña, ¿qué sucede?. Comprobar si la cuenta ha sido bloqueada en la vista de base de datos correspondiente.

```
SQL>connect prueba01
Enter password: ERROR:
ORA-01017: invalid username/password; logon denied Warning: You are
no longer connected to ORACLE.
...
SQL>connect prueba01
Enter password: ERROR:
```

ORA-28000: the account is locked

```
SQL> select username, lock_date from dba_users where username like 'PRUEBA%';  
          USERNAME  
          LOCK_DATE  
-----  
----- PRUEBA03  
PRUEBA00  
PRUEBA01           22-NOV-04
```

47. Crear un usuario "prueba04" con el parámetro "password expire", sus "tablespace" por defecto y temporal serán USERS (cuota 0k) y TEMP. Asignar los roles CONEXIÓN y DESARROLLO. Conectarse como usuario "prueba04", ¿qué sucede?.

```
SQL> create user prueba04 identified by prueba04  
default tablespace users  
temporary tablespace temp  
quota 0k on users 5  
*      password expire
```

User created.

```
SQL> grant conexion, desarrollo to prueba04;  
Grant succeeded.
```

```
SQL> connect prueba04  
Enter password: ERROR:  
ORA-28001: the password has expired
```

```
Changing password for prueba04  
New  
password:  
Retype new password:  
Password changed  
Connected.
```

48. Bloquear la cuenta del usuario "prueba04", ¿qué sucede al conectarse de nuevo?.

```
SQL> connect system Enter  
password: Connected.
```

```
SQL> alter user prueba04 account lock;  
User altered.  
SQL> connect prueba04  
Enter password: ERROR:  
ORA-28000: the account is locked
```

Warning: You are no longer connected to ORACLE.

49. Modificar el "tablespace" por defecto y el temporal del usuario "prueba01" de forma que sean NOMINA y TEMP_NOMINA.

```
SQL> connect system Enter  
password: Connected.
```

```
SQL> alter user prueba04 default tablespace nomina;
```

User altered.

```
SQL> alter user prueba04 temporary tablespace temp_nomina;  
User altered.
```

50. Comprobar cual es el valor del parámetro OS_AUTHENT_PREFIX en la base de datos.

Editar con vi o ejecutar la sentencia pg sobre el fichero de parametros de inicializacion (init<SID>.ora).

51. Cambia la identificación del usuario "prueba01" de forma que sea identificado por el sistema operativo.

```
SQL> alter user prueba01 identified externally;  
User altered.  
SQL> set head off  
SQL> select * from dba_users where username='PRUEBA01'
```

PRUEBA01	26 EXTERNAL
LOCKED(TIMED)	22-NOV-04 22-MAR-05
ACADEMO	TEMP_ACADEMO
DESARROLLO	DEFAULT_CONSUMER_GROUP

- 52.** Modificar el parámetro OS_AUTHENT_PREFIX de forma que, en adelante, la cadena que identifique a un usuario externo sea "" (cadena vacía).

Editar con vi o ejecutar la sentencia pg sobre el fichero de parametros de inicializacion (init<SID>.ora). Indicar:

```
os_authent_prefix = ""
```

- 53.** Desbloquear la cuenta del usuario "prueba04".

```
SQL> alter user prueba03 account unlock;
```

- 54.** Modificar los valores del perfil DEFAULT según se indica en la siguiente tabla:

Sessions_per_user	5
Cpu_per_session	unlimited
Cpu_per_call	6000
Connect_time	480
Idle_time	60
Failed_login_attempts	3
Password_life_time	180

```
SQL> alter profile default
      limit
sessions_per_user 5
cpu_per_session unlimited
cpu_per_call 6000
connect_time 480
idle_time 60
failed_login_attempts 3
password_life_time 180;
Profile altered.
```

- 55.** Averiguar que usuarios o roles tienen asignado el privilegio "ALTER PROFILE".

```
SQL> select * from dba_sys_privs where privilege='ALTER PROFILE';
GRANTEE          PRIVILEGE          ADM
-----          DBA ALTER
PROFILE          YES
```

- 56.** Asignar el privilegio anterior al rol ADMIN.

```
SQL> grant alter profile to admin;
Grant succeeded.
```

57. Comprobar los valores asignados al perfil "DESARROLLO". Modificar el perfil "DESARROLLO", desde el usuario "administrador", según la siguiente tabla:

Sessions_per_user	5
Connect_time	DEFAULT
Idle_time	30

¿Qué ha sucedido con el resto de los parámetros?. ¿Coincide el valor de "Connect_time" en este perfil con el que tiene en el perfil DEFAULT?.

```
SQL> select profile, substr(resource_name,1,25) nombre_recurso,
substr(limit,1,20) limite from dba_profiles where profile = 'DESARROLLO';
```

PROFILE	NOMBRE_RECURSO	LIMITE
DESARROLLO	COMPOSITE_LIMIT	DEFAULT
DESARROLLO	SESSIONS_PER_USER	2
DESARROLLO	CPU_PER_SESSION	UNLIMITED
DESARROLLO	CPU_PER_CALL	6000
DESARROLLO	LOGICAL_READS_PER_SESSION	DEFAULT
DESARROLLO	LOGICAL_READS_PER_CALL	DEFAULT
DESARROLLO	IDLE_TIME	2
DESARROLLO	CONNECT_TIME	480
DESARROLLO	PRIVATE_SGA	DEFAULT
DESARROLLO	FAILED_LOGIN_ATTEMPTS	2
DESARROLLO	PASSWORD_LIFE_TIME	120
DESARROLLO	PASSWORD_REUSE_TIME	DEFAULT
DESARROLLO	PASSWORD_REUSE_MAX	DEFAULT
DESARROLLO	PASSWORD_VERIFY_FUNCTION	DEFAULT
DESARROLLO	PASSWORD_LOCK_TIME	DEFAULT
DESARROLLO	PASSWORD_GRACE_TIME	DEFAULT

16 rows selected.

```
SQL> alter profile desarrollo
  2  limit SESSIONS_PER_USER 5
  3  connect_time default
  4  idle_time 30;
  5  Profile altered.
```

58. Averiguar los privilegios de sistema y sobre objetos, así como los roles, que tiene asignados los roles por defecto "CONNECT", "RESOURCE", "DBA", "EXP_FULL_DATABASE" e "IMP_FULL_DATABASE".

¿Considera una buena política de seguridad asignar el rol "CONNECT" a todos los usuarios que precisan conectarse a la base de datos?.

```
SQL> select      *
  from      dba_role_privs
  where      grantee    in
('CONNECT','RESOURCE','DBA','EXP_FULL_DATABASE','IMP_FULL_DATABASE') order by
grantee, granted_role
```

GRANTEE	GRANTED_ROLE	ADM	DEF
DBA	DELETE_CATALOG_ROLE	YES	YES
DBA	EXECUTE_CATALOG_ROLE	YES	YES
DBA	EXP_FULL_DATABASE	NO	YES
DBA	GATHER_SYSTEM_STATISTICS	NO	YES
DBA	IMP_FULL_DATABASE	NO	YES
DBA	SELECT_CATALOG_ROLE	YES	YES
EXP_FULL_DATABASE	EXECUTE_CATALOG_ROLE	NO	YES
EXP_FULL_DATABASE	SELECT_CATALOG_ROLE	NO	YES
IMP_FULL_DATABASE	EXECUTE_CATALOG_ROLE	NO	YES
IMP_FULL_DATABASE	SELECT_CATALOG_ROLE	NO	YES

```
SQL> select      *      from      dba_sys_privs      where      grantee      in
('CONNECT','RESOURCE','DBA','EXP_FULL_DATABASE','IMP_FULL_DATABASE') order by
grantee,privilege;
```

GRANTEE	PRIVILEGE	ADM
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE DATABASE LINK	NO
CONNECT	CREATE SEQUENCE	NO
CONNECT	CREATE SESSION	NO
CONNECT	CREATE SYNONYM	NO
CONNECT	CREATE TABLE	NO
CONNECT	CREATE VIEW	NO
DBA	ADMINISTER DATABASE TRIGGER	YES
DBA	ADMINISTER RESOURCE MANAGER	YES
DBA		

SQL> select grantee, table_name, privilege from dba_tab_privs where grantee in ('CONNECT','RESOURCE','DBA','EXP_FULL_DATABASE','IMP_FULL_DATABASE') order by grantee, table_name, privilege

DBA	DBMS_DEFER_QUERY EXECUTE
DBA	DBMS_DEFER_SYS EXECUTE

59. ¿Puede asignarse el perfil “DESARROLLO” al rol “CONNECT”? . ¿Y el perfil “DEFAULT” al perfil “DESARROLLO”?:

No.

60. Averiguar que usuarios o roles de la base de datos tienen asignado el privilegio “DROP PROFILE”.

SQL> select * from dba_sys_privs where privilege='DROP PROFILE';

GRANTEE	PRIVILEGE	ADM
DBA	DROP PROFILE	YES
IMP_FULL_DATABASE	DROP PROFILE	NO

61. Asignar el privilegio “DROP PROFILE” al rol “ADMIN.”.

SQL> grant drop profile to admin;
Grant succeeded.

62. Conectarse como usuario “administrador” e intentar eliminar el perfil “DEFAULT”, ¿qué ocurre?.

SQL> connect administrador Enter
password:
Connected.

SQL> drop profile default;
drop profile default
* ERROR at
line 1:
ORA-00931: missing identifier

SQL> drop profile default cascade;
drop profile default cascade
* ERROR at
line 1:
ORA-00931: missing identifier

63. Como usuario “administrador” crear el rol “SECRETO” identificado por la contraseña “total” y asignarlo al usuario “prueba04”.

*SQL> connect administrador Enter
password: Connected.*

*SQL> create role secreto identified by total;
Role created.
SQL> grant secreto to prueba04;
Grant succeeded.*

64. Averiguar que usuarios poseen el privilegio “ALTER ANY ROLE” (de forma directa o a través de roles).

*SQL> select * from dba_sys_privs where privilege='ALTER ANY ROLE';
DBA ALTER ANY ROLE YES*

65. ¿Qué valor tiene en la base de datos el parámetro MAX_ENABLED_ROLES?. Modificar su valor para que, en adelante, valga 40. Comprobar esta modificación.

*SQL> show parameters max_enabled_roles
max_enabled_roles integer 30*

66. Averiguar que usuarios poseen el privilegio “GRANT ANY ROLE” (de forma directa o a través de roles).

*SQL> select * from dba_sys_privs where privilege='GRANT ANY ROLE';
DBA GRANT ANY ROLE YES*

67. Como usuario “administrador”, desasignar el rol “SECRETO” al usuario “prueba04”.

*SQL> revoke secreto from prueba04;
Revoke succeeded.*

68. Asignar el privilegio “GRANT ANY ROLE” al rol “ADMIN.” .

*SQL> connect system Enter
password: Connected.*

*SQL> grant grant any role to admin;
Grant succeeded.*

69. Averiguar de nuevo que usuarios poseen el privilegio “GRANT ANY ROLE” (de forma directa o a través de roles).

```
SQL> select * from dba_sys_privs where privilege='GRANT ANY ROLE';
```

ADMIN	GRANT ANY ROLE	NO
DBA	GRANT ANY ROLE	YES

70. Averiguar que usuarios poseen el privilegio “DROP ANY ROLE” (de forma directa o a través de roles).

```
SQL> select * from dba_sys_privs where privilege='DROP ANY ROLE';
```

DBA	DROP ANY ROLE	YES
IMP_FULL_DATABASE	DROP ANY ROLE	NO

71. Asignar permiso de conexión al usuario "prueba03", asignar el rol "SECRETO" al mismo usuario. Conectarse como este usuario e intentar borrar el rol.

```
SQL> connect system Enter
password: Connected.
```

```
SQL> grant conexion to prueba03; Grant
succeeded.
```

```
SQL> grant secreto to prueba03; Grant
succeeded.
```

```
SQL> connect prueba03 Enter
password: Connected.
```

```
SQL> drop role secreto; drop role
secreto
*
```

```
ERROR at line 1:
ORA-01031: insufficient privileges
```

72. En caso de que no lo tenga asignado, asignar el rol “CONEXION” y el rol “DESARROLLO” al usuario “prueba04”. Hacer que solo el rol “CONEXIÓN” este activo cuando se conecte.

```
SQL> select * from dba_role_privs where grantee='PRUEBA04'; GRANTEE
GRANTED_ROLE          ADM      DEF
```

PRUEBA04	CONEXION	NO	YES
PRUEBA04	DESARROLLO	NO	YES

```
SQL> alter user prueba04 default role conexion; User altered.
```

73. Comprobar en la vista apropiada del diccionario de datos los roles activos en la sesión.

```
SQL> select * from dba_role_privs where grantee='PRUEBA04';
```

GRANTEE	GRANTED_ROLE	ADM	DEF
PRUEBA04	CONEXION	NO	YES
PRUEBA04	DESARROLLO	NO	NO

74. Conectado como usuario "prueba04", activar el rol "DESARROLLO" y comprobar de nuevo en la vista apropiada del diccionario de datos los roles activos en la sesión.

```
SQL> connect prueba04 Enter
password: Connected.
```

```
SQL> select * from session_roles;
```

```
ROLE
```

```
-----
```

```
CONEXION
```

```
SQL> set role all; Role set.
```

```
SQL> select * from session_roles;
```

```
ROLE
```

```
----- CONEXION
```

```
DESARROLLO
```

75. Asignar el rol "CONNECT" al usuario "ADMIN". ¿Es preciso asignarle los permisos "CREATE PROCEDURE", "CREATE PUBLIC SYNONYM", "CREATE ROLE", "CREATE TRIGGER"?; ¿Los tiene ya asignados?.

```
SQL> grant connect to admin;
```

```
Grant succeeded.
```

```
SQL> select * from dba_sys_privs where grantee='CONNECT';
```

GRANTEE	PRIVILEGE	ADM
CONNECT	CREATE VIEW	NO
CONNECT	CREATE TABLE	NO
CONNECT	ALTER SESSION	NO
CONNECT	CREATE CLUSTER	NO
CONNECT	CREATE SESSION	NO
CONNECT	CREATE SYNONYM	NO
CONNECT	CREATE SEQUENCE	NO
CONNECT	CREATE DATABASE LINK	NO

76. Conectarse como usuario SYSTEM y otorgar al usuario "prueba02" el permiso para seleccionar datos de la tabla códigos (pertenece al usuario "prueba01").
¿Qué sucede?, ¿por qué?.

SQL> show user USER es "SYSTEM"

*SQL> grant select on prueba01.codigos to prueba02;
grant select on prueba01.codigos to prueba02
* ERROR en línea 1:*

ORA-01031: privilegios insuficientes

77. Conectarse como usuario "prueba01" y otorgar al usuario "prueba02" el permiso para seleccionar datos de la tabla códigos; hacerlo de forma que "prueba02" también pueda otorgar el permiso a otros usuarios (opción ADMIN).

*SQL> connect prueba01
Introduzca su clave: Connected.*

SQL> grant select on prueba01.codigos to prueba02 with grant option;

Grant succeeded.

78. Conectarse como usuario "prueba02" y otorgar al usuario "prueba03" el permiso para seleccionar datos de la tabla códigos.

*SQL> connect prueba02 Enter
password: Connected.*

*SQL> grant select on prueba01.codigos to prueba03 ;
Grant succeeded.*

*SQL> connect prueba03 Enter
password: Connected.*

*SQL> select * from prueba01.codigos;
no rows selected*

79. Conectarse como usuario "prueba01" y revocar al usuario "prueba02" el permiso para seleccionar datos de la tabla códigos.

*SQL> connect prueba01 Enter
password: Connected.*

*SQL> revoke select on prueba01.codigos from prueba02;
Revoke succeeded.*

80. Conectarse como usuario "prueba03" e intentar consultar la tabla códigos.
¿Qué ocurre?, ¿por qué?.

*SQL> connect prueba03 Enter
password: Connected.*

*SQL> select * from prueba01.codigos; select * from
prueba01.codigos
* ERROR at line 1:
ORA-00942: table or view does not exist*

6.3.2. Prácticas PROPUESTAS

Para todas y cada una de las operaciones propuestas se debe probar y demostrar su funcionalidad y efectividad, para ello puede ser necesario crear y manipular objetos (BBDD, tablas, etc).

6.3.2.1. Práctica – A

- Usando los comandos *CREATE USER*, *DROP USER*, ...:
 - Dar de alta al usuario *ALUMNO*.
 - Dar de baja *ALUMNO*.

6.3.2.2. Práctica – B

Realizar las siguientes operaciones:

- Crear un **TABLESPACE** denominado *TAREA4* de un 1MB de tamaño que debe estar contenida en dos ficheros, y será el espacio de almacenamiento para todos los usuarios que se creen en esta tarea.
 - Crea el usuario *AD1DAM* como ABD (debe tener todos los privilegios) y los podrá otorgar a otros usuarios.
 - A través del usuario *AD1DAM*:
 - Crear los usuarios: *U1DAM_A*, *U1DAM_B*, *U1DAM_C*.
 - A través del usuario *U1DAM_A*, el usuario *U1DAM_B*, podrá:
 - Crear la tabla *U1DAM_A.TA* (*uno* , *dos* , *tres*) y realizar todas las operaciones posibles sobre ella.
 - En la tabla *U1DAM_A.TB* (*x* , *y* , *z*):
 - Insertar valores.
 - Modificar los valores del atributo *y*.
 - Podrá manipular (I-B-M-C) la tabla *U1DAM_A.TC* (*a* , *b* , *c*). Estas operaciones las podrá otorgar a otros usuarios.
 - El usuario *U1DAM_B* permite al usuario *U1DAM_C* sobre la tabla *TC*; Insertar valores en la tabla pero solo para los atributos *a* , *b*.
 - El usuario *U1DAM_B* permite al usuario *U1DAM_C* consultar la tabla *TC*.
 - El usuario *U1DAM_A* retira los permisos que tiene *U1DAM_B* sobre la tabla *TC*; ¿el usuario *U1DAM_C* puede consultar el contenido de *TC*?
- Si ya no recordases las operaciones anteriores, ¿Cómo averiguarías los permisos que tiene *U1DAM_A*, sobre la tabla *TA*?

6.3.2.3. Práctica – C

- CREAR UN USUARIO (**USUA**):
- DARLE SOLO PERMISOS PARA:
 - Conectarse.
 - Insertar en una determinada tabla cualquiera.
 - Consultar datos en cualquier tabla.
 - QUITAR EL PRIVILEGIO DE INSERTAR.
- CREAR UN USUARIO (**USUB**):
- DARLE EL ROL *connect*.
 - Conseguir que otro usuario le dé y quite privilegios sobre un objeto.

- Darle permisos sobre un objeto y que este se los pueda pasar a otro usuario. Comprobar que si retira los permisos el primero, los pierden los otros dos usuarios.
- QUITAR TODOS LOS PRIVILEGIOS Y ROLES AL USUARIO (A).
 - CREAR UN ROL QUE SOLO PERMITA CONECTARSE Y VER EL CONTENIDO DE UNA TABLA DE UN DETERMINADO USUARIO. DAR ESE ROL AL USUARIO (A), Y COMPROBAR QUE FUNCIONA.

6.3.2.4. Práctica – D

- CREAR el Usuario DAM1:
 - Cuota de 500K.
 - Privilegios:
 - Conectarse.
 - Crear sus propias tablas: Probar si tiene alguna limitación al operar con ellas (I-B-M-C-...)
- CREAR el Usuario DAM2:
 - Sin Cuota.
- El usuario DAM1 gestiona permisos sobre DAM2:
 - I-B-M-C-... en una de sus tablas (T1).
 - I-B-M-C en un atributo de una de sus tablas (T2).
 - Le retira algunos de los permisos concedidos sobre T1 y T2.
- Al usuario DAM1 se le concede el permiso de CREAR VISTAS en cualquier esquema para que se lo pase a DAM2, este a su vez debe poder conceder el permiso de insertar valores en T1 el a un tercer usuario DAM3.

IMPORTANTE:

- Se debe probar el funcionamiento de cada una de las operaciones, y comprobar que han sido reflejadas en el DD.
- Todas las operaciones deben ser guardadas y comentadas en un fichero PRACTICA-1 de ADM.SQL.

6.3.2.5. Práctica – E

Se desea gestionar los accesos de un grupo de usuarios organizados en tres categorías: Directivos, Jefes y Empleados. Cada una de las categorías tendrá diferentes grupos de privilegios; los más privilegiados serán los Directivos y los menos los Empleados. Para ello debes:

- Organizar el espacio de almacenamiento lo más detallado posible.
- Crear varios usuarios para cada uno de los grupos limitando y controlando su operatividad.
- Otorgarles los permisos que consideres oportunos de la forma más eficiente posible.

Realizar algunas modificaciones posteriores, tales como:

- Modificar cuota y los límites de algún usuario.
- Quitar permisos a algún usuario y grupo.
- Cambiar el espacio de almacenamiento.

IMPORTANTE: Despues de cada operación se deben comprobar las correspondientes tablas del DD implicadas en la operación, así como la efectividad de la operación.