UD 05 - TRATAMIENTO DE DATOS

1. Introducción

Para el tratamiento de datos pueden usarse instrucciones SQL (flexibilidad, rapidez, mayor detalle) o herramientas gráficas que permiten hacerlo de forma más visual pero con menos detalle.

2. Edición de información mediante herramientas gráficas

Hay herramientas gráficas y herramientas en modo texto (terminal, consola, línea de comandos: para esto usaremos SQL).

Oracle
Oracle Database Express con Application Express (Base de datos Oracle 11g Express Edition > Get started)

MySQL
MySQL Community Server dispone de Workbench para trabajar en modo gráfico. También está PhpMyAdmin y Navicat.

2.1. Inserción de registros

Oracle
Basicamente se pulsa en Datos e Insertar fila. Se escriben los datos. Se pulsa en crear. En caso de que se produzca un error podrá haber un error con un campo de tipo numérico.
MySQL
Lo mismo.. Símbolo +, Apply...

2.2. Modificación de registros

Seleccionar la fila, modificarla, Apply... Si se han modificado correctamente mostrará mensaje de éxito y de error en caso contrario.

2.3. Borrado de registros

Oracle: Elegir Datos y hacer click en Editar. Darle a suprimir. Darle a Aceptar para confirmar supresión. Puede ser que no deje eliminar por una restricción de integridad. Debe eliminarse el registro que hace referencia a él o modificarlo para que haga referencia a otro registro.
MySQL: En el menú contextual se pulsa "Select Rows", aparecen las filas o registros con botones de insertar, modificar, eliminar. Se sitúa el cursor en la fila que se quiere eliminar. Se selecciona el botón con (-) que elimina una fila. Se hace click en Apply para aplicar los cambios. Sale mensaje de éxito o de error.
Antes de aplicar o confirmar la operación MySQL muestra el SQL correspondiente a la sentencia que se va a ejecutar.

3. Edición de la información mediante sentencias SQL

Oracle: Apliquémoslas desde Application Express de Oracle (entorno web) usando el botón SQL Workshow y pulsando SQL commands.
MySQL: Desde la ventana SQL de workbench o desde la línea de comandos o cliente en modo texto.
MySQL Workbench tiene activa la opción SQL_SAFE_UPDATEs que no permite ejecutar sentencias de actualización y borrado. Para ejecutar las sentencias deberá desactivarse esta opción en EDIT, PREFERENCES y en SQL Editor desmarcar Safe Updates.

3.1. Inserción de registros

La sentencia INSERT se puede hacer en dos formas básicas:

- Añadir una fila indicando los valores que deben tomar las columnas
- Extraer las filas de una tabla ya existente y añadirlas a otra tabla (Consultas SELECT con INSERT)

```
INSERT INTO nombre_tabla (lista_campos) VALUES (lista_valores);
```

- Puede hacerse el INSERT con la lista de campos y la lista de valores
- Si se hace un INSERT en el que no se especifiquen los valores de todos los campos se obtiene NULL en los campos que no se han indicado
- Es posible omitir la lista de campos si se indican todos los valores de cada campo y en el orden en el que se encuentra definidos en la tabla.

```
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES
('AS0015','Julia', 'Hernández','Sáez','M',
'11/10/1992','BK Programación');

INSERT INTO departamento VALUES ('10','I+D', 'Sevilla');

INSERT INTO departamento (cddep,nombre) VALUES ('11','Ventas');
```

- Si la lista de campos no se corresponde con la lista de valores se obtiene un error en la ejecución. (Por ej.: ORA-00913: demasiados valores)

Insert extendido

Consiste en insertar varias filas con una única sentencia INSERT.

```
INSERT INTO departamento VALUES ('20','DEPAR20','ALMERÍA'),('21','DEPART21','ALMERIA'),
('22','DEPART22','GRANADA');
```

Para los casos en los que sean valores AUTONUMÉRICOS se puede:

- Omitir el nombre de la columna autonomérica
- Poner un 0 y el sistema calculará automáticamente el valor

```
INSERT INTO alumnos (nombre, apellidos, idcurso)
VALUES('Daniel', 'Ro',2);

INSERT INTO alumnos (idalumnos, nombre, apellidos, idcurso) VALUES (0,'Manuel','Bellido',
2);
```

Insert con replace

Si el valor de la clave primaria a insertar coincide con el existente lo borra para inserrtar el nuevo

```
REPLACE INTO departamento VALUES (‘09’, ‘InformáticaReplace’, ‘Almería’)
```

3.2. Modificación de registros

La estructura de UPDATE es:

```
UPDATE nombre_tabla SET nombre_campo = valor [, nombre_ campo = valor]...
[ WHERE condición ];
```

Se puede especificar los nombres de los campos que se deseen de la tabla indicada. A cada campo se le debe asociar el nuevo valor usando el signo =. Cada emparejamiento de campo=valor debe separarse usando comas.

La cláusula WHERE seguida de la condición es opcional para indicar que solo afecte a los registros que cumplan esa condición.

```
UPDATE CONFERENCIA SET precio = 20;
UPDATE CONFERENCIA SET precio = 20, tema = NULL;
UPDATE CONFERENCIA SET Credito = 30 WHERE turno = 'T';
```

Al terminar la sentencia UPDATE se muestra la cantidad de registros actualizados o el error si lo hubiese.

3.3. Borrado de registros

La estructura es la siguiente:

```
DELETE FROM nombre_tabla [ WHERE condición ];
```

Ejemplos:

```
DELETE FROM PONENTE WHERE especialidad = 'Programación';
DELETE FROM empleado  WHERE cddep='22';
```

Igualmente devuelve el mensaje de error o el número de filas actualizadas.

5. Subconsultas y composiciones en órdenes de edición

5.1. Inserción de registros a partir de una consulta

```
INSERT [INTO] <tabla> [(<columna>,...)]  SELECT ...
```

La inserción de los registros se puede hacer también usando una sentencia SELECT:

```
-- Este INSERT
INSERT INTO ASISTENTE (codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa) VALUES
('AS0015','Julia', 'Hernández','Sáez','M', '11/10/1992','BK Programación');

-- Puede ser equivalente a este
INSERT INTO (SELECT codigo, nombre, apellido1, apellido2, sexo, fechaNac, empresa FROM
ASISTENTE VALUES ('AS0015','Julia', 'Hernández','Sáez','M', '11/10/1992','BK Programación');
```

Es posible insertar en la tabla valores que se obtienen directamente del resultado de una consulta si tienen la misma estructura que la tabla asistente. **No es necesario usar la palabra VALUES**

```
INSERT INTO ASISTENTES_SUPLENTES SELECT * FROM ASISTENTE WHERE empresa = null;
```

Se puede hacer esto, por ejemplo. Fíjate que hacemos una consulta y estamos insertando el cdemp y el 'ECS' por defecto:

```
INSERT INTO trabaja (cdemp,cdpro) SELECT cdemp,'ECS' FROM empleado WHERE   cddep = '03';

INSERT INTO trabaja (cdemp,cdpro) SELECT cdemp,'ECS' FROM empleado WHERE  cdemp NOT IN
(SELECT DISTINCT cdemp FROM trabaja);
```

5.2. Modificación de registros a partir de una consulta

La actualización también puede hacerse con consultas para realizar modificaciones más complejas de los datos.

Modifica el precio de las conferencias que se celebren en salas cuya capacidad sea mayor que 200. El precio que se les asignará será el de la conferencia que tenga mayor precio.

```
UPDATE CONFERENCIA SET precio = (SELECT MAX(precio) FROM CONFERENCIA) WHERE sala IN <br />
(SELECT nombre FROM SALA WHERE capacidad >200);<br />
```

Si queremos asignar a los empleados sin departamento, al departamento o uno de los departamentos en cuyos proyectos se ha trabajado cero horas, la sentencia sería:

```
UPDATE empleado
SET cddep = (SELECT p.cddep FROM proyecto p
                        INNER JOIN trabaja t ON t.cdpro=p.cdpro
                        GROUP BY t.cdpro
                        HAVING SUM(t.nhoras)=0
                         ORDER By cddep<br />
                          LIMIT 1)
                          WHERE cddep IS NULL;
```

En caso de que para poder realizar la actualización de una tabla necesitemos consultar la misma tabla que se está actualizando, MySQL daría un error ya que la tabla que se está actualizando estaría bloqueada y no permitiría realizar consultas.

Se puede creando una **tabla temporal** de la tabla que se quiere usar.

```
-- Esta falla
UPDATE EMPLEADO SET SALARIO=SALARIO+100 WHERE SALARIO <= (SELECT AVG(SALARIO) FROM
EMPLEADO);

--- Asi si
UPDATE EMPLEADO SET SALARIO=SALARIO+10 WHERE SALARIO <= (SELECT AVG(SALARIO) FROM (SELECT *
FROM EMPLEADO) AS EMP);
```

5.3. Supresión de registros a partir de una consulta

```
DELETE FROM (SELECT * FROM ASISTENTE WHERE Empresa='Bigsoft' AND sexo='H');

DELETE FROM (SELECT * FROM ASISTENTE WHERE Empresa='Bigsoft') WHERE sexo='H';

DELETE FROM proyecto WHERE cdpro NOT IN (SELECT cdpro FROM trabaja);

DELETE FROM empleado
WHERE cdemp IN (SELECT t.cdemp FROM trabaja t
INNER JOIN proyecto p ON p.cdpro=t.cdpro
WHERE p.cddep='02');
```

Igualmente, no puedes hacer subconsulta sobre la tabla en la que se van a borrar registros. Debe usarse **tabla temporal.**

```
DELETE FROM  EMPLEADO  WHERE SALARIO >
(SELECT AVG(SALARIO) FROM (SELECT * FROM EMPLEADO) AS EMP1);
```

5. Integridad referencial

Dos tablas están relacionadas si tienen en común uno o más campos llamados **clave ajena** o **clave foránea**

La restricción de integridad referencial requiere que haya coincidencia en todos los valores que tengan en común ambas tablas. Cada valor del campo que forma parte de la integridad referencial definida debe corresponder en la otra tabla con otro registro que contenga el mismo valor en el campo referenciado.

Ejemplo: PARTIDAS con FK que apunta a JUEGOS. No puede existir ninguna partida cuyo código de juego no se corresponda con los juegos de la tabla JUEGOS.

Hablaremos de:

- Clave ajena: Campo o conjunto de campos incluidos en la definición de la restricción que deben hacer referencia a una clave de referencia
- Clave de referencia: Clave única o primaria de la tabla a la que se hace referencia desde clave ajena.
- Tabla hija o dependiente: Tabla que incluye la clave ajena y depende de los valores de la clave de referencia
- Tabla padre o de referencia: Tabla que es referenciada por la clave ajena en la tabla hija. Determina las inserciones o actualizaciones que son permitidas en la tabla hija.

Motores de almacenamiento MySQL

- **Motor MyISAM**: No gestiona integridad referencial, ni transacciones. Los datos se almacenan en un formato independiente que permite pasar tablas entre distintas plataformas. Útiles para datos estables que solo tienen operaciones de lectura.
- **Motor InnoDB**: Gestiona integridad referencial con claves foráneas, soporta control de transacciones y bloqueo por registro. Para lograr esto se sacrifica un poco la velodiad de MySQL. Las tablas MyISAM se bloquean al realizar inserción o actualización en un registro. En las tablas InnoDB el bloqueo solo se hace en el registro que se esté modificando para que más usuarios usen de forma simultánea la base de datos.

5.1. Integridad en actuación y supresión de registros

- Si se modifica el valor de la clave ajena en la tabla hija debe ser un valor que haga referencia a la clave principal de uno de los registros de la tabla padre.
- No se puede modificar el valor de la clave principal de un registro si una clave ajena hace referencia a dicho registro.
- No pueden suprimirse registros que son referenciados con clave ajena desde otra tabla.

Cuando se hace borrado de registros en tabla de referencia se puede configurar la clave ajena para que se conserve la integridad referencial:

- RESTRICT **No permitir supresión**: Opción por defecto. No permite borrar en la tabla de referencia un registro que está siendo referenciado desde otra tabla. Produce error en la operación de borrado.
- CASCADE **Supresión en cascada**: Los registros de la tabla hija que hacían referencia se borran también
- SET NULL **Definir nulo en suprimir**. Los valores de la clave ajena que hacen referencia a los registros que hayan sido borrados de la tabla de referencia son cambiados por NULL.
- NO ACTION No se hace nada en las otras

Modificaciones Actuará según lo indicando en `ON UPDATE (RESTRICT, NO ACTION, CASCADE, SET NULL`
Eliminaciones Actuará según lo indicado en `ON DELETE (RESTRICT, NO ACTION, CASCADE, SET NULL)`

5.2. Supresión en cascada

Las opciones de cascada / nulo se pueden establecer en la creación de tablas. En la pestaña de restricciones (crear, borrar, activar, desactivar restricciones)

En SQL:
`CONSTRAINT JUEGOS_CON FOREIGN KEY (Cod_Juego) REFERENCES JUEGO (Codigo) ON DELETE CASCADE`

6. Transacciones

Una transacción es una unidad atómica (indivisible) de trabajo que contiene una o más sentencias SQL. A todas ellas se les aplica una operación COMMIT o ROLLBACK.

Mientras que no se haga COMMIT los resultados pueden deshacerse. **El efecto de una sentencia DML no es permanente hasta no hacer COMMIT sobre la transacción en BBDD**

Propiedades básicas de las transacciones (ACID):

- **Atomicidad** (Atomicity): O se hacen todas o no se hace ninguna
- **Consistencia** (Consistency): Parte un estado consistente hasta otro estado consistente
- **Aislamiento** (Isolation): No es visible por otras transacciones hasta que no finaliza
- **Durabilidad** (Durability): Los cambios por transacciones que vuelvan sus modificaciones son permanentes.
  Las tablas que soportan transacciones son más seguras y fáciles de recuperar si se produce algún fallo. Aunque pueden aumentar el tiempo de proceso de instrucciones.

Las sentencias de control de transacciones gestionan las sentencias DML y las agrupan en transacciones. Se puede:

- Hacer permanentes los cambios `COMMIT`
- Deshacerlos `ROLLBACK` desde que fue iniciada o desde un punto de restauración (marcador que se puede establecer en el contexto de la trasancción `ROLLBACK TO SAVEPOINT`). Ojo ROLLBACK finaliza la transacción, ROLLBACK TO SAVEPOINT, no.
- Establecer punto de restauración `SAVEPOINT
- Indicar propiedades de la transacción `SET TRANSACTION`
- Especificar si una restricción se comprueba tras cada sentencia DML o cuando se realize el commit `SET CONSTRAINT`

6.1. Hacer cambios permanentes

La transacción comienza con la primera SQL ejecutable. Se puede

- Usar COMMIT
- Usar una sentencia DDL como CREATE, DROP, RENAME, ALTER (ejecuta un COMMIT antes y después de cada DDL)
- Cerrar adecuadamente las aplicaciones de gestión (vuelca la transacción)

Existe opción de "confirmación automática" (autocommit) en las aplicaciones gráficas.

Iniciar transacción con `START TRANSACTION` o `BEGIN`
Confirmar con `COMMIT` o deshacer con `ROLLBACK` (ante cancelación, excepción, fallo de sistema).

InnoDB siempre tiene actividad de usuario entre transacción. MySQL siempre comienza nueva conexión con la ejecución automática habilitada (AUTOCOMMIT = 1) Cada instrucción SQL se trata como transacción en sí misma y se confirma en cuanto la ejecución termina.
Puede verse el valor: `show variables like '%autocommit%';`
Se puede cambiar con `SET autocommit = OFF;`

Si la conexión tiene ejecución automática, igualmente el usuario puede llevar a cabo una transacción si lo hace con los comandos habituales:

```
START TRANSACTION;
UPDATE TRABAJA SET nhoras = nhoras + 50 WHERE cfpro='DAG';
COMMIT;
```

Instrucciones que implican COMMIT automático

DDL que definen o modifican objetos: CREATE, ALTER, DROP, RENAME, TRUNCATE
Tablas de la base de datos administrative MySQL: GRANT, REVOKE
Control de transacciones y bloqueo de tablas: START TRANSACTION, BEGIN, LOCK, UNLOCK
Carga de datos: LOAD DATA
Administración de tablas: ANALIZE, CHECK, OPTIMIZE, REPAIR

6.2. Deshacer cambios

Siempre se recomienda finalizar explícitamente las transacciones.

Veamos un ejemplito en Oracle:

```
CREATE TABLE emp_name AS SELECT employee_id, last_name FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);
CREATE TABLE emp_sal AS SELECT employee_id, salary FROM employees;
CREATE UNIQUE INDEX empsal_ix ON emp_sal (employee_id);
CREATE TABLE emp_job AS SELECT employee_id, job_id FROM employees;
CREATE UNIQUE INDEX empjobid_ix ON emp_job (employee_id);

DECLARE
   emp_id       NUMBER(6);
   emp_lastname VARCHAR2(25);
   emp_salary   NUMBER(8,2);
   emp_jobid    VARCHAR2(10);
BEGIN
   SELECT employee_id, last_name, salary, job_id INTO emp_id, emp_lastname,
     emp_salary, emp_jobid FROM employees WHERE employee_id = 120;
   INSERT INTO emp_name VALUES (emp_id, emp_lastname);
   INSERT INTO emp_sal VALUES (emp_id, emp_salary);
   INSERT INTO emp_job VALUES (emp_id, emp_jobid);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      ROLLBACK;
      DBMS_OUTPUT.PUT_LINE('Inserts have been rolled back');
END;
/
```

6.3. Deshacer cambios parcialmente

- SAVEPOINT id: Crea un SAVEPOINT.
- ROLLBACK TO SAVEPOINT id: ROLLBACK al SAVEPOINT
- RELEASE SAVEPOINT id: Elimina el SAVEPOINT creado

Cualquier operación COMMIT o ROLLBACK sin argumentos elimina todos los save points creados.

SAVEPOINT con ROLLBACK

```
CREATE TABLE emp_name AS SELECT employee_id, last_name, salary FROM employees;
CREATE UNIQUE INDEX empname_ix ON emp_name (employee_id);

DECLARE
   emp_id        employees.employee_id%TYPE;
   emp_lastname  employees.last_name%TYPE;
   emp_salary    employees.salary%TYPE;
BEGIN
   SELECT employee_id, last_name, salary INTO emp_id, emp_lastname,
     emp_salary FROM employees WHERE employee_id = 120;
   UPDATE emp_name SET salary = salary * 1.1 WHERE employee_id = emp_id;
   DELETE FROM emp_name WHERE employee_id = 130;
   SAVEPOINT do_insert;
   INSERT INTO emp_name VALUES (emp_id, emp_lastname, emp_salary);
EXCEPTION
   WHEN DUP_VAL_ON_INDEX THEN
      ROLLBACK TO do_insert;
      DBMS_OUTPUT.PUT_LINE('Insert has been rolled back');
END;
/
```

```
START TRANSACTION;
INSERT INTO proyectosx.departamento VALUES(‘88’,’depTrans1’, ‘ciudad1’);
SAVEPOINT sp1;
INSERT INTO proyectosx.departamento VALUES (‘90’, ‘depTrans2’, ‘ciudad2’), (‘91’,
‘depTrans3’, ‘ciudad3’);
ROLLBACK TO  SAVEPOINT sp1;
```

7. Problemas asociados al acceso simultáneo a los datos

En las bases de datos multiusuario las sentencias contenidas en varias transacciones simultáneas pueden actualizar los datos simultáneamente. Estas deben generar resultados consistentes.
Por tanto se debe asegurar:

- Concurrencia de datos: Los usuarios pueden acceder a los datos al mismo tiempo
- Consistencia de datos: Asegura que cada usuario tiene una vista consistente de los datos. Incluye los cambios visibles de las transacciones del mismo usuario y de otros.

Debe haber mecanismos (bloqueos) para prevenir modificación concurrente del mismo dato. Así se cumplen los requerimientos de:

- Consistencia: Los datos consultados por un usuario no son cambiados por otros hasta no finalizar la operación completa.
- Integridad: Los datos y sus estructuras deben reflejar todos los campos efectuados en orden correcto.

Los bloqueos se realizan de forma automática. No requieren la actuación del usuario en MySQL y Oracle.

7.1. Políticas de bloqueo

Bloquear una tabla o vista permite que nadie más pueda hacer uso de la misma de modo que tenemos la exclusividad de lectura y/o escritura sobre ella.

Los bloqueos afectan a la interacción de lectores (consultas) y escritores (modificaciones) del recurso.

- Un registro es bloqueado solo cuando es modificado por un escritor: Al actualizar la sentencia un registro, la transacción obtiene bloqueo solo para ese registro.
- Un escritor de un registro bloquea a otro escritor concurrente del mismo registro: Una transacción que modifica una fila impide que otra transacción modifique el mismo registro
- Un lector nunca bloquea a un escritor: La única excepción es `SELECT .. FOR UPDATE`
- Un escritor nunca bloquea a un lector. Mientras se modifica la BBDD proporciona al lector una vista del registro sin los cambios que se están realizando.

Existe:

- **Bloqueo pesimista**: Lo hace inmediatamente en cuanto el bloqueo se solicita. Garantiza que el registro es actualizado.
- **Bloqueo optimista**: El acceso al registro o tabla se cierra en el momento en el que los cambios a ese registro se actualizan en disco. Esto solo es apropiado cuando hay poca posibilidad de que alguien necesite acceder al registro mientras está bloqueado o no podremos estar seguro de que la actualización tenga éxito porque el intento de actualizar el registro puede dar error si otro usuario lo actualiza mientras.

Supongamos que un usuario está en proceso de modificación de un registro, y otro en ese mismo momento quiere leer ese mismo registro. ¿Qué tipo de bloqueo debes establecer para que el segundo usuario obtenga los datos con los cambios que está efectuando el primero? (Pesimista)

7.2. Bloqueos compartidos y exclusivos

Un registro solo puede obtener un bloqueo exclusivo pero puede tener varios bloqueos compartidos.

Hay dos tipos de bloqueos:

- **Bloqueo exclusivo**: Previene que sea compartido el recurso asociado. La transacción obtiene bloqueo exclusivo al modificar los datos. La primera transacción que bloquea un recurso es la que puede modificar hasta que el bloqueo exclusivo es liberado.
- **Bloqueo compartido**: Permite que sea compartido el recurso asociado, según la operación. Varios usuarios que leen datos pueden compartir los datos, realizando bloqueos compartidos

para evitar el acceso concurrente de un escritor que desea un bloqueo exclusivo. Varias transacciones obtiene bloqueos compartidos del mismo recurso.

Ejemplo: Una transacción que usa `SELECT ... FOR UPDATE` obtiene bloqueo exclusivo del registro (permite a otras sesiones modificar registros que no sean el bloqueado) y bloqueo compartido de la tabla (evita que otras sesiones modifiquen la estructura de la tabla).

En MySQL la política de bloqueos:

1. Ordena internamente las tablas a bloquear
2. Si una tabla debe bloquearse en lectura y escritura, hace la solicitud de bloqueo de escritura en primer lugar
3. Se bloquea cada tabla hasta que la sesión obtiene todos sus bloqueos (evita el deadlock o bloqueo mutuo) (ningun proceso podría obtener sus bloqueos)

En InnoDB los bloqueos se obtienen a nivel de fila. Así varios usuarios pueden bloquear varias filas simultáneamente.
Lo dicho también aplica para InnoDB:

- Bloqueo compartido: Permite lectura de filas. Varias transacciones adquieren bloqueos sobre las filas pero ninguna transacción puede modificarlas hasta que no se liberen
- Bloqueo exclusivo: Permite a la transacción bloquear filas para actualización o borrado. Las transacciones deben esperar a que se libere el bloqueo de las filas afectadas.

También hay bloqueo de múltiple granularidad: Una transacción indica que bloquea algunas filas de una tabla para lectura o escritura (intención de bloqueo) así MySQL puede gestionar conflictos y evitar deadlocks porque varias transacciones compartiran la reserva de la tabla ya que el bloqueo se da fila a fila en el momento de la modificación. Las dos filas reservan la misma tabla y solo cuando una de ellas modifique la tabla esta quedará bloqueada.

7.3. Bloqueos automáticos

Oracle y MySQL bloquean automáticamente recursos usados por transacción para prevenir que otras transacciones requieran acciones que impliquen usos exclusivos sobre el recurso. Se divide en:

- **Bloqueos DML**: Protegen los datos, garantizando la integridad. (sobre un registro o sobre la tabla completa) Ej.: Evitar que dos clientes compren en mismo artículo.
- **Bloqueos DDL**: Protegen la definición del esquema mientras operación DDL actúa sobre él. Los usuarios no pueden solicitar este bloqueo
- **Bloqueo del sistema**: Para proteger BBDD interna y estructuras de memoria

7.4. Bloqueos manuales

Pueden hacerse bloqueos manuales útiles para aplicaciones que requieren consistencia a nivel de transacciones o lecturas repetitivas o para aplicaciones que requieres que las transacciones tengan acceso exclusivo a un recurso sin esperar a que otras transacciones finalicen.

Oracle
La cancelación de bloqueos automáticos se puede hacer a nivel de sesión o de transacción.
Se puede a nivel de sesión establecer el nivel requerido de aislamiento de la transacción con `ALTER SESSION`

A nivel de transacción se omite el bloqueo por defecto si las sentencias incluyen:

- `SET TRANSACTION ISOLATION LEVEL`
- `LOCK TABLE`
- `SELECT .. FOR UPDATE`
  Los bloqueos finalizan una vez que la transacción finaliza.

MySQL
Los clientes pueden bloquear tablas para evitar que puedan modificar datos que se necesiten en la sesión con `LOCK`:

`LOCK TABLES >nombre_tabla [[AS] alias] tipo_bloqueo [, nombre_tabla [[AS] alias] tipo_bloqueo READ [LOCAL]  | [LOW_PRIORITY]  WRITE`

Para desbloquear: `UNLOCK TABLES`

Tipos de bloqueo
`READ`
La sesión o cliente puede leer pero ni él ni nadie puede escribir. Lo pueden adquirir varios a la vez. Permite que cualquiera pueda leer. LOCAL permite que haya inserciones concurrentes mientras dura el bloqueo.

`WRITE`
La sesión o cliente puede leer y escribir pero nadie más puede acceder a ella ni bloquear.

Los bloqueos de escritura tienen prioridad sobre los de lectura. El modificador LOW PRIORITY permite que los bloqueos de lectura se adquieran antes que los de escritura (De hecho escritura se adquirirá cuando no queden bloqueos lectura pendientes):

```
USE proyectosx;
LOCK TABLES departamento READ
SELECT  * FROM departamento;  //realiza la lectura y muestra los departamentos.
UPDATE departamento    //da error, pues la tabla está bloqueada para lectura
SET ciudad="ciudad" WHERE nombre='depar20';
UNLOCK TABLES;       //desbloquea la tabla
```