

## ÍNDICE

<b>1. MANIPULACIÓN DE DATOS .....</b>	<b>2</b>
1.1. INSERCIÓN DE DATOS (INSERT INTO) .....	2
ACTIVIDAD. INSERCIÓN DE DATOS.....	3
1.2. BORRADO DE DATOS (DELETE) .....	4
ACTIVIDAD. BORRADO DE DATOS.....	4
1.3. MODIFICACIÓN DE DATOS (UPDATE) .....	5
ACTIVIDAD. ACTUALIZACIÓN DE DATOS.....	5
<b>2. MANIPULACIÓN DE DATOS UTILIZANDO CONSULTAS .....</b>	<b>6</b>
2.1. INSERCIÓN DE DATOS CON CONSULTAS .....	6
ACTIVIDAD. INSERCIÓN DE DATOS CON CONSULTAS .....	7
ACTIVIDAD. INSERCIÓN DE DATOS CON CONSULTAS Y CONSTANTES.....	8
2.2. BORRADO DE DATOS DE DATOS CON CONSULTAS .....	9
ACTIVIDAD. BORRADO DE DATOS CON CONSULTAS .....	9
2.3. MODIFICACIÓN DE DATOS CON CONSULTAS.....	10
ACTIVIDAD. ACTUALIZACIÓN DE DATOS CON CONSULTAS .....	10
<b>3. CREACIÓN DE TABLAS CON CONSULTAS .....</b>	<b>11</b>
ACTIVIDAD. CREACIÓN DE TABLAS CON CONSULTAS .....	11
ACTIVIDAD. COMPARACIÓN ENTRE VISTAS Y TABLAS CREADAS CON CONSULTAS .....	12
<b>4. TRANSACCIONES .....</b>	<b>13</b>
ACTIVIDAD. TRANSACCIONES.....	14
▪ COMMIT IMPLÍCITO Y EXPLÍCITO .....	15
▪ ROLLBACK AUTOMÁTICO .....	15

En este documento, a no ser que se indique lo contrario, se reflejan los comandos SQL referidos al Sistema Gestos de Base de Datos (SGBD) Oracle Database.

## 1. Manipulación de datos

### 1.1. Inserción de datos (INSERT INTO)

Para realizar inserciones de datos en una base de datos relacional se emplea el comando **INSERT INTO**. Este comando forma parte del lenguaje de manipulación de datos (LMD).

La sintaxis básica es la siguiente:

Opcional: lista de columnas

```
INSERT INTO nombreTabla [(columna, [columna] ...)]
VALUES (valor, [valor] ...);
```

Lista de valores que se asignarán por orden a las columnas

Si no se especifican las columnas se consideran todas las columnas de la tabla, en orden.

La utilidad de especificar las columnas radica en poder no dar valor a algunas columnas (siempre que no sean obligatorias) o en cambiar el orden de los datos.

**Ejemplos:** Se tiene la tabla PROFESOR

Describe profesor;

Nombre	¿Nulo?	Tipo
COD_CENTRO		NUMBER(4)
DNI	NOT NULL	VARCHAR2(9)
ESPECIALIDAD		VARCHAR2(20)
NOMBRECOMPLETO		VARCHAR2(30)

Se indican valores para todos los campos de la tabla, en orden

```
insert into profesor
values (10, '88888888N', 'Ruiz Castillo, Juan', 'FÍSICA');
```

Se indican valores para todos los campos de la tabla, en distinto orden

```
insert into profesor (cod_centro, dni, especialidad, nombrecompleto)
values (10, '88888888N', 'FÍSICA', 'Ruiz Castillo, Juan');
```

Se indican valores para tres columnas de la tabla

```
insert into profesor (cod_centro, dni, especialidad)
values (10, '88888888N', 'FÍSICA');
```

Se indican valores para tres columnas de la tabla. A "nombreCompleto" se le asigna el valor null (vacío). Recuerda que si se utiliza "default" en CREATE TABLE, este formato no asignará el valor por defecto definido.

```
insert into profesor
values (10, '88888888N', null, 'FÍSICA');
```

Consideraciones:

- ▶ **Las inserciones son siempre de un registro completo**
- ▶ La asignación de valores a columnas es posicional (a la primera columna se le asigna el primer valor, etc)
- ▶ Los valores que se dan deben coincidir con el tipo definido para la columna (varchar2, date...)
- ▶ Los valores de textos y fechas deben ir entrecomillados con comillas simples ('')

### Actividad. Inserción de datos

En la tabla PROFESOR, inserta dos profesores con los comandos:

```
insert into profesor (dni, especialidad, nombrecompleto)
values ('11111111A', 'Ciencias Sociales', 'Elisa Sánchez');

insert into profesor values (null, '22222222A', 'Literatura', 'Erika Jiménez');
```

Observa, mediante un SELECT, qué código de centro tiene cada profesor en cada caso.

Existe la posibilidad de realizar inserciones múltiples mediante un único comando:

#### INSERT ALL

```
INTO nombreTabla [(columna, [columna] ...)] VALUES (valor, [valor] ...)
INTO nombreTabla [(columna, [columna] ...)] VALUES (valor, [valor] ...)
...
select * from dual;
```

Ejemplo:

```
insert all
into profesor(cod_centro,dni,especialidad) values (10, '77777777N', 'FÍSICA')
into profesor(cod_centro,dni,especialidad) values (10, '88888888N', 'FÍSICA')
into profesor(cod_centro,dni,especialidad) values (10, '99999999N', 'FÍSICA')
select * from dual;
```


Se indican valores para tres columnas de la tabla

## 1.2. Borrado de datos (DELETE)

Para realizar borrados de datos en una base de datos relacional se emplea el comando **DELETE**. Este comando forma parte del lenguaje de manipulación de datos (LMD).

La sintaxis básica es la siguiente:

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



Condición que deben cumplir los registros para ser borrados

Consideraciones:



### Los borrados son siempre de registros completos

- ▶ Se borran los registros que cumplen la condición especificada en la cláusula WHERE. **Si se omite esa cláusula, no hay condiciones y se borra todo el contenido de la tabla.**
- ▶ Este comando no modifica la estructura de la tabla, solo borra registros.

### Actividad. Borrado de datos

- En la tabla CENTROS, borra el centro con código 50.

### 1.3. Modificación de datos (UPDATE)

Para realizar actualizaciones de datos en una base de datos relacional se emplea el comando **UPDATE**. Este comando forma parte del lenguaje de manipulación de datos (LMD).

La sintaxis básica es la siguiente:

```
UPDATE nombreTabla SET columna1=valor1, columna2=valor2, ...  
[WHERE condición];
```

Condición que deben cumplir los registros para ser actualizados

Consideraciones:

- ▶ La actualización se produce en columnas concretas de registros
- ▶ Se actualizan los registros que cumplen la condición especificada en la cláusula WHERE. **Si se omite esa cláusula, no hay condiciones y se actualizan todos los registros de la tabla.**
- ▶ Este comando no modifica la estructura de la tabla, solo modifica valores.

Ejemplos:

Se cambiará el valor de 2 columnas en un registro

```
update centros set direccion='C/Pilón 13',num_plazas=295  
where cod_centro=22;
```

Se cambiará el valor de 2 columnas en todos los registros

```
update centros set direccion='C/Pilón 13',num_plazas=295;
```

Aumenta en 10 el número de plazas de todos los centros

```
update centros set num_plazas=num_plazas+10;
```

#### Actividad. Actualización de datos

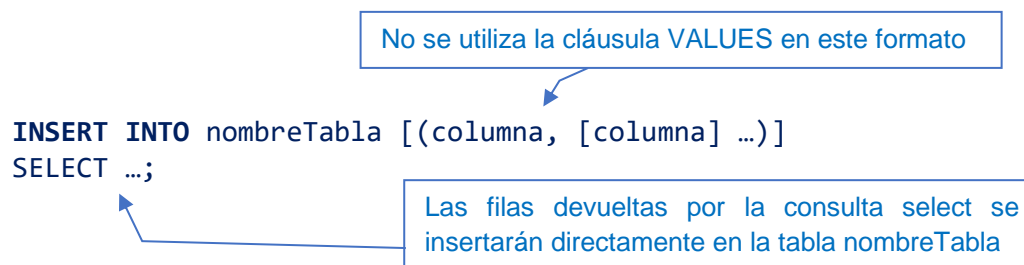
En la tabla EMPLE, a todos los empleados del departamento 30 aumentales el salario en 100 euros y la comisión en un 10%. Emplea un único comando.  
Observa qué ocurre con los empleados que no tienen comisión.

## 2. Manipulación de datos utilizando consultas

### 2.1. Inserción de datos con consultas

Es posible realizar inserciones de datos tomando como datos de entrada el resultado de una consulta.

La sintaxis básica es la siguiente:



Consideraciones:

- ▶ El listado devuelto por **SELECT** debe coincidir en cantidad de columnas y tipo de columnas a las columnas que inserta **INSERT**.
- ▶ La inserción y la consulta pueden realizarse a distintas tablas.
- ▶ El comando **SELECT** puede ser tan complejo como se necesite.

**Ejemplos:** Se tiene la tabla EMPLE y se crea una nueva tabla EMPLE\_SINCOMISION con la misma estructura que EMPLE. En esta nueva tabla se desean guardar los empleados que no tienen comisión. Podría hacerse con el comando:

```

insert into emple_sincomision
select * from emple where comision is null;

```

TABLA EMPLE\_SINCOMISION:

EMP_NO	APELLIDO	OFICIO	DIR	FECHA_ALT	SALARIO	COMISION	DEPT_NO
8000	MARTÍNEZ	EMPLEADOR	7902	27/01/15	1300	(null)	(null)
8001	PALOMERA	VENDEDOR	7902	08/05/17	1505	(null)	(null)
7839	REY	PRESIDENTE	(null)	17/11/19	4100	(null)	10
7566	JIMENEZ	DIRECTOR	7839	02/04/20	2900	(null)	20
7902	FERNANDEZ	ANALISTA	7566	03/12/20	3000	(null)	20
7369	SANCHEZ	DESARROLLADOR	7902	17/12/19	1040	(null)	20
7698	NEGRO	DIRECTOR	7839	01/05/20	3105	(null)	30
7782	CEREZO	DIRECTOR	7839	09/06/21	2885	(null)	10
7788	GIL	ANALISTA	7566	09/11/20	3000	(null)	20
7876	ALONSO	DESARROLLADOR	7788	23/09/20	1430	(null)	20
7900	JIMENO	DESARROLLADOR	7698	03/12/21	1435	(null)	30
7934	MUÑOZ	DESARROLLADOR	7782	23/01/20	1690	(null)	10

- Se tiene la tabla EMPLE y se crea una nueva tabla EMPLE\_DEP20 para almacenar los empleados del departamento 20. La estructura de esta tabla no es idéntica a la de EMPLE:

TABLA EMPLE:

Nombre	¿Nulo?	Tipo
EMP_NO	NOT NULL	NUMBER(4)
APELLIDO		VARCHAR2(10)
OFICIO		VARCHAR2(15)
DIR		NUMBER(4)
FECHA_ALT		DATE
SALARIO		NUMBER(7)
COMISION		NUMBER(7)
DEPT_NO		NUMBER(2)

TABLA EMPLE\_DEP20:

Nombre	¿Nulo?	Tipo
EMP_NO	NOT NULL	NUMBER(4)
APELLIDO		VARCHAR2(10)
OFICIO		VARCHAR2(15)
SALARIO		NUMBER(7)

Para cargar los datos en la nueva tabla podría hacerse con el comando:

```
insert into emple_dep20
  select emp_no, apellido, oficio, salario from emple where dept_no=20;
```

Si solo se hubiese querido guardar los números de empleado:

```
insert into emple_dep20 (emp_no)
  select emp_no from emple where dept_no=20;
```

### Actividad. Inserción de datos con consultas

- Crea la tabla NOMBRES (ver en Recursos). Es una tabla sencilla para hacer pruebas. Inserta en esa tabla, en el campo “nombre”, los apellidos de los empleados de EMPLE que pertenecen al departamento 20.
- Crea una nueva tabla DIRECTORES que tenga el mismo formato que EMPLE, excepto por el campo “oficio”, que no estará. Inserta en esta tabla todos los empleados cuyo oficio sea “DIRECTOR”.

En las inserciones con consultas puede resultar útil el empleo de constantes en los SELECT.

Observa esta consulta:

Los datos en verde son constantes, es decir, datos que no se extraen de la tabla sino que son devueltos tal cual (sysdate es la fecha actual)

```
select 1112, 'QUIROGA', oficio, dir, sysdate, salario, comision,
       dept_no from emple
       where apellido='GIL';
```

Esta consulta devuelve una mezcla entre constantes y datos de la tabla EMPLE:

1112	'QUIROGA'	OFICIO	DIR	SYSDATE	SALARIO	COMISION	DEPT_NO
1112	QUIROGA	ANALISTA	7566	11/02/22	3000	(null)	20

**Ejemplo:** Insertar en la tabla EMPLE un nuevo empleado de apellido “Quiroga”, con número de empleado 1112. Los datos restantes de ese empleado serán los mismos que los de “Gil” y la fecha de alta será el día de hoy.

```
insert into emple
select 1112, 'QUIROGA', oficio, dir, sysdate, salario, comision, dept_no
from emple where apellido='GIL';
```

### Actividad. Inserción de datos con consultas y constantes

- a) Tablas ALUM y NUEVOS. Estas tablas guardan información sobre alumnos de un centro: los actuales (ALUM) y los que han reservado plaza (NUEVOS). Una vez se ha completado el plazo de matriculación, los alumnos que habían reservado plaza pasan a ser alumnos actuales del centro. Inserta todos los alumnos de la tabla NUEVOS en la tabla ALUM, y después borra todo el contenido de la tabla NUEVOS.

Nota: ten en cuenta que algunos alumnos de NUEVOS ya están en ALUM (realmente solo hay que insertar los alumnos de NUEVOS que no están en ALUM)

- b) Tabla EMPLE. Se da de alta a un nuevo empleado en el mismo departamento que el empleado “SALA”. Inserta a ese empleado teniendo en cuenta que todos sus datos son como los de “SALA”, excepto los siguientes: su número de empleado es 2000, su apellido “SAAVEDRA”, su salario es un 20% superior al de “SALA” y la fecha de alta será el día de ayer (utiliza sysdate).



## 2.2. Borrado de datos de datos con consultas

Es posible realizar borrados de datos tomando como datos de entrada el resultado de una consulta.

La sintaxis básica utiliza el formato de las subconsultas:

```
DELETE [FROM] nombreTabla
WHERE valor Operador (SELECT ...);
```

Consideraciones:

- ▶ El listado devuelto por **SELECT** debe ser compatible con el operador que se emplea en la condición.
- ▶ El borrado y la consulta pueden realizarse a distintas tablas.
- ▶ El comando **SELECT** puede ser tan complejo como se necesite.

**Ejemplo:** Un prestamista tiene una tabla con sus clientes (**CLIENTES**) y otra con dnis que han sido morosos con algún otro prestamista en algún momento (**MOROSOS**):

TABLA CLIENTES:

DNI_CLIENTE	CUENTA_BANCARIA
11111111A	60000254122590021451
11111111B	60000111122233344452
11111111B	60000556664449023333
11111111C	60000555589633380004
11111111D	60000444456678991225

TABLA MOROSOS:

DNI_MOROSO
11111111B
22222222F

El prestamista desea borrar de tus clientes todas aquellas personas que están es su lista de morosos. Para ello utiliza el comando:

```
delete from clientes
where dni_cliente in (select dni_moroso from morosos);
```

Como resultado:

DNI_CLIENTE	CUENTA_BANCARIA
11111111A	60000254122590021451
11111111C	60000555589633380004
11111111D	60000444456678991225

### Actividad. Borrado de datos con consultas

- Tablas **ALUM** y **ANTIGUOS**. Borra de la tabla **ANTIGUOS** los alumnos que estén también en la tabla **ALUM**. Se considera que un alumno es el mismo si coincide su nombre, edad y localidad
- Tablas **EMPLE** y **DEPART**. Borra todos los departamentos de la tabla **DEPART** que no tengan empleados.

## 2.3. Modificación de datos con consultas

Se pueden incluir subconsultas en una sentencia UPDATE, tanto en la cláusula WHERE como en la cláusula SET.

```
UPDATE nombreTabla
SET columna1=valor1, columna2=(SELECT ...), ...
WHERE columna operador (SELECT ...);
```

SELECT debe devolver un único valor del mismo tipo que columna2

El valor o valores devueltos por SELECT deben ser coherentes con el operador empleado

Existe un formato simplificado que permite asignar valores a varias columnas a partir de una única subconsulta:

**Ejemplo:** en la tabla CENTROS se almacenan centros educativos con la estructura:

CENTROS (cod\_centro, tipo\_centro, nombre, direccion, telefono, num\_plazas)

Se quiere actualizar la dirección y número de plazas del centro 10, a los mismos valores del centro 50:

```
update centros set (direccion, num_plazas) =
(select direccion, num_plazas from centros where cod_centro=50)
where cod_centro=10;
```

```
UPDATE nombreTabla
SET (columna1, columna 2, ...)=(SELECT ...)
[WHERE condición];
```

SELECT debe devolver tantos valores como columnas hay en SET (y del mismo tipo)

Nota: si la subconsulta devolviese más de una fila, la consulta daría error.

Este formato reducido sólo es válido si se trata de una subconsulta. No sería posible (devolverá error):

```
update centros set (dirección, num_plazas) = ('C/Los Torneros 21', 310)
where cod_centro=10;
```

### Actividad. Actualización de datos con consultas

Tablas EMPLE y DEPART. Para todos los empleados del departamento "Contabilidad" se modifica su salario al doble del salario de "Sánchez".

### 3. Creación de tablas con consultas

La sentencia CREATE TABLE permite crear una tabla nueva a partir de la consulta de otra tabla o tablas ya existentes. **La nueva tabla contendrá la estructura y los datos devueltos por la consulta.**

No es necesario especificar los tipos de datos, ya que se calcularán automáticamente a partir de los datos obtenidos en la consulta. **La nueva tabla no contendrá ninguna restricción de las tablas originales** (se creará sin clave primaria).

```
CREATE TABLE nombreTabla [(columna1, columna2,...)]
AS SELECT ...;
```

Opcionalmente se pueden indicar nombres de columnas, si se desea que sean diferentes a las devueltas en la consulta

**Ejemplo:** crear una tabla llamada CENTROS\_P donde se almacenan solo los centros de tipo P (públicos) de la tabla CENTROS. La estructura de la nueva tabla debe ser igual a la de CENTROS:

```
create table centros_p
as select * from centros where tipo_centro='P';
```

- Crear una tabla llamada CENTROS\_P2 donde se almacenan solo los centros de tipo P (públicos) de la tabla CENTROS. La estructura de la nueva tabla debe ser igual a la de CENTROS, excepto por el campo “tipo\_centro”, que no aparecerá:

```
create table centros_p2
as select cod_centro,nombre,direccion,telefono,num_plazas
from centros where tipo_centro='P';
```

**Ejemplo:** a partir de EMPLE y DEPART, crear una tabla llamada EMPLEYDEPART donde aparezca el número y apellido de empleado y el nombre del departamento al que pertenece. Las columnas se nombrarán como “número”, “empleado” y “departamento”:

```
create table empleydepart (numero, empleado, departamento)
as select emp_no, apellido, dnombre
from emple, depart
where emple.dept_no=depart.dept_no;
```

#### Actividad. Creación de tablas con consultas

- A partir de EMPLE y DEPART crea la tabla DESARROLLADORES, que contendrá el número de empleado, apellido y nombre de departamento de todos los desarrolladores de la empresa (oficio desarrollador). El nombre de las columnas será “numero\_emple”, “empleado” y “departamento”.
- Visualiza la estructura de la tabla (DESCRIBE) y su contenido (SELECT).
- Comprueba las restricciones de DESARROLLADORES (vistas USER\_CONSTRAINTS y USER\_CONS\_COLUMNS).
- Modifica la tabla para que la clave primaria sea “numero”, y además sea foreign key referenciando a la tabla EMPLE, con borrado en cascada. Comprueba de nuevo las vistas USER\_CONSTRAINTS y USER\_CONS\_COLUMNS.

### Actividad. Comparación entre vistas y tablas creadas con consultas

- a) Crea una tabla nueva con el siguiente comando:

```
create table tabla_peor_pagados  
as select * from emple where salario=(select min(salario) from emple);
```

Observa que en ella se almacenarán los empleados que cobren el menor salario de toda la empresa. Comprueba su contenido:

```
select * from tabla_peor_pagados;
```

- b) A modo de recordatorio: una vista (VIEW) en base de datos es una tabla lógica. Se maneja como una tabla (INSERT, DELETE, SELECT...), pero realmente está obteniendo los datos de otras tablas (TABLES).

Para que el usuario alumno pueda crear vistas, primero hay que darle permisos (si no los tiene). Con el **usuario administrador SYSTEM**, da permisos a "alumno" para crear vistas:

```
alter session set container=xepdb1;  
grant create view to alumno;
```

Con el **usuario alumno**, crea la siguiente vista:

```
create view vista_peor_pagados  
as select * from emple where salario=(select min(salario) from emple);
```

Observa que en ella también se almacenarán los empleados que cobren el menor salario de toda la empresa. Comprueba su contenido:

```
select * from vista_peor_pagados;
```

- c) Inserta otro empleado que cobre el salario mínimo:

```
insert into emple values  
(9999, 'CATALÍN', 'ANALISTA', 7902, sysdate, 1040, null, 10);
```

- d) Vuelve a listar el contenido de la tabla y de la vista, y compara los resultados.

## 4. Transacciones

### Introducción

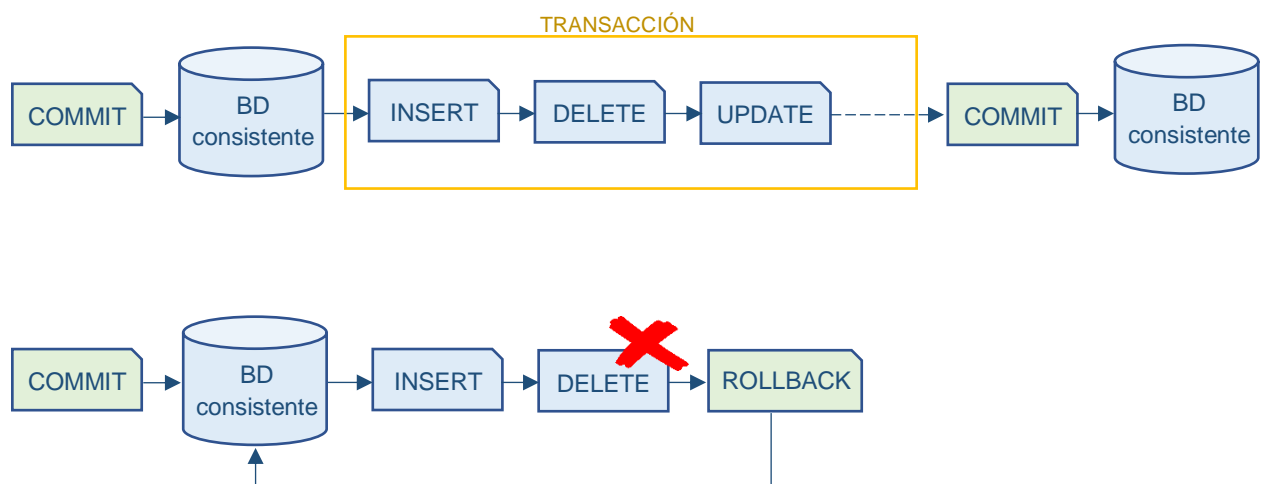
Una **transacción** es un conjunto de una o más sentencias SQL que juntas forman una unidad de trabajo.

Representa una única operación lógica (por ejemplo, la acción de transferir fondos de una cuenta bancaria a otra, aún cuando involucra cambios en varias tablas y diversos comandos).

**Cada transacción debe ejecutarse completamente o no ejecutarse. Si una transacción se ejecuta parcialmente (solo algunos comandos), puede llevar a la base de datos a un estado incoherente.**

→ **COMMIT** es el comando que se emplea para confirmar una transacción. Debe ejecutarse cuando se haya finalizado la ejecución de todos los comandos SQL que forman parte de la transacción. Una vez ejecutado, los cambios de la transacción pasan a ser permanentes.

→ **ROLLBACK** es el comando que se ejecuta para dar marcha atrás a una transacción no finalizada. Si en la ejecución de un comando SQL que forma parte de una transacción se detecta que hay algún problema (el comando no puede ejecutarse), se da marcha atrás a la transacción completa, deshaciendo las operaciones realizadas por los comandos ya ejecutados. Al hacer un rollback la base de datos vuelve al estado en que estaba en el último COMMIT ejecutado, es decir, vuelve al último estado coherente.



El concepto de transacción facilita que una base de datos relacional cumpla con los parámetros ACID:

**A**tomicidad: una transacción se ejecuta completamente o no se ejecuta (es indivisible)

**C**onsistencia: Una transacción lleva a la BD de un estado consistente a otro

**I**solation (aislamiento): aunque haya transacciones concurrentes, no se interfieren entre ellas

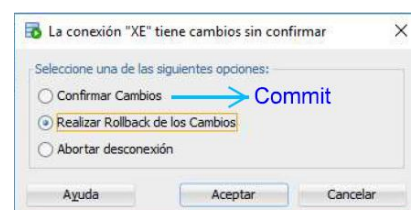
**D**urabilidad: una vez confirmada la transacción, los cambios perduran.

**Autocommit:** parámetro de la base de datos que establece un COMMIT automático después de cada sentencia SQL de tipo LMD (INSERT, DELETE o UPDATE).

<code>show autocommit;</code>	Comprobar el valor del parámetro
<code>set autocommit on;</code>	Activar el autocommit
<code>set autocommit off;</code>	Desactivar el autocommit

Realiza la siguiente prueba:

- Desde SQLDeveloper, desactiva el autocommit.
- Ejecuta el comando: `insert into depart values(99, 'BECAS', 'MADRID');`
- Comprueba que se ha creado el registro: `select * from depart;`
- Cierra SQLDeveloper. Debe aparecer una ventana de confirmación donde nos pregunta si queremos confirmar la transacción o realizar rollback. Pulsa “realizar Rollback de los cambios”.
- Abre de nuevo SQLDeveloper y comprueba el contenido de la tabla DEPART.



### Actividad. Transacciones

- Desactiva el autocommit y crea la siguiente tabla de prueba:  
`create table NUMERO (num number(2) primary key);`
- Ejecuta los siguientes comandos e interpreta el resultado:  
`insert into NUMERO values(1);`  
`commit;`  
`insert into NUMERO values(2);`  
`rollback;`  
`select * from NUMERO;`
- Ejecuta el comando: `insert into NUMERO values(3);`  
Abre una segunda conexión con la base de datos XEPDB1, sin cerrar la anterior, y comprueba el contenido de la tabla con `select * from NUMERO;`  
Confirma la conexión desde la primera conexión y comprueba de nuevo, desde la segunda, el contenido de la tabla.

- **Commit implícito y explícito**

Cuando se ejecutan ciertos comandos, la base de datos realiza COMMIT automáticamente (COMMIT implícito):

QUIT	DISCONNECT	CREATE VIEW	ALTER
EXIT	CREATE TABLE	DROP VIEW	REVOKE
CONNECT	DROP TABLE	GRANT	AUDIT
			NOAUDIT

Sólo es necesario realizar un COMMIT explícito empleando el comando en las sentencias LMD (lenguaje de manipulación de datos): INSERT, DELETE, UPDATE.

- **Rollback automático**

Si se produce un fallo de sistema como por ejemplo un corte de alimentación, el SGBD realiza un ROLLBACK automático para devolver a la base de datos al último estado consistente. Eso conllevará tener que repetir el trabajo realizado y no confirmado antes del fallo de sistema.