TRANSACCIONES

- Una transacción es una o más sentencias SQL procesadas como una acción simple en la Base de Datos
- Por defecto todas las transacciones son multisentencia
- La transacción comienza con la primera sentencia SQL realizada después de una sentencia **`connect`**, **`commit`** o **`rollback`**
- La transacción finaliza cuando se utiliza una sentencia **`commit`** o **`rollback`**
- La finalización de una sesión o la desconexión de la base de datos implica la ejecución de una sentencia ***commit*** implícita

TRANSACCIONES

- Las actualizaciones de la Base de Datos realizadas durante una transacción no son visibles para los demás usuarios hasta que la transacción finaliza con **`commit`**
- Durante cada transacción se bloquean las páginas accedidas para preservar la consistencia de los datos
- Las páginas quedan liberadas al finalizar la transacción
- Las sentencias relacionadas con el control de transacciones son: **`commit`**, **`rollback`** y **`savepoint`**

TRANSACCIONES. SENTENCIAS

- La sentencia **`commit`** finaliza una transacción haciendo que las actualizaciones realizadas en la base de datos sean permanentes
- La sentencia **`rollback`** finaliza una transacción haciendo que las actualizaciones realizadas en la base de datos no tengan efecto
- La sentencia **`rollback`** en conjunción con la sentencia **`savepoint`** permiten deshacer parcialmente los efectos de una transacción sin finalizar la transacción.

TRANSACCIONES. SENTENCIA *COMMIT*

COMMIT ;

EXEC SQL COMMIT ;

```
insert into PRESTAMO values (132, 74567890, 5023024, 6,
to_date('4/5/2012', 'dd/mm/yyyy'), NULL);

insert into PRESTAMO values (144, 74567890, 5024020, 6,
to_date('4/5/2012', 'dd/mm/yyyy'), NULL);

commit;
```

TRANSACCIONES. SENTENCIA *ROLLBACK*

ROLLBACK ;

EXEC SQL ROLLBACK ;

```
select * from univ.prestamo where Cod_Lector='74567890';

insert into UNIV.PRESTAMO values (927, 74567890, 5025496, 9,
to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

insert into UNIV.PRESTAMO values (928, 74567890, 5025364, 9,
to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

select * from univ.prestamo where Cod_Lector='74567890';

rollback;

select * from univ.prestamo where Cod_Lector='74567890';
```

TRANSACCIONES. SENTENCIA *ROLLBACK*

```
SQL> select * from univ.prestamo where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
      3324   74567890 5023936             1 23-JAN-11

SQL> insert into UNIV.PRESTAMO values (927, 74567890, 5025496, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

1 row created.

SQL> insert into UNIV.PRESTAMO values (928, 74567890, 5025364, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

1 row created.

SQL> select * from univ.prestamo where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
       927   74567890 5025496             9 06-MAY-12
       928   74567890 5025364             9 06-MAY-12
      3324   74567890 5023936             1 23-JAN-11

SQL> rollback;

Rollback complete.

SQL> select * from univ.prestamo where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
      3324   74567890 5023936             1 23-JAN-11

SQL>
```

TRANSACCIONES. SENTENCIAS *ROLLBACK* y *SAVEPOINT*

SAVEPOINT *nombre* ;

ROLLBACK TO *nombre* ;

EXEC SQL SAVEPOINT *nombre* ;

EXEC SQL ROLLBACK TO *nombre* ;

TRANSACCIONES. SENTENCIAS *ROLLBACK* y *SAVEPOINT.* EJEMPLO

```
insert into UNIV.PRESTAMO values (927, 74567890,
5025496, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

savepoint uno;

insert into UNIV.PRESTAMO values (928, 74567890,
5025364, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

select * from univ.prestamo
where Cod_Lector='74567890';

rollback to uno ;

select * from univ.prestamo where
Cod_Lector='74567890';

rollback;

select * from univ.prestamo where
Cod_Lector='74567890';
```

TRANSACCIONES. SENTENCIAS *ROLLBACK* y *SAVEPOINT.* EJEMPLO

```
SQL> insert into UNIV.PRESTAMO values (927, 74567890, 5025496, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

1 row created.

SQL> savepoint uno;

Savepoint created.

SQL> insert into UNIV.PRESTAMO values (928, 74567890, 5025364, 9, to_date('6/5/2012', 'dd/mm/yyyy'), NULL);

1 row created.

SQL> select * from univ.prestamo                      where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
       927   74567890 5025496             9 06-MAY-12
       928   74567890 5025364             9 06-MAY-12
      3324   74567890 5023936             1 23-JAN-11

SQL> rollback to uno ;

Rollback complete.

SQL> select * from univ.prestamo where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
       927   74567890 5025496             9 06-MAY-12
      3324   74567890 5023936             1 23-JAN-11

SQL> rollback;

Rollback complete.

SQL> select * from univ.prestamo where Cod_Lector='74567890';

    CODIGO COD_LECTOR ISBN          COD_SUC FECHA_INI FECHA_DEV
---------- ---------- ---------- ---------- --------- ---------
       276   74567890 5023048            14 12-MAR-01 28-MAR-01
      3324   74567890 5023936             1 23-JAN-11

SQL>
```

TRANSACCIONES MONOSENTENCIA

SET AUTOCOMMIT ON | OFF ;

EXEC SQL SET AUTOCOMMIT ON | OFF ;

- Si se ha utilizado **`autocommit on`** cada sentencia es una transacción que finaliza automáticamente con un **`commit`** implícito
- No se puede cambiar el **`autocommit`** en medio de una transacción
- Si se abre un cursor cuando está **`autocommit on`**, no se realizará el **`commit`** hasta que se cierre el cursor, ya que los cursores no pueden permanecer abiertos entre transacciones

Niveles de aislamiento de transacciones en Oracle

Leer comprometido

- Es el nivel predeterminado.
- Cada consulta ejecutada por una transacción solo ve los datos que se confirmaron antes de que comenzara la consulta (no la transacción).
- Una consulta de Oracle nunca lee datos no confirmados.
- Este nivel no impide que otras transacciones modifiquen los datos leídos por una consulta o añada nuevos datos; los datos pueden ser modificados por otras transacciones entre dos ejecuciones de la consulta: se pueden dar lecturas no repetibles y lecturas fantasma.

Niveles de aislamiento de transacciones en Oracle

Serializable

- Las transacciones serializables solo ven los cambios que se confirmaron en el momento en que comenzó la transacción, además de los cambios realizados por la propia transacción.
- Las transacciones serializables no experimentan lecturas no repetibles o ni lecturas fantasmas.

Solo lectura

- Las transacciones de solo lectura solo ven los cambios que se confirmaron en el momento en que comenzó la transacción y no permiten declaraciones `INSERT`, `UPDATE` y `DELETE`.

TRANSACCIONES EN ORACLE

Se puede establecer el nivel de aislamiento de una transacción utilizando una de estas declaraciones al comienzo de una transacción:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION READ ONLY;
```

Se puede usar la declaración ALTER SESSION para establecer el nivel de aislamiento para todas las transacciones posteriores

```
ALTER SESSION SET ISOLATION_LEVEL SERIALIZABLE;
ALTER SESSION SET ISOLATION_LEVEL READ COMMITTED;
```

GESTIÓN DE TRANSACCIONES

- Las transacciones deben ser lo más cortas posibles para evitar alargar los bloqueos.
- Es importante evitar mantener abierta una transacción mientras se espera una entrada de usuario.
- Para minimizar las situaciones de interbloqueo, es recomendable que las transacciones que accedan a los mismos datos, hagan siempre los accesos en el mismo orden.
- Siempre que sea posible, se deberán organizar las operaciones de una transacción de modo que las escrituras se hagan todas seguidas al final de la transacción.

GENERACIÓN DE PLANES DE EJECUCIÓN

EXPLAIN PLAN [SET STATEMENT_ID = 'text'] FOR statement;

SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY);

DELETE from plan_table;

- El comando **`explain`** plan muestra la ejecución de los planes elegidos por el optimizador de Oracle para SELECT, UPDATE, INSERT y DELETE sin llegar a ejecutar dicha consulta.
- Opcionalmente se puede incluir una cláusula para etiquetar el plan de ejecución almacenado en PLAN_TABLE
- Se consulta la tabla de plan utilizando el procedimiento DBMS_XPLAN
- Borrar las tuplas cuando se finalice

GENERACIÓN DE PLANES DE EJECUCIÓN

```
SQL> explain plan for select * from lector where nombre like 'M%';

Explained.

SQL> select * from table (DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan hash value: 595797030

--------------------------------------------------------------------------------
| Id  | Operation                    | Name       | Rows  | Bytes | Cost (%CPU)| Time     |
--------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |            |    10 |   690 |     2   (0)| 00:00:01 |
|   1 |  TABLE ACCESS BY INDEX ROWID| LECTOR     |    10 |   690 |     2   (0)| 00:00:01 |
|*  2 |   INDEX RANGE SCAN           | INDICEPROV |    10 |       |     1   (0)| 00:00:01 |
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
---------------------------------------------------

   2 - access("NOMBRE" LIKE 'M%')
       filter("NOMBRE" LIKE 'M%')

15 rows selected.

SQL> delete from plan_table;

3 rows deleted.

SQL> commit;

Commit complete.
```

```
SQL> explain plan for select * from lector l, prestamo p where l.codigo=p.cod_lector and nombre like 'M%';

Explained.

SQL> select * from table (DBMS_XPLAN.DISPLAY);

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Plan hash value: 1435669034

-------------------------------------------------------------------------------------------------
| Id  | Operation                     | Name          | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |               |    45 |  4590 |     4   (0)| 00:00:01 |
|   1 |  NESTED LOOPS                 |               |    45 |  4590 |     4   (0)| 00:00:01 |
|   2 |   TABLE ACCESS FULL           | PRESTAMO      |   664 | 21912 |     3   (0)| 00:00:01 |
|*  3 |   TABLE ACCESS BY INDEX ROWID| LECTOR        |     1 |    69 |     1   (0)| 00:00:01 |
|*  4 |    INDEX UNIQUE SCAN          | SYS_C00412188 |     1 |       |     0   (0)| 00:00:01 |
-------------------------------------------------------------------------------------------------

PLAN_TABLE_OUTPUT
--------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   3 - filter("NOMBRE" LIKE 'M%')
   4 - access("L"."CODIGO"="P"."COD_LECTOR")

17 rows selected.
```