:: EXERCISE 7 ::

[1] What is Snapshot? How to create Snapshot?

[2] What are the difference between snapshot table and virtual table?

5.8 WHAT ARE LOCKS?

In multi-user systems, many users may update the same information at the same time. Locking allows only one user to update a particular data block while another person cannot modify the same data.

The basic idea of locking is that when a user modifies data, that modified data is locked by that transaction until the transaction is committed or rolled back. The lock is held until the transaction is complete - this known as ***data concurrency.*** (સમાંતર)

The second purpose of locking is to ensure (ખાતરી) that all processes can always access (read) the original data as they were at the time the query began (uncommitted modification), This is known as ***read consistency.*** (અસંગતતા)

[1] Locking issue :

- **Lost updates :** (ગુમાવેલ - updates)

  A lost update is a classic database problem. Actually, it is a problem in all multiuser-environments. Simply put, a lost update occurs when the following events occur, such like:

  (1) A transaction in Session1 retrieves (queries) (મેળવે-ક્વેરી) a row of data into local memory and displays it to an end user, User1.

  (2) Another transaction in Session2 retrieves that same row, but displays the data to a different end user, User2.

  (3) User1, using the application, modifies that row and has the application update the database and commit. Session1's transaction is now complete.

  (4) User2 modifies that row also, and has the application update the database and commit. Session2's transaction is now complete.

  This process is referred to as a lost update because all of the changes made in Step 3 will be lost.

  For example, an employee update screen that allows a user to change an address, city, work number, and so on. The application itself is very simple: a small search screen to generate a list of employees and then the ability to drill down into the details of each employee. This should be a piece of cake. So, we write the application with no locking on our part, just simple SELECT and UPDATE commands.

- **Pessimistic Locking :** (નિરાશાવાદી)

  Pessimistic Locking prevents any other application or user from fetching or updating the same record at the same time. Pessimistic locking can be a very powerful mechanism, but often databases can impose limitations. Some

databases, such as Oracle, lock a single record; others (for example, Sybase) lock a page or group of records; still others, including Microsoft Access, do not support pessimistic locking at all. Some databases, such as Oracle, prevent anyone else from fetching an object while others, such as Sybase, raise an exception when an attempt is made to update a record that has already been locked and updated.

When a user queries some data and picks a row to change the below statement is used:

Example :

```
selectempno, ename, sal from emp
where empno = :empno and ename = :ename and sal = :sal
for update nowait;
```

➤ **Optimistic Locking :**

Optimistic locking is a technique for SQL database applications that does not lock rows for viewing, updating or deleting.

The rows with DML query changes are locked until changes are committed.

Since no locks are taken out during the read, it doesn't matter if the user goes to lunch after starting a transaction, and deadlocks are all but eliminated since users should never have to wait on each other's locks.

The Oracle database uses optimistic locking by default. Any command that begins with UPDATE...SET that is not preceded by a SELECT...FOR UPDATE is an example of optimistic locking. Concurrency control is the mechanism that ensures that the data being written back to the database is consistent with what was read from the database in the first place - that is that no other transaction has updated the data after it was read.

➤ **Blocking :**

Blocking occurs when one session holds a lock on a resource that another session is requesting.

As a result, the requesting session will be blocked—it will hang until the holding session gives up the locked resource.

In almost every case, blocking is avoidable. The five common DML statements that will block in the database are INSERT, UPDATE, DELETE, MERGE, and SELECT FOR UPDATE. The solution to a blocked SELECT FOR UPDATE is trivial: simply add the NOWAIT clause and it will no longer block. Instead, your application will report back to the end user that the row is already locked.

➤ **Deadlocks :**

Deadlocks occur when you have two sessions, each of which is holding a resource that the other wants.

For example, if I have two tables, A and B, in my database, and each has a single row in it, I can demonstrate a deadlock easily. All I need to do is open two sessions (e.g., two SQL*Plus sessions). In session A, I update table A. In session

B, I update table B. Now, if I attempt to update table A in session B, I will become blocked. Session A has this row locked already. This is not a deadlock; it is just blocking. I have not yet deadlocked because there is a chance that session A will commit or roll back and session B will simply continue at that point.

If I go back to session A and then try to update table B, I will cause a deadlock. One of the two

Sessions will be chosen as a victim and will have its statement rolled back. For example, the attempt by session B to update table A may be rolled back, with an error such as the following:

```
Update a set x = x+1;

ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource
```

Session A's attempt to update table B will remain blocked—Oracle will not roll back the entire transaction. Only one of the statements that contributed to the deadlock is rolled back. Session B still has the row in table B locked, and session A is patiently waiting for the row to become available. After receiving the deadlock message, session B must decide whether to commit the outstanding work on table B, roll it back, or continue down an alternate path and commit later. As soon as this session does commit or roll back, the other blocked session will continue on as if nothing happened.

- **Lock Escalation :**

When lock escalation occurs, the system is decreasing the granularity of your locks. An example would be the database system turning your100 row-level locks against a table into a single table-level lock. Lock escalation is used frequently in databases that consider a lock to be a scarce resource and overhead to be avoided.

Oracle will take a lock at the lowest level possible (i.e., the least restrictive lock possible) and convert that lock to a more restrictive level if necessary. For example, if you select a row from a table with the FOR UPDATE clause, two locks will be created. One lock is placed on the row(s) you selected the other lock, a ROW SHARE TABLE lock, is placed on the table itself. This will prevent other sessions from placing an exclusive lock on the table and thus prevent them from altering the structure of the table, for example. Another session can modify any other row in this table without conflict. As many commands as possible that could execute successfully given there is a locked row in the table will be permitted.

Lock escalation is not a database "feature." It is not a desired attribute. The fact that a database supports lock escalation implies there is some inherent overhead in its locking mechanism and significant work is performed to manage hundreds of locks.

Take Note : Oracle will never escalate a lock. Oracle typically refers to the process as Lockconversion.

B, I update table B. Now, if I attempt to update table A in session B, I will become blocked. Session A has this row locked already. This is not a deadlock; it is just blocking. I have not yet deadlocked because there is a chance that session A will commit or roll back and session B will simply continue at that point.

If I go back to session A and then try to update table B, I will cause a deadlock. One of the two

Sessions will be chosen as a victim and will have its statement rolled back. For example, the attempt by session B to update table A may be rolled back, with an error such as the following:

```
Update a set x = x+1;
ERROR at line 1:
ORA-00060: deadlock detected while waiting for resource
```

Session A's attempt to update table B will remain blocked—Oracle will not roll back the entire transaction. Only one of the statements that contributed to the deadlock is rolled back. Session B still has the row in table B locked, and session A is patiently waiting for the row to become available. After receiving the deadlock message, session B must decide whether to commit the outstanding work on table B, roll it back, or continue down an alternate path and commit later. As soon as this session does commit or roll back, the other blocked session will continue on as if nothing happened.

- **Lock Escalation :**

When lock escalation occurs, the system is decreasing the granularity of your locks. An example would be the database system turning your100 row-level locks against a table into a single table-level lock. Lock escalation is used frequently in databases that consider a lock to be a scarce resource and overhead to be avoided.

Oracle will take a lock at the lowest level possible (i.e., the least restrictive lock possible) and convert that lock to a more restrictive level if necessary. For example, if you select a row from a table with the FOR UPDATE clause, two locks will be created. One lock is placed on the row(s) you selected the other lock, a ROW SHARE TABLE lock, is placed on the table itself. This will prevent other sessions from placing an exclusive lock on the table and thus prevent them from altering the structure of the table, for example. Another session can modify any other row in this table without conflict. As many commands as possible that could execute successfully given there is a locked row in the table will be permitted.

Lock escalation is not a database "feature." It is not a desired attribute. The fact that a database supports lock escalation implies there is some inherent overhead in its locking mechanism and significant work is performed to manage hundreds of locks.

Take Note : Oracle will never escalate a lock. Oracle typically refers to the process as Lockconversion.

[2] Lock Types :

There a number of different types of locks as listed below :

- **DML Locks** - DML (data manipulation language), in general SELECT, INSERT, UPDATE and DELETE. DML locks will be locks on a specific row of data, or a lock at the table level, which locks every row in the table.
- **DDL locks** - DDL (data definition language), in general CREATE, ALTER and so on. DDL locks protect the definition of the structure of objects.
- **Latches** - These are locks that Oracle uses to protect its internal data structure.
- **DML Locks :**

Data manipulation language (DML) lock is placed over the row. This lock prevents other processes from updating (or locking) the row. This lock is released only when the locking process successfully commits the transaction to the database (i.e., makes the updates to that transaction permanent) or when the process is rolled back.

The complete set of DML locks are

Row Share	permits concurrent access but prohibits others from locking table for exclusive access
Row Exclusive	same as row share but also prohibits locking in share mode
Share	permits concurrent queries but prohibits updates to the table
Share Row Exclusive	prevent others from locking in share mode or updating the rows on the whole table
Exclusive	permits queries but no DML against the table but select ok

Example : `DML Lock using NOWAIT`

```
select * from employee for update nowait;
select * from employee for update wait 10;
```

Take Note : the above commands will abort if the lock is not release in the specified time period.

- **DDL locks :**

Data dictionary language (DDL) lock is placed over the table to prevent structural alterations to the table. For example, this type of lock keeps the DBA from being able to remove a table by issuing a DROP statement against the table. This lock is released only when the locking process successfully commits the transaction to the database or when the process is rolled back.

DDL locks are automatically placed against objects during a DDL operation to protect them from changes by other sessions.

There are three types of DDL locks

- **Exclusive DDL Locks :** These prevent other sessions from gaining a DDL lock or TM locks themselves. You can query a table but not modify it. Exclusive locks will normally lock the object until the statement has finished. However in some instances you can use the ONLINE option which only uses a low-level lock this still locks DDL operations but allows DML to occur normally.
- **Share DDL Locks :** This protects the structure of the referenced object against modification by other sessions, but allows modification to the data. Shared DDL locks allow you to modify the contents of a table but not their structure.
- **Breakable Parse Locks :** This allows an object, such as a query plan cached in the shared pool to register its reliance on some objects. If you perform a DDL against that object, Oracle will review the list of objects that have registered their dependence, and invalidate them. Hence these locks are breakable; they do not prevent the DDL from occurring. Breakable parse locks are used when a session parses a statement, a parse lock is taken against every object referenced by that statement. These locks are taken in order to allow the parsed, cached statement to be invalidated (flushed) in the shared pool if a reference object is dropped or alter in some way. Use the below SQL to identify any parse locks on views, procedures, grants, etc

➢ **Latches :**

Latches are locks that are held for short period of time, for example the time it takes to modify an in-memory data structure. They are used to protect certain memory structures such as the database block buffer cache or the library cache in the shared pool.

They are lightweight low-level serialization mechanism to protect the in-memory data structures of the SGA. They do not support queuing and do not protect database objects such as tables or data files.

Oracle uses atomic instructions like "test and set" and "compare and swap" for operating on latches. Since the instructions to set and free latches are atomic, the operating system itself guarantees that only one process gets to test and set the latch even though many processes may be going for it simultaneously. Since the instruction is only one instruction, it can be quite fast (but the overall latching algorithm itself is many CPU instructions!). Latches are held for short periods of time and provide a mechanism for cleanup in case a latch holder dies abnormally while holding it. This cleanup process would be performed by PMON.

It is possible to use manual locking using the FOR UPDATE statement or LOCK TABLE statement, or you can create your own locks by using the DBMS_LOCK package.

➤ **Manual Locking and User-Defined Locks :**
When we update a table, Oracle places a TM lock on it to prevent other sessions from dropping that table (or performing most DDL).We have TX locks that are left on the various blocks we modify so others can tell what data we own. The database employs DDL locks to protect objects from change while we ourselves are changing them. It uses latches and locks internally to protect its own structure.

There are two options for locking table

Manually lock data via a SQL statement.

Create our own locks via the DBMS_LOCK package.

➤ **Manual Locking :**
The SELECT...FOR UPDATE statement is the predominant method of manually locking data. We can also manually lock data using the LOCK TABLE statement. This statement is used rarely, because of the coarseness of the lock. It simply locks the table, not the rows in the table. If you start modifying the rows, they will be locked as normal. So, this is not a method to save on resources (as it might be in other RDBMSs). You might use the LOCK TABLE IN EXCLUSIVE MODE statement if you were writing a large batch update that would affect most of the rows in a given table and you wanted to be sure that no one would block you. By locking the table in this manner, you can be assured that your update will be able to do all of its work without getting blocked by other transactions. It would be the rare application; however, that has a LOCK TABLE statement in it.

➤ **Creating Your Own Locks :**
Oracle actually exposes to developers the enqueue lock mechanism that it uses internally, via the DBMS_LOCK package. You might use this package to serialize access to some resource external to Oracle. The DBMS_LOCK package allows you to manually release a lock when you are done with it, or to give it up automatically when you commit, or even to keep it as long as you are logged in.

:: EXERCISE 8 ::

[1] What is Locking in Database?

[2] What are the Locking Issues in database? Explain any two of them.

[3] Explain types of Locking.