**Oracle Transactions**

A multiuser database must provide the following:

* The assurance that users can access data at the same time ([**data concurrency**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-D7E696DB-944C-4798-B70D-5C2381FE971F))
* The assurance that each user sees a consistent view of the data ([**data consistency**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-B016467E-5663-4AC8-B54D-181CA1B8198E)), including visible changes made by the **user's own transactions** and **committed transactions** of other users

To describe consistent transaction behavior when transactions run concurrently, database researchers have defined a transaction isolation model called [**serializability**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-41BC7551-23D4-4778-8109-B886FBFC85EE). A serializable transaction operates in an environment that makes it appear as if no other users were modifying data in the database.

While this degree of isolation between transactions is generally desirable, running many applications in serializable mode can seriously compromise application throughput. Complete isolation of concurrently running transactions could mean that one transaction cannot perform an insertion into a table being queried by another transaction. In short, real-world considerations usually require a compromise between perfect transaction isolation and performance.

Oracle Database maintains data consistency by using a [**multiversion consistency model**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-FCCF7DC5-C860-4F1C-8811-03AE76597104) and various types of locks and transactions. In this way, the database can present a view of data to multiple concurrent users, with each view consistent to a point in time. Because different versions of data blocks can exist simultaneously, transactions can read the version of data committed at the point in time required by a [**query**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-CCF91C9F-A98A-498F-A84B-58A0FA16CD6E) and return results that are consistent to a single point in time.

Oracle Database never permits a dirty read, which occurs when a transaction reads uncommitted data in another transaction.

To illustrate the problem with dirty reads, suppose one transaction updates a column value without committing. A second transaction reads the updated and dirty (uncommitted) value. The first session rolls back the transaction so that the column has its old value, but the second transaction proceeds using the updated value, corrupting the database. Dirty reads compromise [**data integrity**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-9DE527B3-8901-4F4C-A18F-D2C2C307AFE1), violate foreign keys, and ignore unique constraints.

Oracle Database always enforces [**statement-level read consistency**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-2AA6E377-1D7A-4164-902C-67D3BB355A6F), which guarantees that data returned by a single query is committed and consistent for a single point in time. The point in time to which a single SQL statement is consistent depends on the transaction isolation level and the nature of the query:

In the **read committed** isolation level, this point is the time at which the *statement* was opened.

In a **serializable** or **read-only** transaction, this point is the time the *transaction* began.

Oracle Database can also provide read consistency to all queries in a transaction, known as [**transaction-level read consistency**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-504CA548-5AFF-4480-9F36-5C7D5FB457A5). In this case, each statement in a transaction sees data from the *same* point in time, which is the time at which the transaction began.

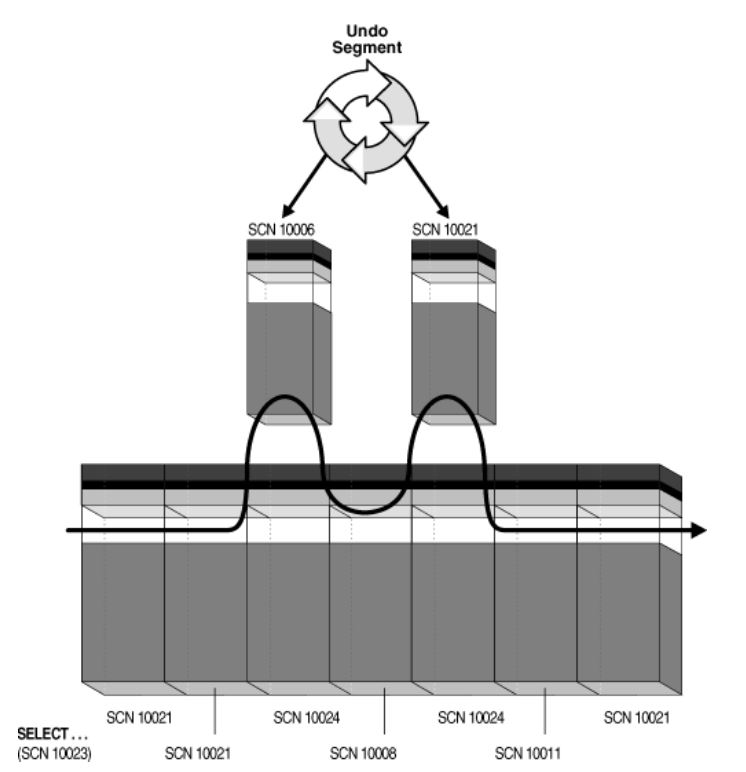
Queries made by a serializable transaction see changes made by the transaction itself. For example, a transaction that updates employees and then queries employees will see the updates. Transaction-level read consistency produces repeatable reads and does not expose a query to phantom reads.

To manage the multiversion read consistency model, the database must create a read-consistent set of data when a table is simultaneously queried and updated. Oracle Database achieves read consistency through [**undo data**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-297B963A-989C-4720-B061-A2352FF72892).

Whenever a user modifies data, Oracle Database creates undo entries, which it writes to undo segments. The undo segments contain the old values of data that have been changed by uncommitted or recently committed transactions. Thus, multiple versions of the same data, all at different points in time, can exist in the database. The database can use snapshots of data at different points in time to provide read-consistent views of the data and enable nonblocking queries.

As the database retrieves data blocks on behalf of a query, the database ensures that the data in each block reflects the contents of the block when the query began.

The database uses an internal ordering mechanism called an [**SCN**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-4B14A746-A8B3-4123-A02A-3FC1C293042C) to guarantee the order of transactions. As the SELECT statement enters the execution phase, the database determines the SCN recorded at the time the query began executing.



Blocks with SCNs *after* 10023 indicate changed data, as shown by the two blocks with SCN 10024. The SELECT statement requires a version of the block that is consistent with committed changes. The database copies current data blocks to a new buffer and applies undo data to reconstruct previous versions of the blocks. These reconstructed data blocks are called *consistent read (CR) clones*.

The database creates two CR clones: one block consistent to SCN 10006 and the other block consistent to SCN 10021. The database returns the reconstructed data for the query.

The [**block header**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-78394D5D-A1F6-4978-8F13-028185F514B5) of every segment block contains an [**interested transaction list (ITL)**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-0452FF71-5069-4F5C-B275-F1CDB50DB02D). The database uses the ITL to determine whether a transaction was uncommitted when the database began modifying the block.

Entries in the ITL describe which transactions have rows locked and which rows in the block contain committed and uncommitted changes. The ITL points to the transaction table in the undo segment, which provides information about the timing of changes made to the database.

In a sense, the block header contains a recent history of transactions that affected each row in the block. The INITRANS parameter of the CREATE TABLE and ALTER TABLE statements controls the amount of transaction history that is kept.

The SQL standard, which has been adopted by ANSI, defines four levels of transaction isolation. These levels have differing degrees of impact on transaction processing throughput.

These isolation levels are defined in terms of phenomena that must be prevented between concurrently executing transactions. The preventable phenomena are:

**1. Dirty reads**

A transaction reads data that has been written by another transaction that has not been committed yet.

**2. Nonrepeatable (fuzzy) reads**

A transaction rereads data it has previously read and finds that another committed transaction has modified or deleted the data. For example, a user queries a row and then later queries the same row, only to discover that the data has changed.

**3. Phantom reads**

A transaction reruns a query returning a set of rows that satisfies a search condition and finds that another committed transaction has inserted additional rows that satisfy the condition.

For example, a transaction queries the number of employees. Five minutes later it performs the same query, but now the number has increased by one because another user inserted a record for a new hire. More data satisfies the query criteria than before, but unlike in a fuzzy read the previously read data is unchanged.

The SQL standard defines four levels of isolation in terms of the phenomena that a transaction running at a particular isolation level is permitted to experience.

Read uncommitted (possible: 1,2,3)

Read committed (possible: 2,3)

Repeatable read (possible: 3)

Serializable (possible: none)

Read Committed Isolation Level

In the [**read committed isolation level**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-794EF3E1-8D64-439A-89C8-CA9A03EB6043), which is the default, every query executed by a transaction sees only data committed before the query - not the transaction - began.

A query in a read committed transaction avoids reading data that commits while the query is in progress. For example, if a query is halfway through a scan of a million-row table, and if a different transaction commits an update to row 950,000, then the query does not see this change when it reads row 950,000. However, because the database does not prevent other transactions from modifying data read by a query, other transactions may change data *between* query executions. Thus, a transaction that runs the same query twice may experience fuzzy reads and phantoms.

In a read committed transaction, a **conflicting write** occurs when the transaction attempts to change a row updated by an uncommitted concurrent transaction, sometimes called a *blocking transaction*. The read committed transaction waits for the blocking transaction to end and release its row lock.

If the blocking transaction rolls back, then the waiting transaction proceeds to change the previously locked row as if the other transaction never existed.

If the blocking transaction commits and releases its locks, then the waiting transaction proceeds with its intended update to the newly changed row.

Serializable Isolation Level

In the **serializable isolation level**, a transaction sees only changes committed at the time the transaction - not the query - began and changes made by the transaction itself. A serializable transaction operates in an environment that makes it appear as if no other users were modifying data in the database.

Oracle Database permits a serializable transaction to modify a row only if changes to the row made by other transactions were *already* committed when the serializable transaction began. The database generates an error when a serializable transaction tries to update or delete data changed by a different transaction that committed *after* the serializable transaction began:

ORA-08177: Cannot serialize access for this transaction

The [**read-only isolation level**](https://docs.oracle.com/database/121/CNCPT/glossary.htm#GUID-2AEC7E6A-AFDF-4380-974B-16A8A5AAE779) is similar to the serializable isolation level, but read-only transactions do not permit data to be modified in the transaction.

Setting the isolation level:

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE | READ ONLY| READ COMMITTED

ALTER SESSION SET ISOLATION LEVEL SERIALIZABLE | READ COMMITTED

ALTER SYSTEM …

T1 T2

1. read(X)

2. write(X)

3. commit

4. read(X)

Question: Can T1 see the new X value in step 4?

**Example:**

CREATE TABLE tr\_proba(sorsz NUMBER(4), szam NUMBER, szoveg VARCHAR2(40));

INSERT INTO tr\_proba VALUES(1, 10, 'First row');

INSERT INTO tr\_proba VALUES(2, 20, 'Second row');

1. session 2. session

SET AUTOCOMMIT OFF

------------------------> SET AUTOCOMMIT OFF

SELECT \* FROM tr\_proba;

------------------------> SELECT \* FROM tr\_proba;

UPDATE tr\_proba

SET szam=szam+1

WHERE sorsz=1;

------------------------> SELECT \* FROM tr\_proba;

COMMIT;

------------------------> SELECT \* FROM tr\_proba; **(can see new value)**

----------------------------------------------------------

COMMIT;

SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;

SELECT \* FROM tr\_proba;

------------------------> SELECT \* FROM tr\_proba;

UPDATE tr\_proba

SET szam=szam+1

WHERE sorsz=1;

------------------------> SELECT \* FROM tr\_proba;

COMMIT;

------------------------> SELECT \* FROM tr\_proba; **(cannot see new value)**

Locks in Oracle

In general, the database uses two types of locks: exclusive locks and share locks. Only one exclusive lock can be obtained on a resource such as a row or a table, but many share locks can be obtained on a single resource. For tables there are other lock modes as well.

(V$LOCK table lmode column)

(RS -> 2) LOCK TABLE <tab> IN ROW SHARE MODE

(RX -> 3) LOCK TABLE <tab> IN ROW EXCLUSIVE MODE

(S -> 4) LOCK TABLE <tab> IN SHARE MODE

(SRX -> 5) LOCK TABLE <tab> IN SHARE ROW EXCLUSIVE MODE

(X ->6) LOCK TABLE <tab> IN EXCLUSIVE MODE

**Queries about locks held by sessions:**

SELECT \* FROM v$session WHERE username=USER;

Session id (sid) of the current query can be queried with the following:

SELECT sys\_context('userenv', 'sid') FROM dual;

Which session hold a lock and since when (CTIME)?

SELECT se.sid, se.username, lo.type, lo.lmode, lo.ctime

FROM v$lock lo, v$session se

WHERE se.sid = lo.sid AND username = 'NIKOVITS';

SID USERNAME TYPE LMODE CTIME

-------------------------------------------

305 NIKOVITS TM 3 18

305 NIKOVITS TX 6 18

TM means table lock

TX means row lock

See V$LOCK\_TYPE

Which session waits for a lock (REQUEST > 0), which is a blocking session (BLOCK = 1), and for what time does it wait or block (CTIME)?

SELECT se.sid, se.username, lo.type, lo.lmode,

lo.request, lo.ctime, block

FROM v$lock lo, v$session se

WHERE se.sid = lo.sid AND username = 'NIKOVITS';

SID USERNAME TY LMODE REQUEST CTIME BLOCK

--------- ---------- -- --------- --------- --------- ---------

16 NIKOVITS TM 3 0 4499 0

16 NIKOVITS TX 6 0 4499 **1**

29 NIKOVITS TX 0 **6** 186 0

29 NIKOVITS TM 3 0 186 0

Which objects are locked currently?

SELECT lo.oracle\_username, lo.session\_id, lo.locked\_mode,

db.object\_name, db.object\_type

FROM v$locked\_object lo, dba\_objects db

WHERE lo.object\_id = db.object\_id and oracle\_username = 'NIKOVITS';

ORACLE\_USERNAME SESSION\_ID LOCKED\_MODE OBJECT\_NAME OBJECT\_TYPE

----------------- ---------- ----------- ------------ -------------

NIKOVITS 16 3 TR\_PROBA TABLE