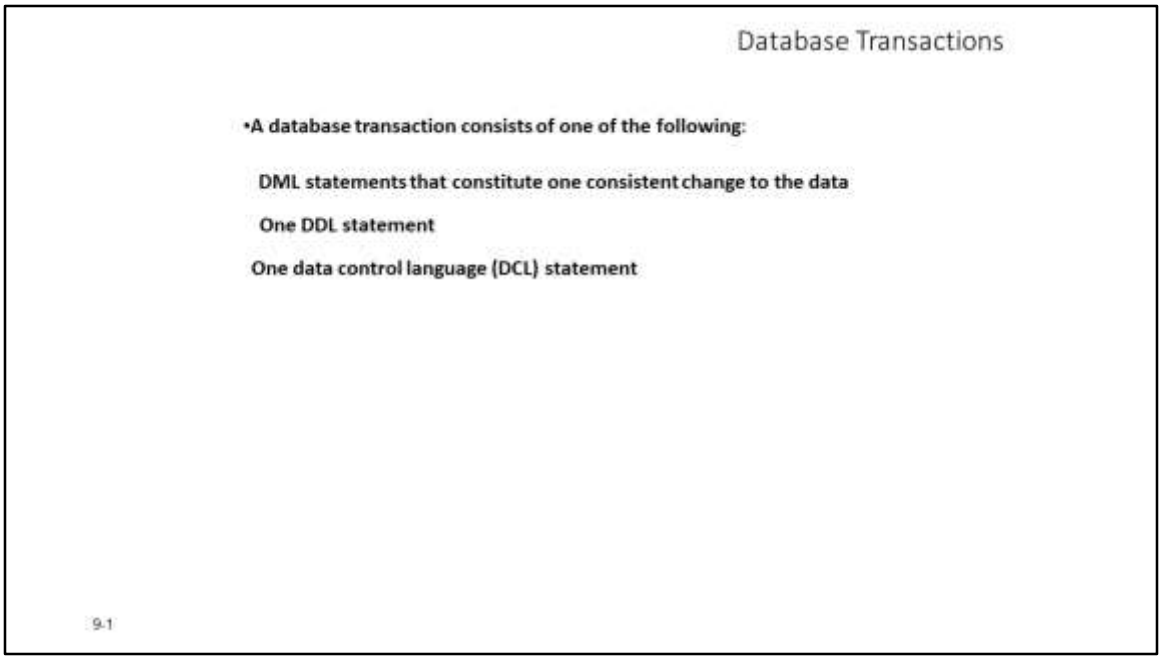


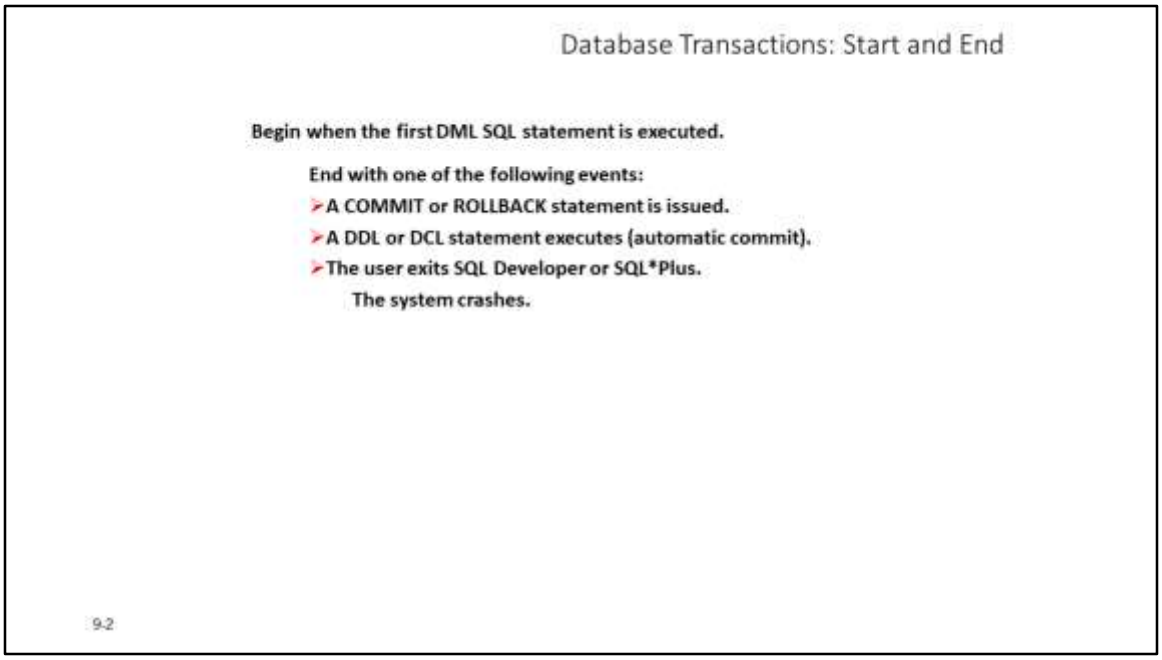
Database Transactions

The Oracle server ensures data consistency based on transactions. Transactions give you more flexibility and control when changing data, and they ensure data consistency in the event of user process failure or system failure.

Transactions consist of DML statements that constitute one consistent change to the data. For example, a transfer of funds between two accounts should include the debit in one account and the credit to another account of the same amount. Both actions should either fail or succeed together; the credit should not be committed without the debit.

Transaction Types

Type	Description
Data manipulation language (DML)	Consists of any number of DML statements that the Oracle server treats as a single entity or a logical unit of work
Data definition language (DDL)	Consists of only one DDL statement
Data control language (DCL)	Consists of only one DCL statement



Database Transaction: Start and End

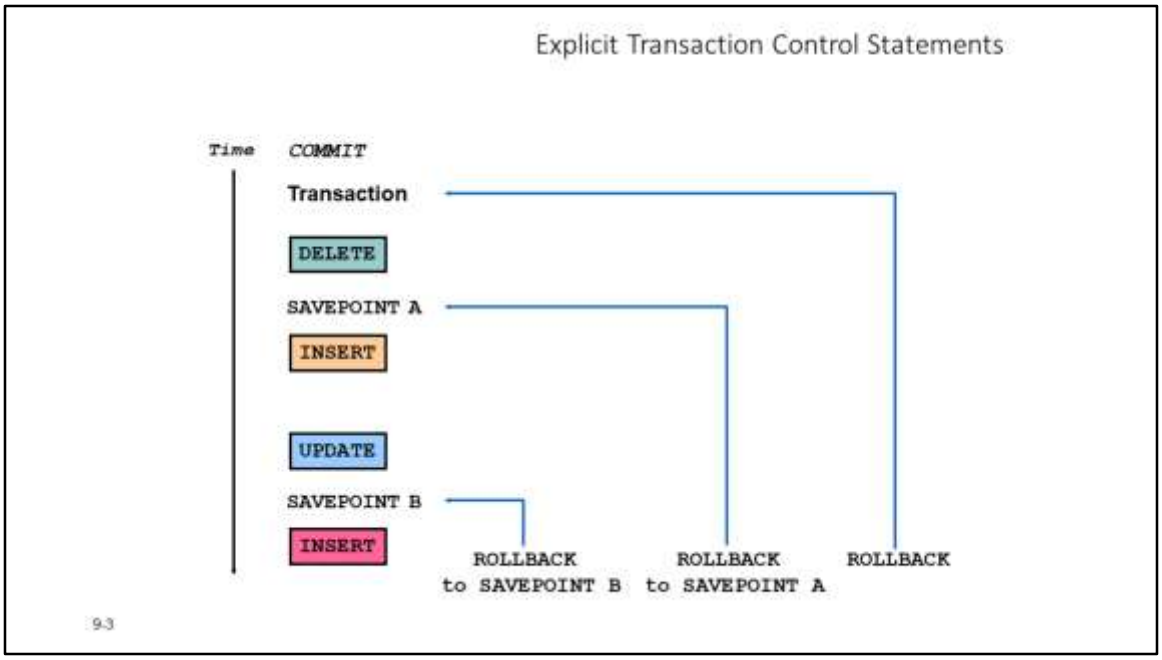
When does a database transaction start and end?

A transaction begins when the first DML statement is encountered and ends when one of the following occurs:

- A `COMMIT` or `ROLLBACK` statement is issued.
- A DDL statement, such as `CREATE`, is issued.
- A DCL statement is issued.
- The user exits SQL Developer or SQL*Plus.
- A machine fails or the system crashes.

After one transaction ends, the next executable SQL statement automatically starts the next transaction.

A DDL statement or a DCL statement is automatically committed and, therefore, implicitly ends a transaction.

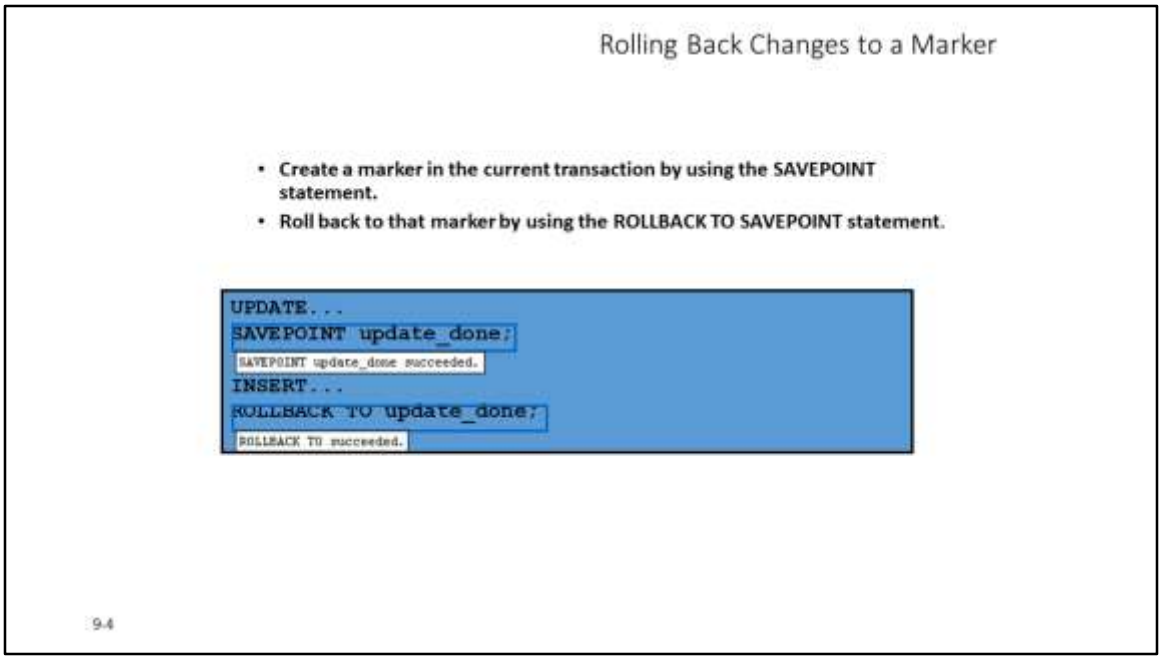


Explicit Transaction Control Statements

You can control the logic of transactions by using the `COMMIT`, `SAVEPOINT`, and `ROLLBACK` statements.

Statement	Description
`COMMIT`	`COMMIT` ends the current transaction by making all pending data changes permanent.
`SAVEPOINT` *name*	`SAVEPOINT` *name* marks a savepoint within the current transaction.
`ROLLBACK`	`ROLLBACK` ends the current transaction by discarding all pending data changes.
`ROLLBACK TO` *SAVEPOINT name*	`ROLLBACK TO SAVEPOINT` rolls back the current transaction to the specified savepoint, thereby discarding any changes and/or savepoints that were created after the savepoint to which you are rolling back. If you omit the `TO SAVEPOINT` clause, the `ROLLBACK` statement rolls back the entire transaction. Because savepoints are logical, there is no way to list the savepoints that you have created.

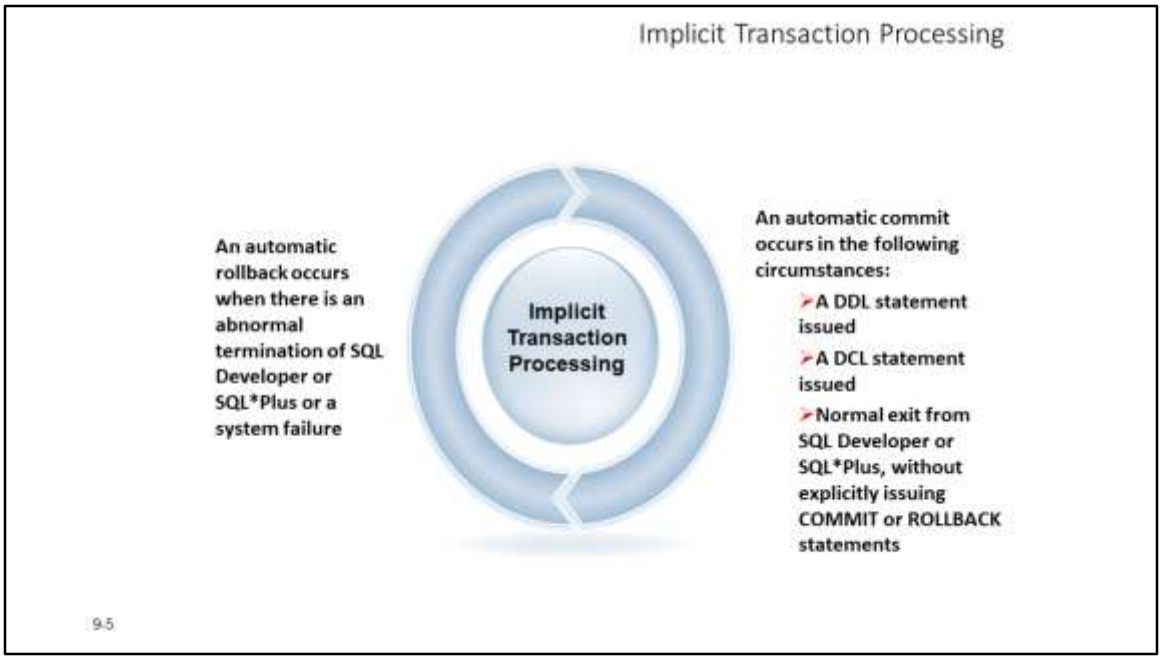
Note: You cannot `COMMIT` to a `SAVEPOINT`. `SAVEPOINT` is not ANSI-standard SQL.



Rolling Back Changes to a Marker

You can create a marker in the current transaction by using the `SAVEPOINT` statement, which divides the transaction into smaller sections. You can then discard pending changes up to that marker by using the `ROLLBACK TO SAVEPOINT` statement.

Note that if you create a second savepoint with the same name as an earlier savepoint, the earlier savepoint is deleted.



Implicit Transaction Processing

Status	Circumstances
Automatic commit	DDL statement or DCL statement issued SQL Developer or SQL*Plus exited normally, without explicitly issuing `COMMIT` or `ROLLBACK` commands
Automatic rollback	Abnormal termination of SQL Developer or SQL*Plus or system failure

Note: In SQL*Plus, the `AUTOCOMMIT` command can be toggled ON or OFF. If set to ON, each individual DML statement is committed as soon as it is executed. You cannot roll back the changes. If set to OFF, the `COMMIT` statement can still be issued explicitly. Also, the `COMMIT` statement is issued when a DDL statement is issued or when you exit SQL*Plus. The `SET AUTOCOMMIT ON/OFF` command is skipped in SQL Developer. DML is committed on a normal exit from SQL Developer only if you have the Autocommit preference enabled.

To enable Autocommit, perform the following:

In the Tools menu, select Preferences. In the Preferences dialog box, expand Database and select Worksheet Parameters.

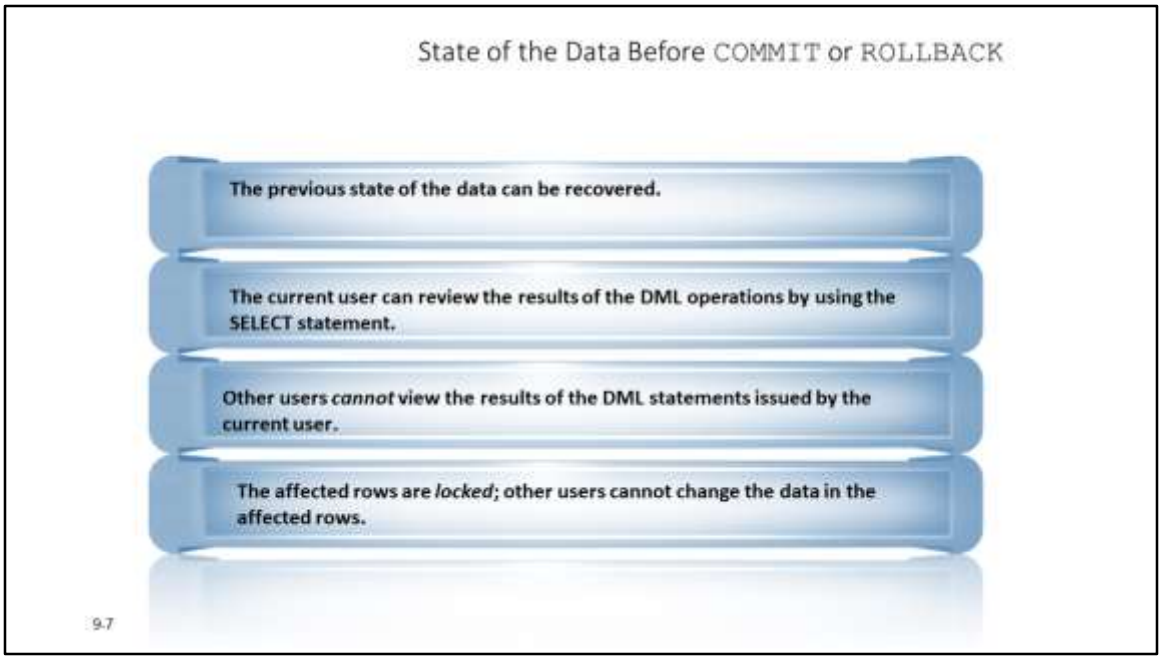
In the right pane, select the “Autocommit in SQL Worksheet” option. Click OK.

Implicit Transaction Processing (continued)

System Failures

When a transaction is interrupted by a system failure, the entire transaction is automatically rolled back. This prevents the error from causing unwanted changes to the data and returns the tables to the state at the time of the last commit. In this way, the Oracle server protects the integrity of the tables.

In SQL Developer, a normal exit from the session is accomplished by selecting Exit from the File menu. In SQL*Plus, a normal exit is accomplished by entering the `EXIT` command at the prompt. Closing the window is interpreted as an abnormal exit.

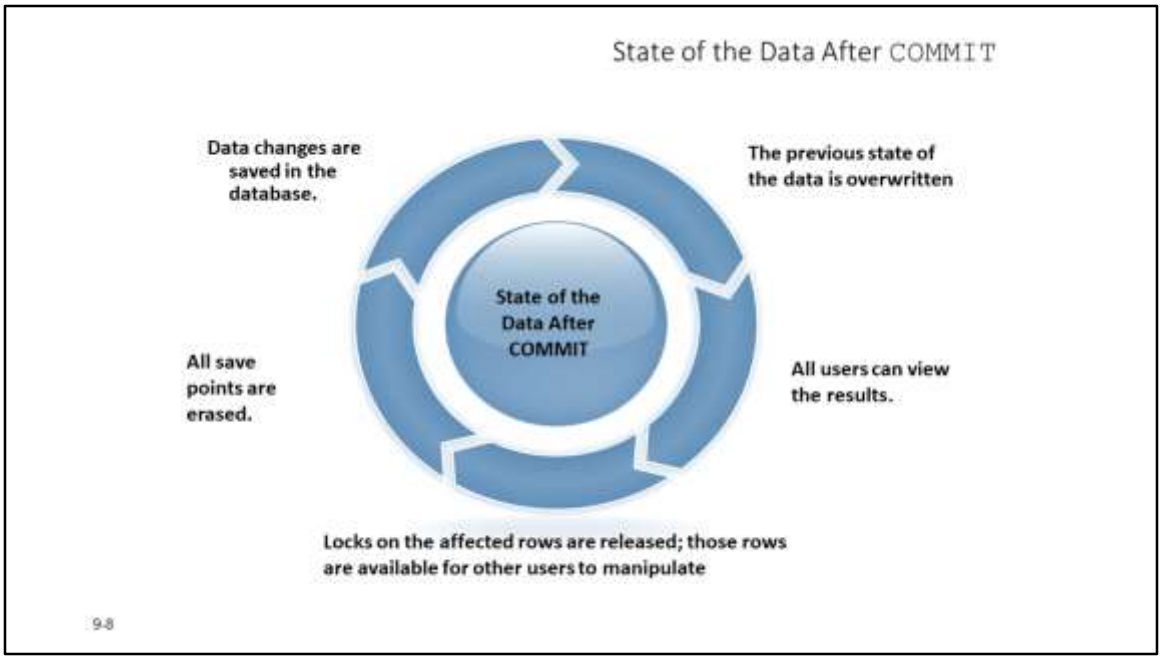


State of the Data Before `COMMIT` or `ROLLBACK`

Every data change made during the transaction is temporary until the transaction is committed.

The state of the data before `COMMIT` or `ROLLBACK` statements are issued can be described as follows:

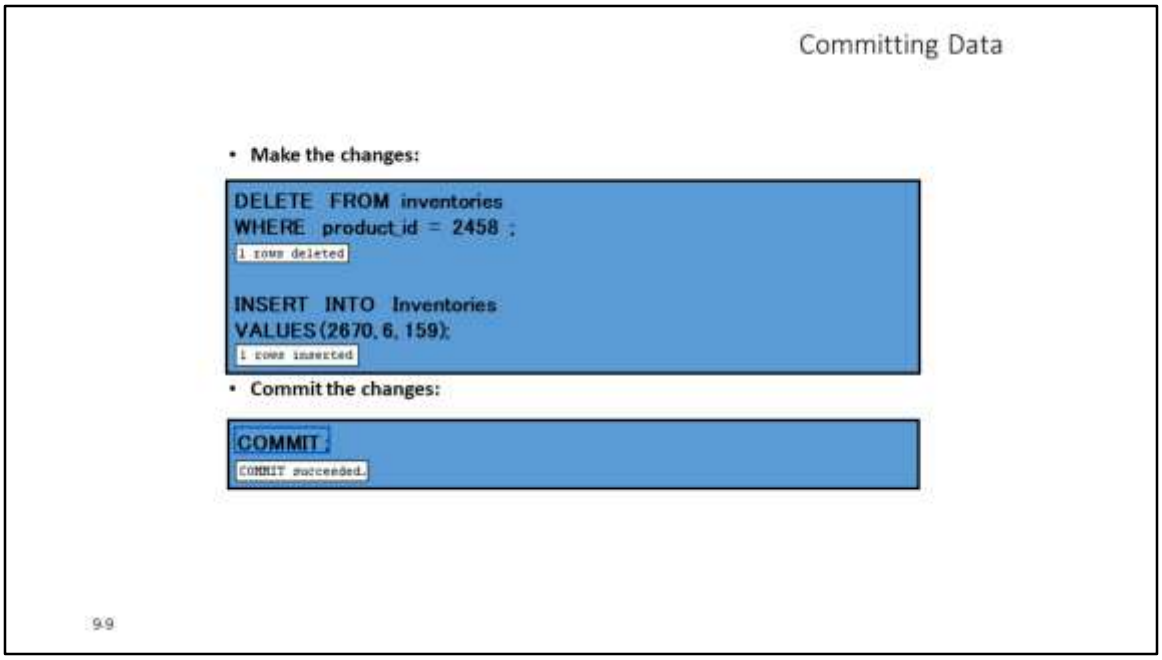
- Data manipulation operations primarily affect the database buffer; therefore, the previous state of the data can be recovered.
- The current user can review the results of the data manipulation operations by querying the tables.
- Other users cannot view the results of the data manipulation operations made by the current user. The Oracle server institutes read consistency to ensure that each user sees data as it existed at the last commit.
- The affected rows are locked; other users cannot change the data in the affected rows.



State of the Data After `COMMIT`

Make all pending changes permanent by using the `COMMIT` statement. Here is what happens after a `COMMIT` statement:

- Data changes are written to the database.
- The previous state of the data is no longer available with normal SQL queries.
- All users can view the results of the transaction.
- The locks on the affected rows are released; the rows are now available for other users to perform new data changes.
- All savepoints are erased.



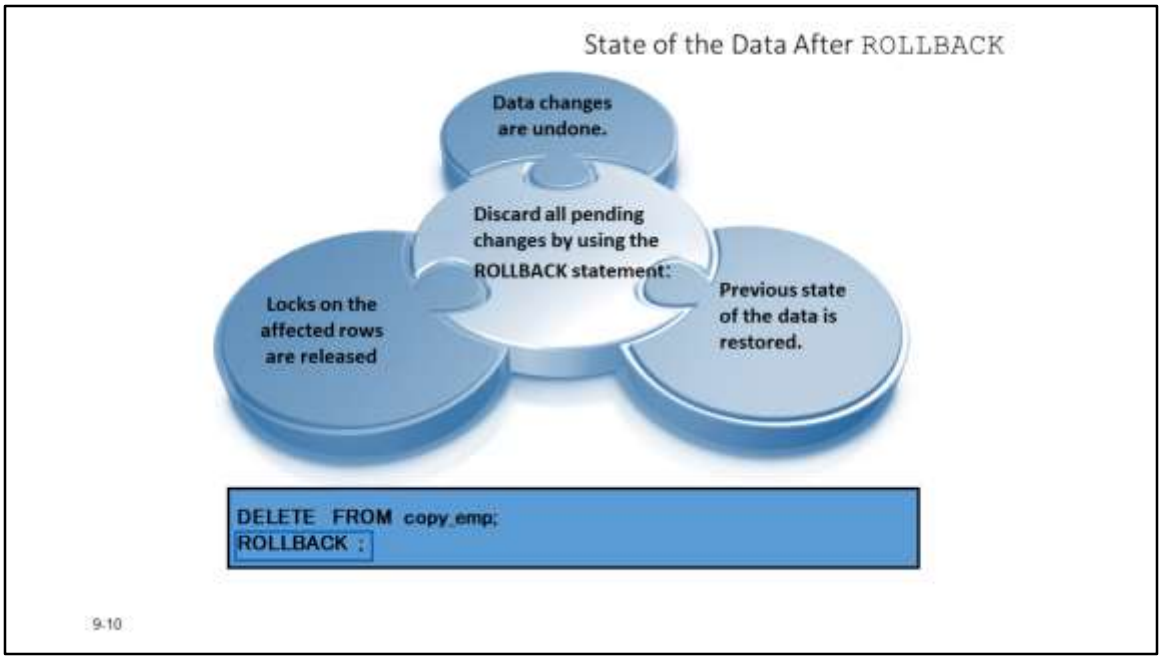
Committing Data

In the example in the slide, a row is deleted from the `INVENTORIES` table and a new row is inserted into the INVENTORIES table. The changes are saved by issuing the `COMMIT` statement.

Example:

Remove departments 290 and 300 in the `DEPARTMENTS` table and update a row in the `EMPLOYEES` table. Save the data change.

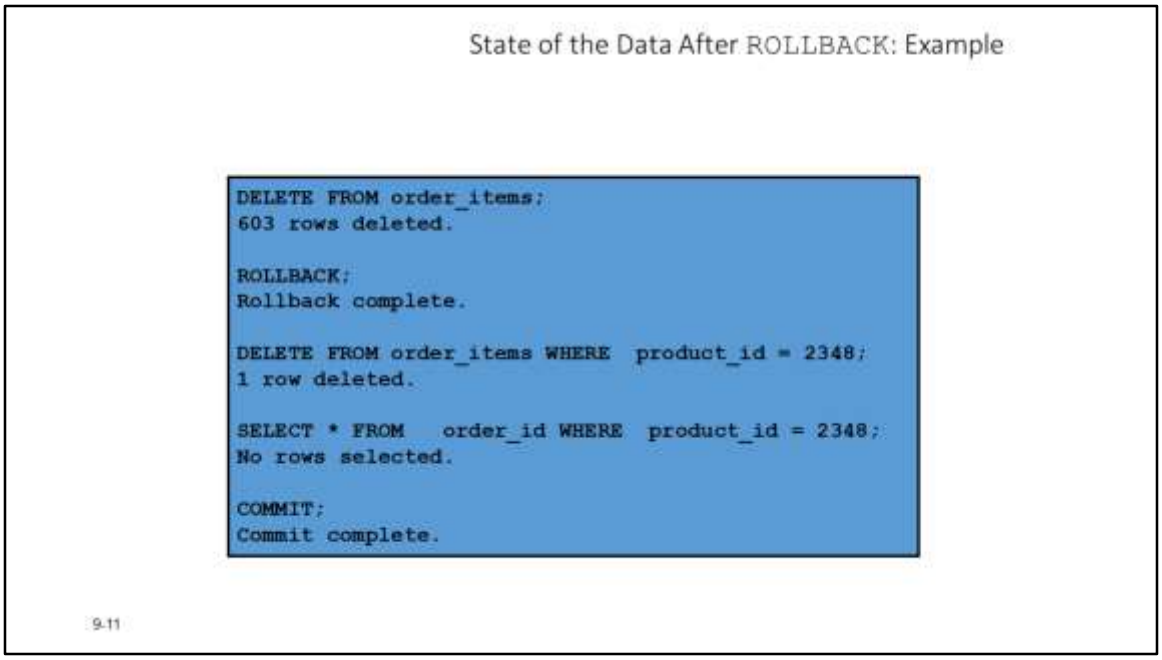
```
DELETE FROM departments
WHERE  department_id IN (290, 300);

UPDATE  employees
  SET   department_id = 80
  WHERE employee_id = 206;

COMMIT;
```



State of the Data After `ROLLBACK`

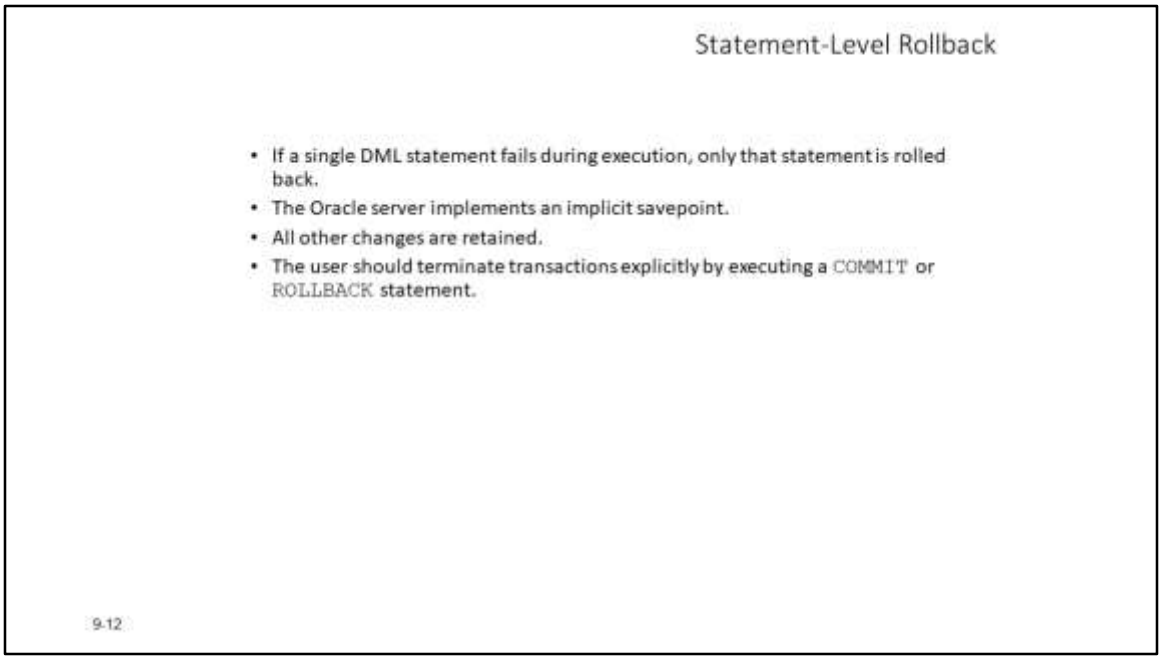
Discard all pending changes by using the `ROLLBACK` statement, which results in the following:

- Data changes are undone.
- The previous state of the data is restored.
- Locks on the affected rows are released.



State of the Data After ROLLBACK: Example

While attempting to remove a record from the TEST table, you may accidentally empty the table. However, you can correct the mistake, reissue a proper statement, and make the data change permanent.

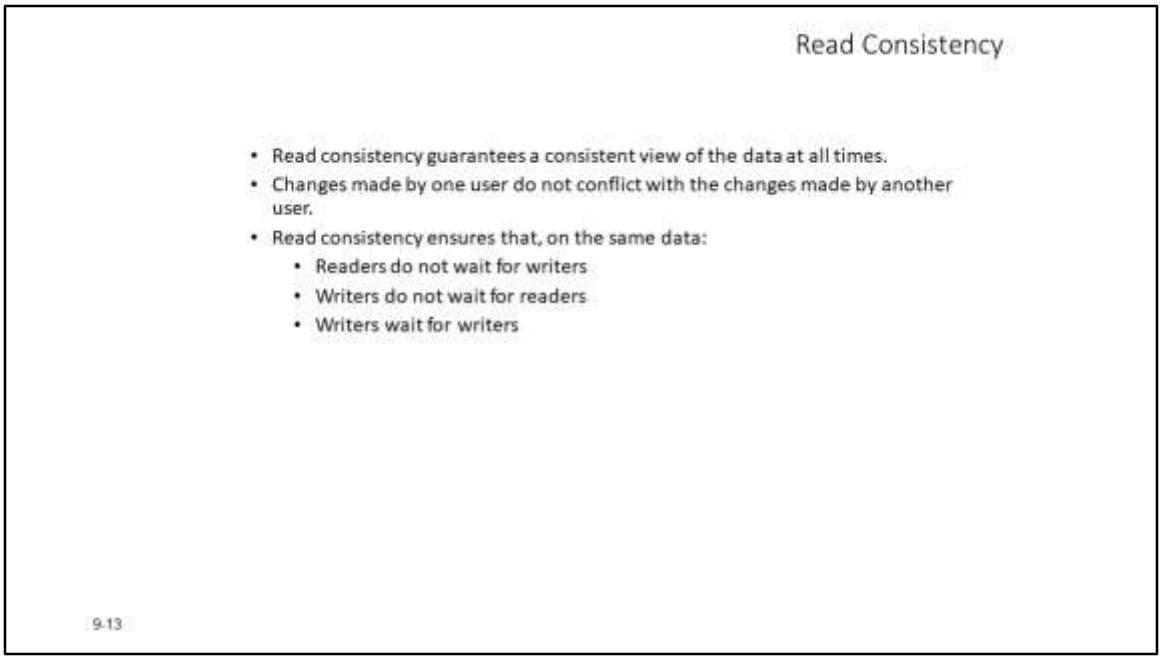


Statement-Level Rollback

A part of a transaction can be discarded through an implicit rollback if a statement execution error is detected. If a single DML statement fails during execution of a transaction, its effect is undone by a statement-level rollback, but the changes made by the previous DML statements in the transaction are not discarded. They can be committed or rolled back explicitly by the user.

The Oracle server issues an implicit commit before and after any DDL statement. So, even if your DDL statement does not execute successfully, you cannot roll back the previous statement because the server issued a commit.

Terminate your transactions explicitly by executing a `COMMIT` or `ROLLBACK` statement.



Read Consistency

Database users access the database in two ways:

Read operations (`SELECT` statement)

Write operations (`INSERT`, `UPDATE`, `DELETE` statements)

You need read consistency so that the following occur:

The database reader and writer are ensured a consistent view of the data.

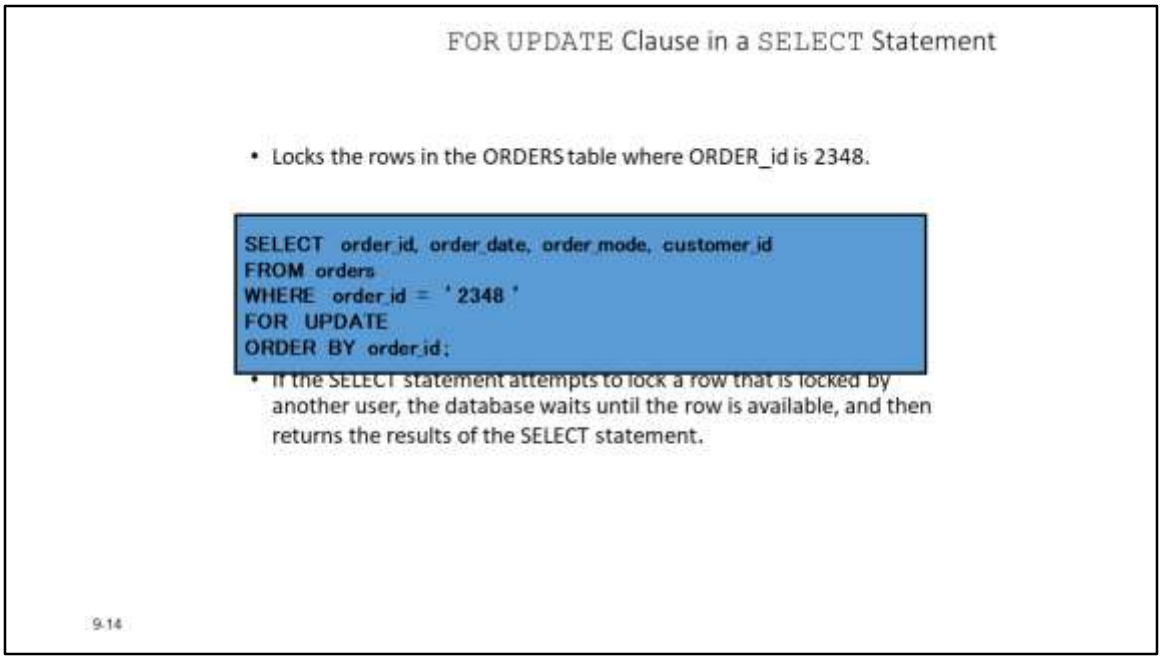
Readers do not view data that is in the process of being changed.

Writers are ensured that the changes to the database are done in a consistent manner.

Changes made by one writer do not disrupt or conflict with the changes being made by another writer.

The purpose of read consistency is to ensure that each user sees data as it existed at the last commit, before a DML operation started.

Note: The same user can log in to different sessions. Each session maintains read consistency in the manner described above, even if they are the same users.



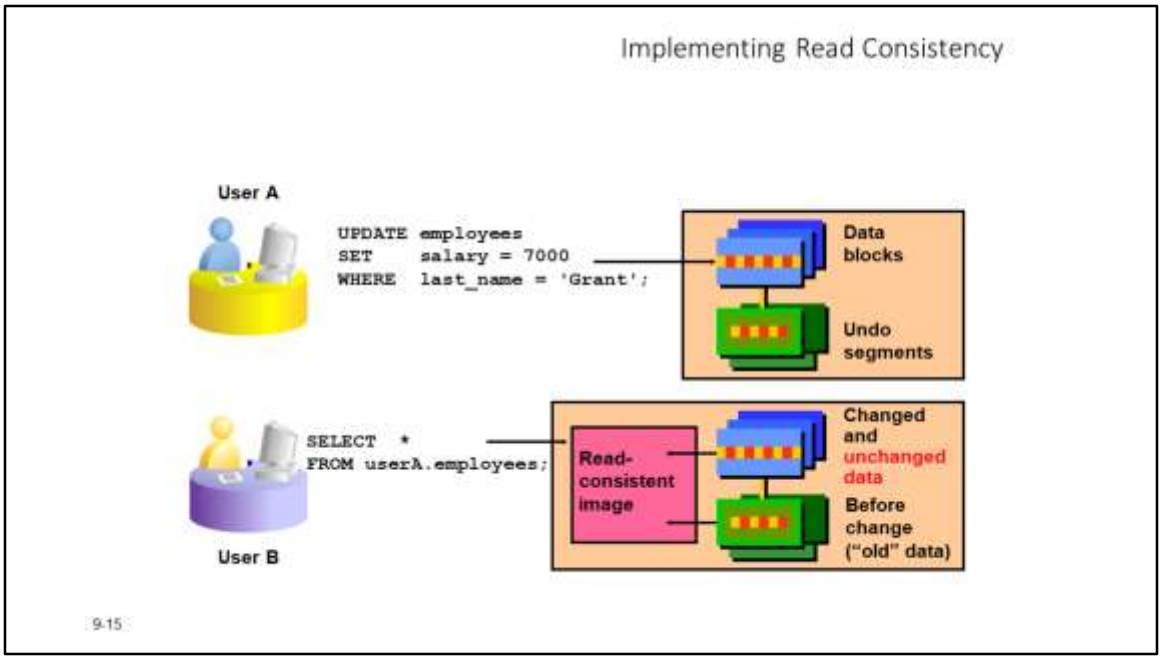
FOR UPDATE Clause in a SELECT Statement

When you issue a SELECT statement against the database to query some records, no locks are placed on the selected rows. In general, this is required because the number of records locked at any given time is (by default) kept to the absolute minimum: only those records that have been changed but not yet committed are locked. Even then, others will be able to read those records as they appeared before the change (the "before image" of the data). There are times, however, when you may want to lock a set of records even before you change them in your program. Oracle offers the FOR UPDATE clause of the SELECT statement to perform this locking.

When you issue a SELECT . . . FOR UPDATE statement, the relational database management system (RDBMS) automatically obtains exclusive row-level locks on all the rows identified by the SELECT statement, thereby holding the records "for your changes only." No one else will be able to change any of these records until you perform a ROLLBACK or a COMMIT.

You can append the optional keyword NOWAIT to the FOR UPDATE clause to tell the Oracle server not to wait if the table has been locked by another user. In this case, control will be returned immediately to your program or to your SQL Developer environment so that you can perform other work, or simply wait for a

period of time before trying again. Without the `NOWAIT` clause, your process will block until the table is available, when the locks are released by the other user through the issue of a `COMMIT` or a `ROLLBACK` command.



Implementing Read Consistency

Read consistency is an automatic implementation. It keeps a partial copy of the database in the undo segments. The read-consistent image is constructed from the committed data in the table and the old data that is being changed and is not yet committed from the undo segment.

When an insert, update, or delete operation is made on the database, the Oracle server takes a copy of the data before it is changed and writes it to an *undo segment*.

All readers, except the one who issued the change, see the database as it existed before the changes started; they view the undo segment's "snapshot" of the data.

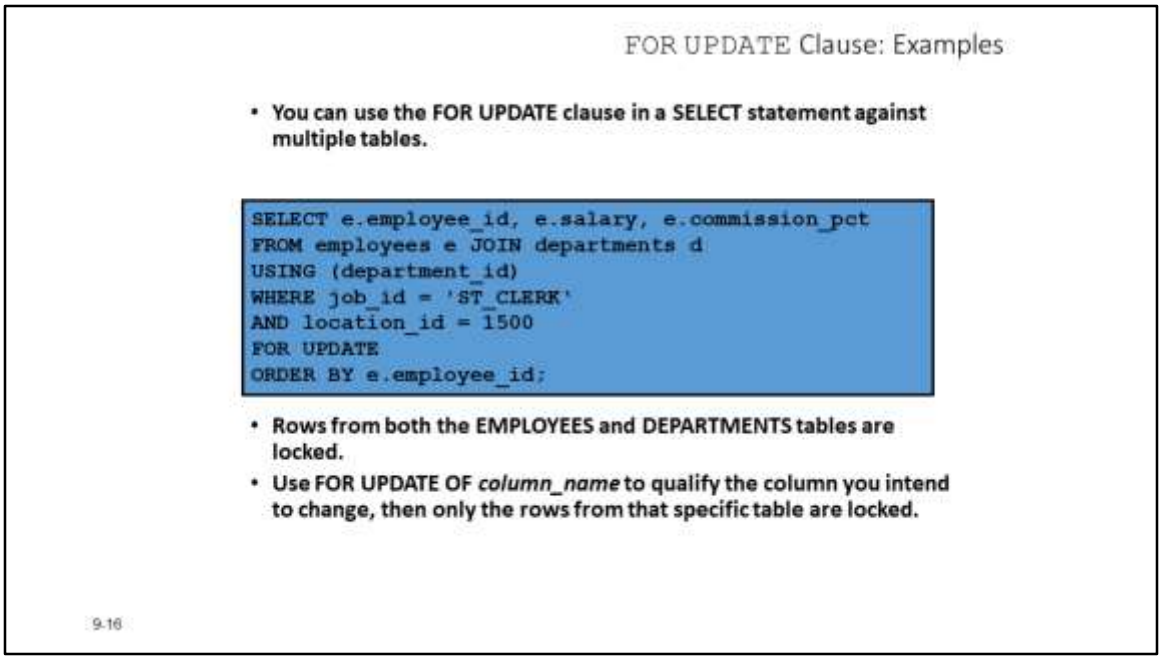
Before the changes are committed to the database, only the user who is modifying the data sees the database with the alterations. Everyone else sees the snapshot in the undo segment. This guarantees that readers of the data read consistent data that is not currently undergoing change.

When a DML statement is committed, the change made to the database becomes visible to anyone issuing a `SELECT` statement *after* the commit is done. The space occupied by the *old* data in the undo segment file is freed for reuse.

If the transaction is rolled back, the changes are undone:

The original, older version of the data in the undo segment is written back to the table.

All users see the database as it existed before the transaction began.



FOR UPDATE Clause: Examples

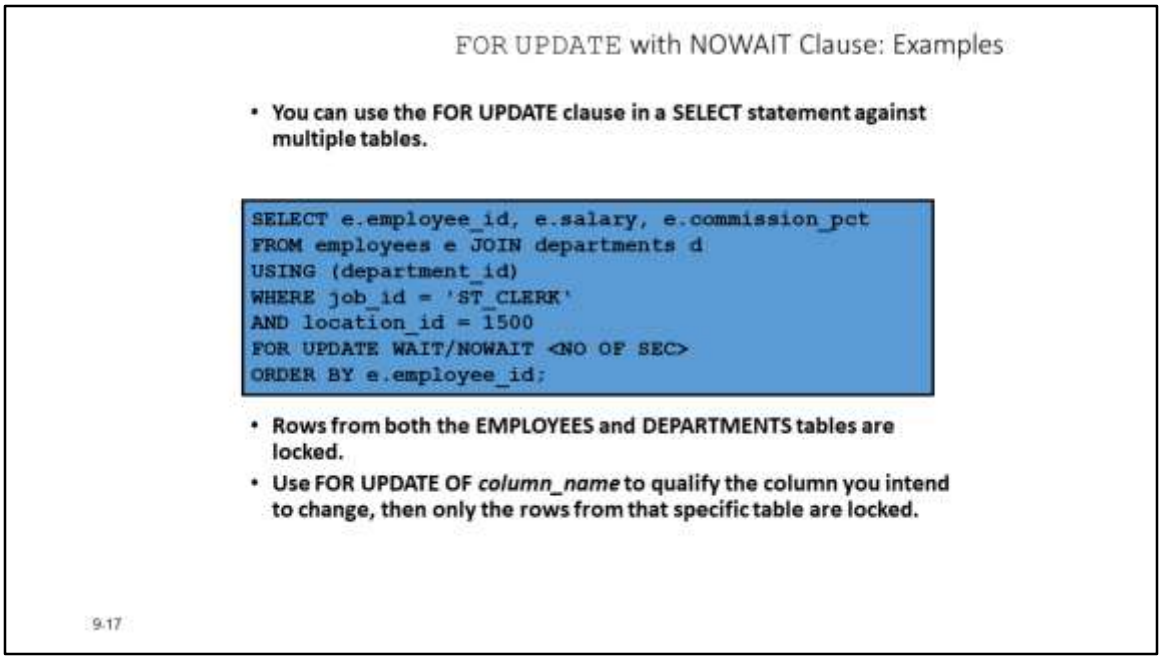
In the example in the slide, the statement locks rows in the EMPLOYEES table with JOB_ID set to ST_CLERK and LOCATION_ID set to 1500, and locks rows in the DEPARTMENTS table with departments in LOCATION_ID set as 1500.

You can use the FOR UPDATE OF *column_name* to qualify the column that you intend to change. The OF list of the FOR UPDATE clause does not restrict you to changing only those columns of the selected rows. Locks are still placed on all rows; if you simply state FOR UPDATE in the query and do not include one or more columns after the OF keyword, the database will lock all identified rows across all the tables listed in the FROM clause.

The following statement locks only those rows in the EMPLOYEES table with ST_CLERK located in LOCATION_ID 1500. No rows are locked in the DEPARTMENTS table:

```
SELECT e.employee_id, e.salary, e.commission_pct
FROM employees e JOIN departments d
USING (department_id)
WHERE job_id = 'ST_CLERK' AND location_id = 1500
FOR UPDATE OF e.salary
```

```
ORDER BY e.employee_id;
```



`FOR UPDATE with NOWAIT` Clause: Examples

This clause will help the user to wait for specific number of seconds.
The default option is WAIT.