Commits and Savepoints

Objectives

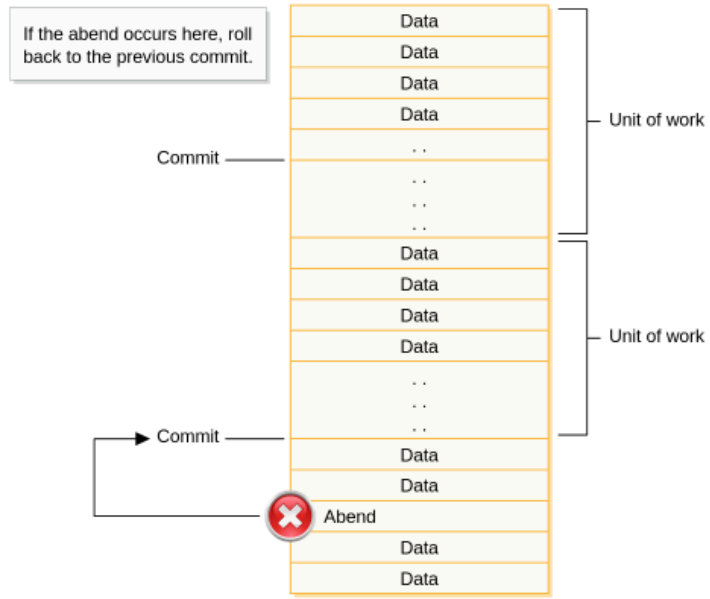
Commits and Savepoints

When application programs are updating Db2 databases, they can fail or require that some updates be undone.

This module examines how to set points in a program to recover to. You will discover how Db2 keeps track of these points and the updates that are performed. You will also see how to make Db2 roll back to those points.

After completing this module you will be able to:

- Define and Use Commit Points, and Understand Their Effect on Cursors
- Define Db2 Rollbacks and How to Force One
- Define and Use Savepoints

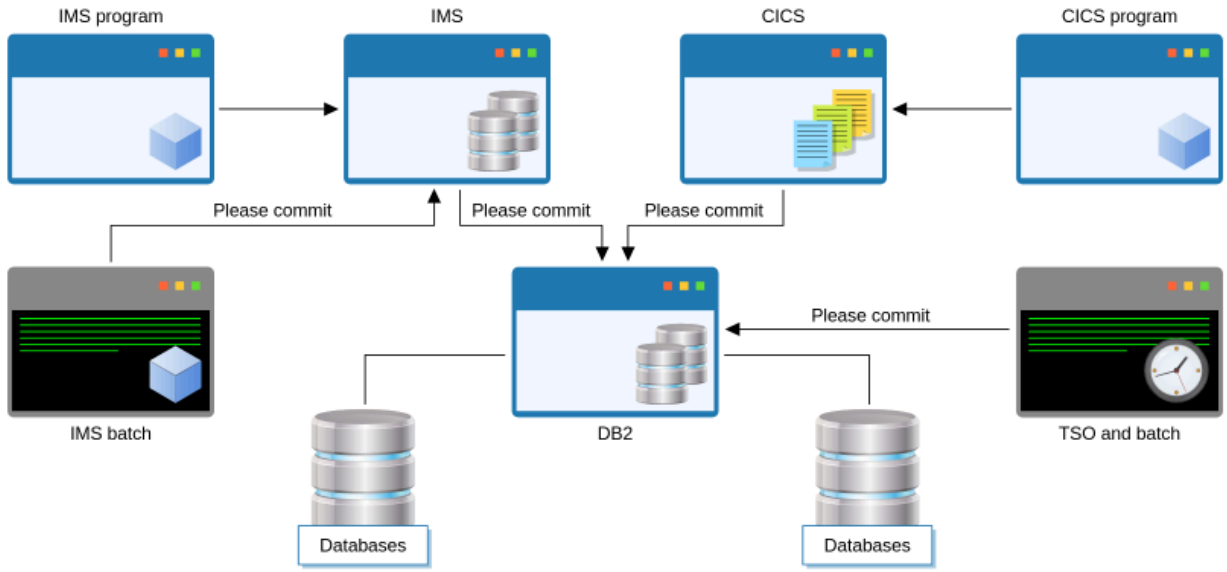


In most database management systems, changes made to data are not considered to be final until a commit point is reached.

This ensures that the data can be restored to a point prior to the updates if a program abends. The frequency of commits depends on the type of application program.

A batch program may issue a commit every 1000 updates or after a certain period of time has passed.

An online program will probably commit every time the user starts a new transaction, for example, a new customer or product.



The way in which a commit point is initiated depends on the environment the program is running in and what type it is. Some of the possible environments are:

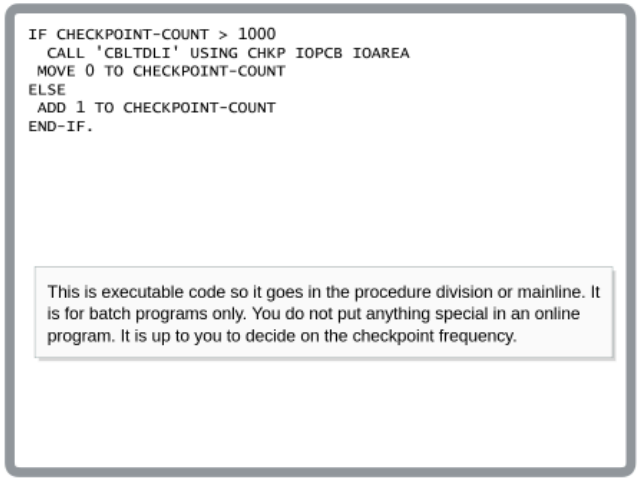
- Called from a REXX exec
- A batch job
- An IMS BMP or DLI program
- An IMS online transaction
- A CICS transaction

```
/* REXX DB2 PROGRAM */
. . .
. . .
/* Establish DB2 communication */
SSID = "DB11"
ADDRESS DSNREXX "CONNECT" SSID
. . .
. . .

/* ****************************************** */
/* ** Execute SQL and Commit on 50th update ** */
/* ****************************************** */
ADDRESS DSNREXX
"EXECSQL EXECUTE S1 USING :HV1,:HV2 "
  IF SQLCODE \= "0" THEN DO
   SAY "Execution Failure "
   CALL DBERROR
   EXIT
  END
   IF COMMCOUNT > 50 THEN DO
   ADDRESS DSNREXX "EXECSQL COMMIT"
. . .
. . .
```

With programs executed from REXX or run in batch, commits are performed by using the COMMIT SQL statement shown in the example. This can only be used when the program is running under the DSN command processor under TSO.

Note: You cannot call a Db2 program from an EXEC. You must connect to Db2 first with DSN and then run the program.

In batch, TSO attaches to Db2 first which then runs the program.



In a batch IMS program, CHKP is used to commit any database changes and as part of the process, establishes a checkpoint from which you can restart your program if need be. If your program failed, then on restart, an XRST call can obtain checkpoint status and determine whether any Db2 indoubt work units need to be resolved.

Within an IMS online program's unit of work, a commit point can occur under the following circumstances:

- During normal program termination
- When a checkpoint call or SYNC call is issued by the program
- When the program receives a new message - for programs processing messages as its input

```
EXEC CICS
  SYNCPOINT
END-EXEC
```

Under CICS the technique is to issue a SYNCPOINT command. This will force Db2 to commit.

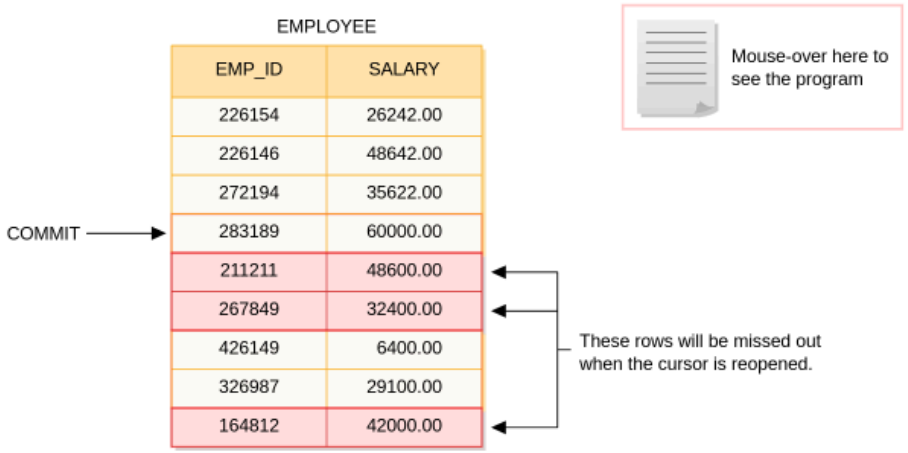
Committing too frequently can impact on the performance of your program so try to balance the frequency of commits against the time you have to roll back should an abend occur.

```
EXEC SQL
  DECLARE EMP_CUR CURSOR FOR
   SELECT EMP_ID, SALARY
   FROM EMPLOYEE
   WHERE EMP_ID > :EMP-ID
END-EXEC.

MOVE LOW-VALUES TO EMP-ID.

EXEC SQL
   OPEN EMP_CUR
END-EXEC.
   . . .
   . . .
IF CHECK-COUNT > 1000
   EXEC SQL
    COMMIT
   END-EXEC.
   EXEC SQL
    OPEN EMP_CUR
   END-EXEC.
   MOVE 0
  TO CHECK-COUNT
END-IF.
```

Any cursors that are open at the time of commit are lost after the commit has completed unless the WITH HOLD option is specified on the DECLARE CURSOR statement. This means that the cursor needs to be opened again in order to continue processing. This usually involves opening the cursor at the position after the last row that was retrieved.

Notice that the program shown here assumes that the rows are returned in order. If the rows are not returned in order, you could miss rows when the cursor is reopened.

Next you will see a demonstration of this.

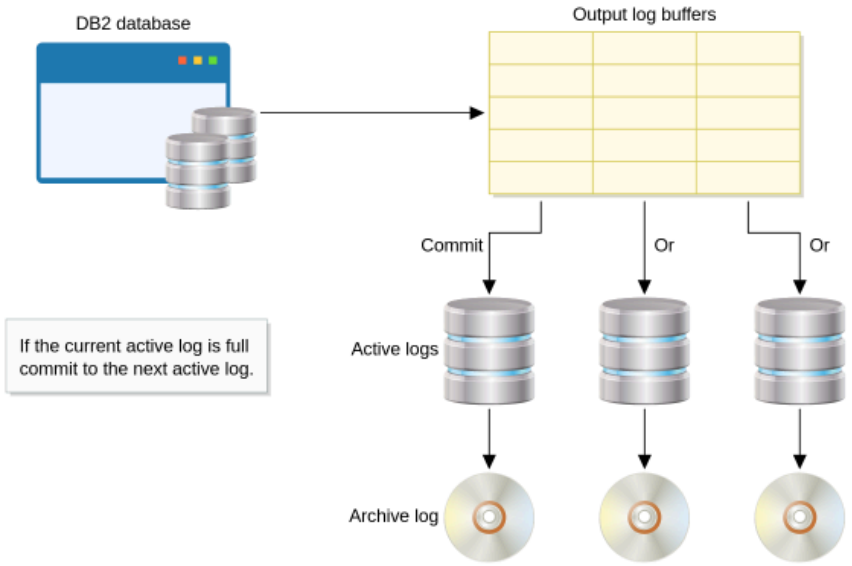


Imagine the program was performing on this table. The program would miss rows when the cursor was reopened.

One way to avoid missing rows is to use an ORDER BY to force the rows into the correct order. If the table is large, this can be a time-consuming operation.

The best solution is to ensure there is an index on the column and Db2 makes use of it when told to ORDER BY that column.

Mouse-over the table to see how rows can be missed through committing.

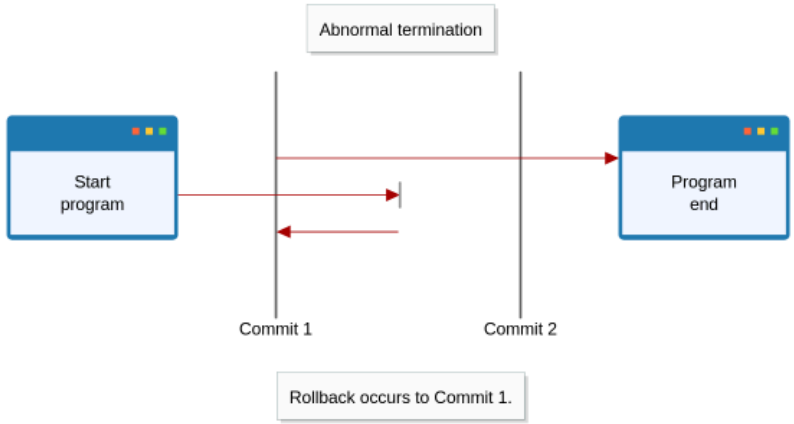


Now you are going to see how Db2 records changes and how changes are backed out on error or request.

All data changes are written to the Db2 output log buffers. Certain actions such as COMMIT will force the output log buffers to be externalized or written to the active logs on DASD.

When the active log fills up, it automatically switches to the next active log. Db2 will then copy the content of the full active log to an archive log that resides on tape or DASD.

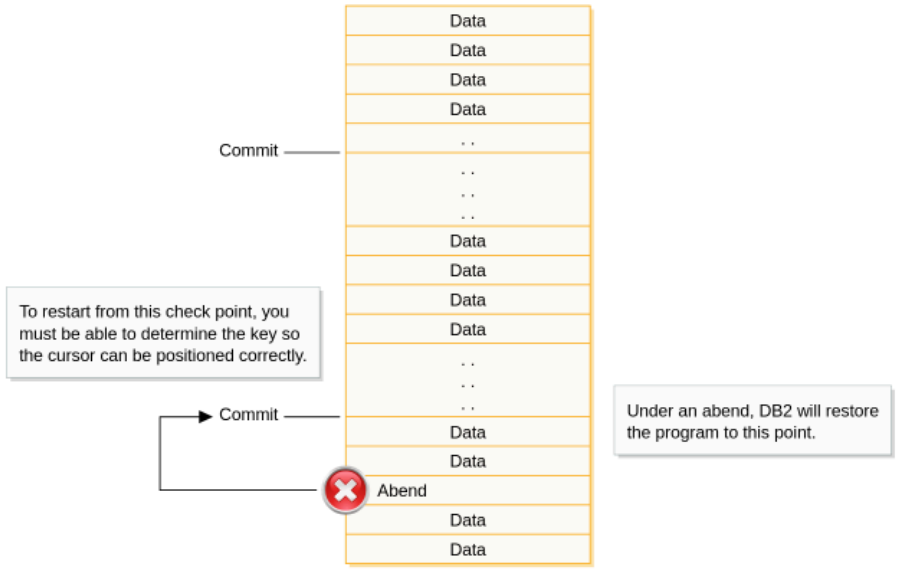
Click Play to see an example of how output log buffers are used.



Commit processing forces the log buffer contents to be written to the active logs. Consider a program that issues two COMMITS while running. If the program terminates abnormally, all data changes will be undone or rolled back to the last commit point. If your program does not take any commit points, all data changes will be undone when a failure occurs.

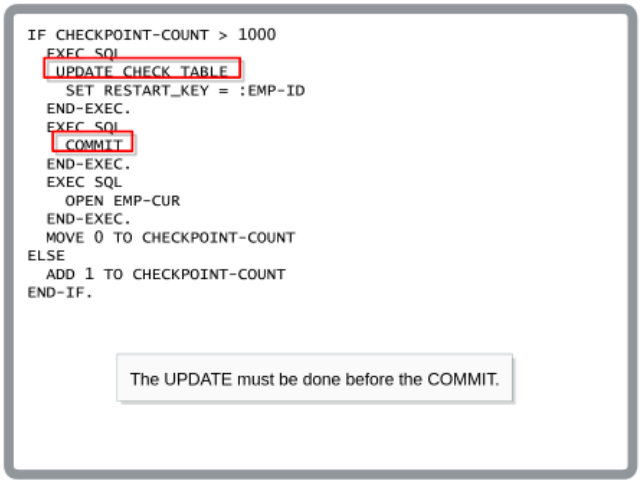
Normal program or online task termination is considered to be an IMPLICIT COMMIT. Abnormal program or task termination will roll back data changes to the last commit point, if one exists, or to the start of the program or task.

Click **Play** to see a demonstration of this.



It is important to note that after an abend or requested rollback, it can take twice as long to undo the changes than to actually make them. This is one reason why checkpoints are so important.

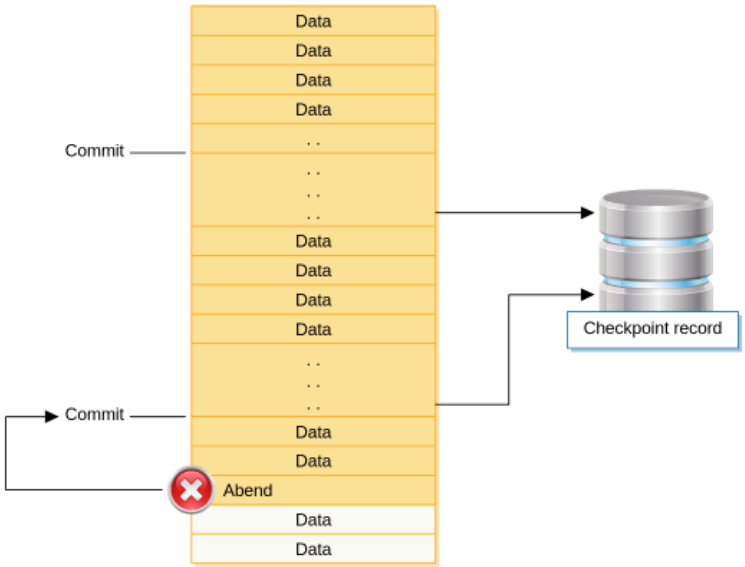
A program will roll back to the beginning if previous successful updates have not been marked with a checkpoint, but to restart from the last checkpoint taken, you must be able to retrieve the key of the row into the program so the cursor can be positioned correctly.



There are many ways of doing this. The method depends on the both the environment the program is running under and the logic of the update being undertaken.

If the program runs under TSO, the key values could be saved in another table at every checkpoint. On restart the program reads the table and uses these values to position the cursor. This is demonstrated in the program shown here.

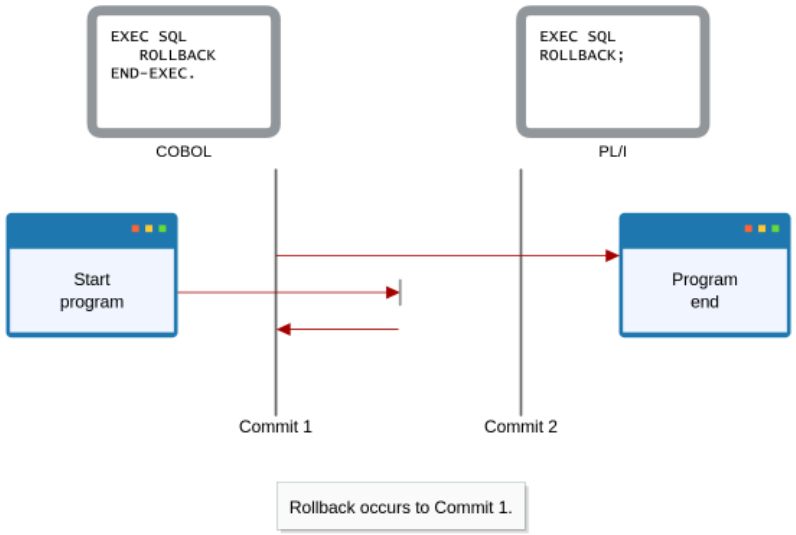
Under IMS, the XRST/CHKP save areas could be used.



Other methods include updating flags or audit records as each cursor row is updated, writing checkpoint information to external files, or using their facilities to record checkpoints when running under CICS or IMS.

The important thing is to be able to determine which tables and rows have been updated and committed and which have not.

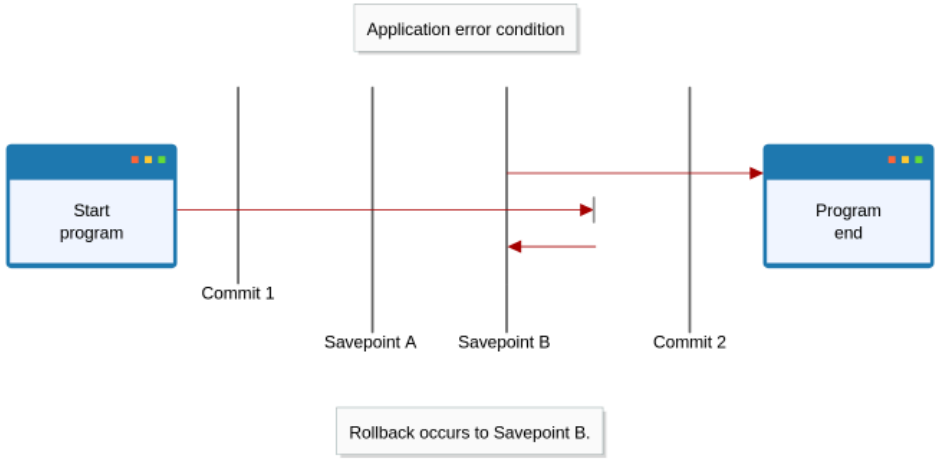
Click **Play** to see a demonstration of how the checkpoint record is used.



Not only program or system failure can cause a rollback. It can also be initiated from within application program code. This is achieved by using the SQL ROLLBACK statement. This functionality is often used in program logic when a condition that invalidates earlier updates is reached, or in online or interactive systems where a user cancels a transaction and updates must be ignored and not committed. Remember, normal program terminations in most environments, that is BATCH, TSO, CICS, and IMS, involve an implicit commit.

Note: Rollback is only available in CICS and IMS when used with save points. You will discover this next.

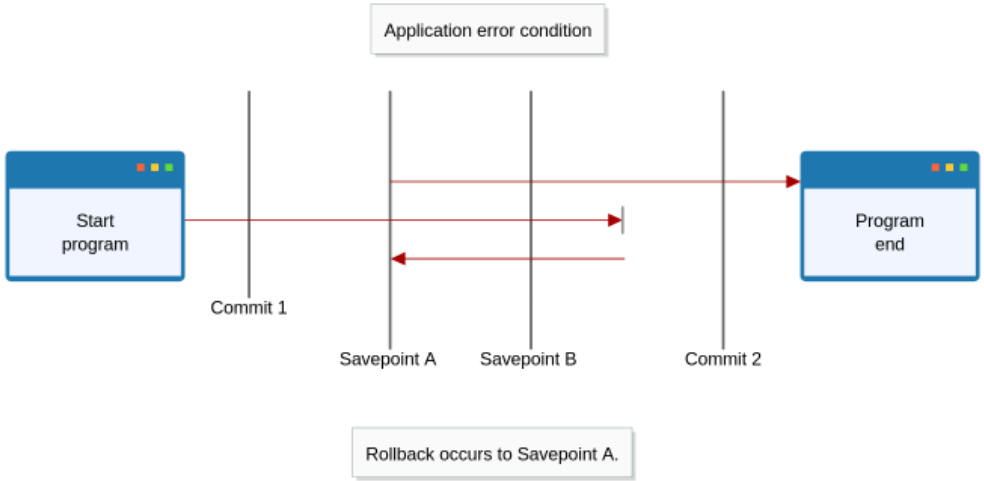
Click **Play** to see a demonstration of application program code initiating a rollback.



Savepoints introduce a smaller level of granularity in a unit of work. They allow you to undo selected changes within a unit of work or transaction.

Your application can set a savepoint within a transaction. Application logic can undo the data changes that were made since the application set the save point without affecting the overall outcome of the transaction.

Click **Play** to see how savepoints are used.



Using savepoints makes coding applications more flexible and gives the developer more control over the outcome of the transaction. For example, the application can decide to roll back to Savepoint A instead of Savepoint B.

Click Play to see how savepoints can be used.

```
EXEC SQL
SAVEPOINT SP_1
ON ROLLBACK RETAIN CURSORS
END-EXEC.

Application logic...

EXEC SQL
SAVEPOINT SP_1 UNIQUE
ON ROLLBACK RETAIN LOCKS
END-EXEC.
```

The name used for your savepoint can be anything, as long as it does not begin with SYS.

The same name can be reused in the same unit of recovery unless the UNIQUE keyword is specified. With the UNIQUE keyword an error will occur.

If a savepoint is created without the UNIQUE option, the next savepoint uses the same name and the previous or older savepoint is destroyed.

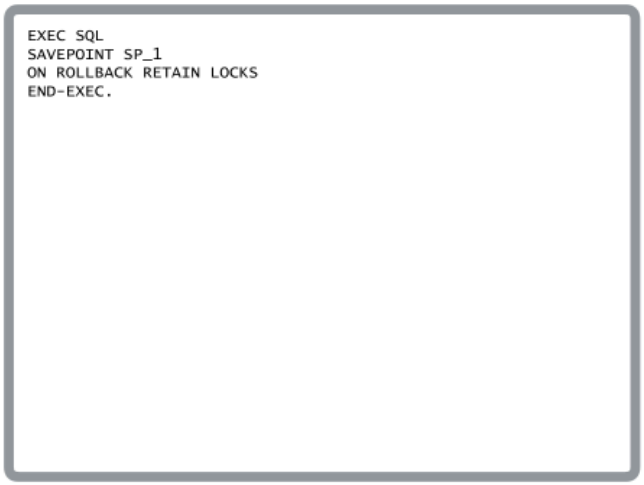
```
EXEC SQL
   SAVEPOINT A
END_EXEC.
...
EXEC SQL
   SAVEPOINT B
END_EXEC.
...
EXEC SQL
   SAVEPOINT C
END_EXEC.
...
EXEC SQL
   RELEASE SAVEPOINT C
END_EXEC.
....
```

Savepoints can be released using the RELEASE command.

At the end of the sequence of commands shown here, Db2 could roll back to SAVEPOINT A or SAVEPOINT B, but it no longer has a record of SAVEPOINT C.

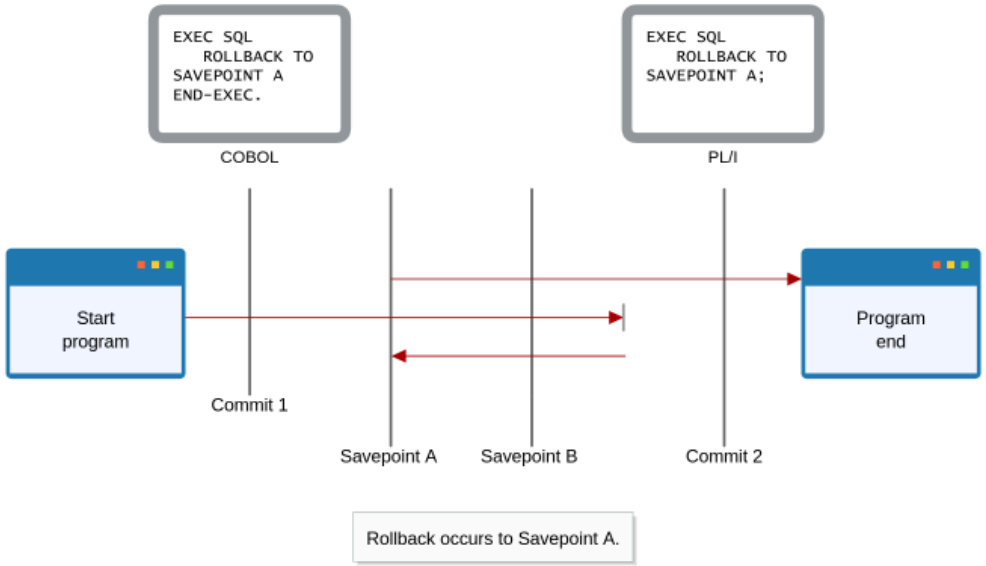
```
EXEC SQL
SAVEPOINT SP_1
ON ROLLBACK RETAIN CURSORS
END-EXEC.
```

In this example, any cursors that were opened after savepoint SP_1 will not be closed if a rollback to savepoint SP_1 is required.



In this example, any locks acquired after savepoint SP_1 will not be released if a rollback to savepoint SP_1 is required. ON ROLLBACK RETAIN LOCKS is the default if nothing is specified.

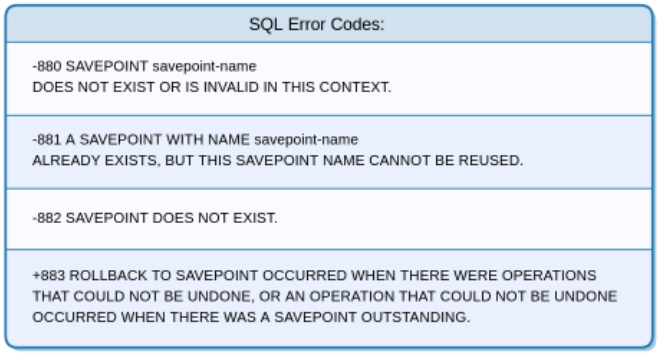
Note that Db2 will automatically release locks following a commit operation, or when the program ends.



You can write a ROLLBACK TO SAVEPOINT statement with or without a savepoint name.

If you do not specify a savepoint name, Db2 rolls back work to the most recently created savepoint. If you specify a savepoint name, Db2 will roll back to that savepoint.

Click Play to see a demonstration of a ROLLBACK TO SAVEPOINT.



Occasionally you may encounter an error associated with rolling back to a savepoint. The SQL codes above indicate some of the more common issues. The negative SQL error codes indicate that the save point was not successfully set (-880 and -881) or that the requested ROLLBACK TO SAVEPOINT was not performed (-882).

The warning code (+883) indicates that the rollback was performed but there are conditions that should be checked.

As usual, a SQLCODE of zero indicates a successful completion.