NAME

DECLARE − define a cursor

SYNOPSIS

DECLARE *name* [BINARY] [ASENSITIVE | INSENSITIVE] [[NO] SCROLL]
CURSOR [{ WITH | WITHOUT } HOLD] FOR *query*

DESCRIPTION

DECLARE allows a user to create cursors, which can be used to retrieve a small number of rows at a time out of a larger query. After the cursor is created, rows are fetched from it using **FETCH**.

Note

This page describes usage of cursors at the SQL command level. If you are trying to use cursors inside a PL/pgSQL function, the rules are different — see Section 41.7.

PARAMETERS

name

The name of the cursor to be created. This must be different from any other active cursor name in the session.

BINARY

Causes the cursor to return data in binary rather than in text format.

ASENSITIVE
INSENSITIVE

Cursor sensitivity determines whether changes to the data underlying the cursor, done in the same transaction, after the cursor has been declared, are visible in the cursor. INSENSITIVE means they are not visible, ASENSITIVE means the behavior is implementation−dependent. A third behavior, SENSITIVE, meaning that such changes are visible in the cursor, is not available in PostgreSQL. In PostgreSQL, all cursors are insensitive; so these key words have no effect and are only accepted for compatibility with the SQL standard.

Specifying INSENSITIVE together with FOR UPDATE or FOR SHARE is an error.

SCROLL
NO SCROLL

SCROLL specifies that the cursor can be used to retrieve rows in a nonsequential fashion (e.g., backward). Depending upon the complexity of the query's execution plan, specifying SCROLL might impose a performance penalty on the query's execution time. NO SCROLL specifies that the cursor cannot be used to retrieve rows in a nonsequential fashion. The default is to allow scrolling in some cases; this is not the same as specifying SCROLL. See Notes below for details.

WITH HOLD
WITHOUT HOLD

WITH HOLD specifies that the cursor can continue to be used after the transaction that created it successfully commits. WITHOUT HOLD specifies that the cursor cannot be used outside of the transaction that created it. If neither WITHOUT HOLD nor WITH HOLD is specified, WITHOUT HOLD is the default.

query

A **SELECT** or **VALUES** command which will provide the rows to be returned by the cursor.

The key words ASENSITIVE, BINARY, INSENSITIVE, and SCROLL can appear in any order.

NOTES

Normal cursors return data in text format, the same as a **SELECT** would produce. The BINARY option specifies that the cursor should return data in binary format. This reduces conversion effort for both the server and client, at the cost of more programmer effort to deal with platform−dependent binary data formats. As an example, if a query returns a value of one from an integer column, you would get a string of 1 with a default cursor, whereas with a binary cursor you would get a 4−byte field containing the internal representation of the value (in big−endian byte order).

Binary cursors should be used carefully. Many applications, including psql, are not prepared to handle binary cursors and expect data to come back in the text format.

Note

When the client application uses the “extended query” protocol to issue a **FETCH** command, the Bind protocol message specifies whether data is to be retrieved in text or binary format. This choice overrides the way that the cursor is defined. The concept of a binary cursor as such is thus obsolete when using extended query protocol — any cursor can be treated as either text or binary.

Unless WITH HOLD is specified, the cursor created by this command can only be used within the current transaction. Thus, **DECLARE** without WITH HOLD is useless outside a transaction block: the cursor would survive only to the completion of the statement. Therefore PostgreSQL reports an error if such a command is used outside a transaction block. Use **BEGIN** and **COMMIT** (or **ROLLBACK**) to define a transaction block.

If WITH HOLD is specified and the transaction that created the cursor successfully commits, the cursor can continue to be accessed by subsequent transactions in the same session. (But if the creating transaction is aborted, the cursor is removed.) A cursor created with WITH HOLD is closed when an explicit **CLOSE** command is issued on it, or the session ends. In the current implementation, the rows represented by a held cursor are copied into a temporary file or memory area so that they remain available for subsequent transactions.

WITH HOLD may not be specified when the query includes FOR UPDATE or FOR SHARE.

The SCROLL option should be specified when defining a cursor that will be used to fetch backwards. This is required by the SQL standard. However, for compatibility with earlier versions, PostgreSQL will allow backward fetches without SCROLL, if the cursor's query plan is simple enough that no extra overhead is needed to support it. However, application developers are advised not to rely on using backward fetches from a cursor that has not been created with SCROLL. If NO SCROLL is specified, then backward fetches are disallowed in any case.

Backward fetches are also disallowed when the query includes FOR UPDATE or FOR SHARE; therefore SCROLL may not be specified in this case.

Caution

Scrollable cursors may give unexpected results if they invoke any volatile functions (see Section 36.7). When a previously fetched row is re−fetched, the functions might be re−executed, perhaps leading to results different from the first time. It's best to specify NO SCROLL for a query involving volatile functions. If that is not practical, one workaround is to declare the cursor SCROLL WITH HOLD and commit the transaction before reading any rows from it. This will force the entire output of the cursor to be materialized in temporary storage, so that volatile functions are executed exactly once for each row.

If the cursor's query includes FOR UPDATE or FOR SHARE, then returned rows are locked at the time they are first fetched, in the same way as for a regular **SELECT** command with these options. In addition, the returned rows will be the most up−to−date versions.

Caution

It is generally recommended to use FOR UPDATE if the cursor is intended to be used with **UPDATE ... WHERE CURRENT OF** or **DELETE ... WHERE CURRENT OF**. Using FOR UPDATE prevents other sessions from changing the rows between the time they are fetched and the time they are updated. Without FOR UPDATE, a subsequent WHERE CURRENT OF command will have no effect if the row was changed since the cursor was created.

Another reason to use FOR UPDATE is that without it, a subsequent WHERE CURRENT OF might fail if the cursor query does not meet the SQL standard's rules for being “simply updatable” (in particular, the cursor must reference just one table and not use grouping or ORDER BY). Cursors that are not simply updatable might work, or might not, depending on plan choice details; so in the worst case, an application might work in testing and then fail in production. If FOR UPDATE is specified,

the cursor is guaranteed to be updatable.

The main reason not to use FOR UPDATE with WHERE CURRENT OF is if you need the cursor to be scrollable, or to be isolated from concurrent updates (that is, continue to show the old data). If this is a requirement, pay close heed to the caveats shown above.

The SQL standard only makes provisions for cursors in embedded SQL. The PostgreSQL server does not implement an **OPEN** statement for cursors; a cursor is considered to be open when it is declared. However, ECPG, the embedded SQL preprocessor for PostgreSQL, supports the standard SQL cursor conventions, including those involving **DECLARE** and **OPEN** statements.

The server data structure underlying an open cursor is called a portal. Portal names are exposed in the client protocol: a client can fetch rows directly from an open portal, if it knows the portal name. When creating a cursor with **DECLARE**, the portal name is the same as the cursor name.

You can see all available cursors by querying the pg_cursors system view.

EXAMPLES

To declare a cursor:

```
DECLARE liahona CURSOR FOR SELECT * FROM films;
```

See **FETCH**(7) for more examples of cursor usage.

COMPATIBILITY

The SQL standard allows cursors only in embedded SQL and in modules. PostgreSQL permits cursors to be used interactively.

According to the SQL standard, changes made to insensitive cursors by UPDATE ... WHERE CURRENT OF and DELETE ... WHERE CURRENT OF statements are visible in that same cursor. PostgreSQL treats these statements like all other data changing statements in that they are not visible in insensitive cursors.

Binary cursors are a PostgreSQL extension.

SEE ALSO

CLOSE(7), **FETCH**(7), **MOVE**(7)