R E P O R T

**제목 : 19. 커서(CURSOR)**

**과 목 명 : 데이터베이스의이론과실제**

 **학 과 : 전산정보학과**

 **학 번 : 200270225**

 **이 름 : 조철범**

 **제 출 일 : 2002년 12월 13일**

 **담당교수 : 여정모 교수님**

**커서 (CURSOR)**

관계형 데이터베이스에서의 데이터에 대한 작업은 열(Row) 집합을 대상으로 한다. 이러한 열의 집합은 SELECT문과 WHERE 조건에 의해 사용자의 요구 조건에 알맞은 데이터 집합을 만들게 되는데 이를 보통 결과 집합(Result Set)이라고 부른다. 이러한 결과 집합의 크기가 작을 경우에는 이 데이터를 사용하는 클라이언트 응용 프로그램이나 온라인 터미널 프로그램이 결과 데이터 집합 모두를 불러와서 사용해도 무리가 없다. 그러나 데이터가 클 경우 사용자는 이러한 데이터를 작은 블록 또는 몇 개의 열로 나누어서 처리하고 싶은 생각을 할 것이다. 커서는 이러한 점에서 출발하였고, 커서는 질의 후의 결과 집합에 대해서 동작을 하게 된다. 커서를 보다 쉽게 이해하려면 워드프로세서에서의 커서라고 생각해도 무방할 것이다. 커서는 결과 데이터 사이에서 위치 정보를 유지하는 기준점과도 같은 것이고, 워드프로세서 상에서의 현재 페이지는 SELECT 질의한 후의 결과 집합을 의미한다.

커서는 다음과 같이 결과 처리를 확장합니다.

●결과 집합의 특정 행에 위치를 지정할 수 있습니다.

●결과 집합의 현재 위치에서 한 행 또는 행 블록을 검색합니다.

●결과 집합의 현재 위치에서 행에 대한 데이터 수정을 지원합니다.

●다른 사용자가 결과 집합에 나타난 데이터베이스 데이터를 변경한 내용을 여러 가지 수준으로 볼 수 있습니다.

●결과 집합의 데이터에 액세스하는 스크립트, 저장 프로시저 및 트리거에 Transact-SQL 문을 제공합니다.

**커서 요청**

Microsoft® SQL Server™ 2000은 두 가지 커서 요청 방법을 지원합니다.

●Transact-SQL

Transact-SQL 언어는 SQL-92 커서 구문 다음에 모델링된 커서를 사용하기 위한 구문을 지원합니다.

●데이터베이스 응용 프로그래밍 인터페이스(API) 커서 함수

SQL Server는 이러한 데이터베이스 API의 커서 기능을 지원합니다.

●ADO(Microsoft ActiveX® Data Object)

●OLE DB

●ODBC(개방형 데이터베이스 연결)

●DB-Library

응용 프로그램에서는 이러한 두 가지 커서 요청 방법을 혼합하여 사용할 수 없습니다. API를 사용하여 커서 동작을 지정한 응용 프로그램은 Transact-SQL DECLARE CURSOR 문을 실행하여 Transact-SQL 커서도 함께 실행할 수 없습니다. 모든 API 커서 특성을 기본값으로 설정한 경우에만 DECLARE CURSOR를 실행해야 합니다.

Transact-SQL과 API 커서 모두 요청하지 않은 경우에는 기본적으로 SQL Server가 [기본 결과 집합](#sql:default_result_set)이라고 하는 전체 결과 집합을 응용 프로그램으로 반환합니다.

**커서 프로세스**

Transact-SQL 커서와 API 커서는 구문이 다르지만 모든 SQL Server 서버에 대해 다음과 같은 일반 프로세스를 사용합니다.

1.커서와 Transact-SQL 문의 결과 집합을 연관시키고 커서 행의 업데이트 가능 여부 등 커서 특징을 정의합니다.

2.Transact-SQL 문을 실행하여 커서를 채웁니다.

3.원하는 커서의 행을 검색합니다. 커서에서 한 행이나 한 행 블록을 검색하는 작업을 반입이라고 합니다. 일련의 반입 작업을 수행하여 행을 앞쪽 또는 뒤쪽 방향으로 검색하는 것을 스크롤 이라고 합니다.

4.또는 커서의 현재 위치에서 행에 대해 수정 작업(업데이트 또는 삭제)을 수행합니다.

5.커서를 닫습니다.

**커서 구현**

Microsoft® SQL Server™ 2000은 커서 구현을 지원합니다.

**Transact-SQL 커서**

: T-SQL 커서는 저장 프로시저 내에서 컬럼 단위의 데이터 프로세싱이 필요한 경우에 사용되는 커서다. T-SQL 커서는 ANSI SQL-92가 제안하는 표준 커서문을 모두 지원하기 때문에 ANSI SQL-92 커서 라고도 했다. ANSI SQL-92 커서는 스크롤 커서를 지원하지만 ANSI 커서와 T-SQL커서의 차이점은 ANSI 커서는 읽기 전용의 스크롤 커서이고 T-SQL 커서는 읽기,쓰기가 가능한 스크롤 커서이다. T-SQL 커서는 DECLARE, OPEN, FETCH등의 커서문을 사용하고 이 문장들은 ANSI 커서와 호환된다.

**응용 프로그래밍 인터페이스(API) 서버 커서**

: 마이크로소프트의 개방형 데이터베이스 접속 미들웨어인 ODBC와 OLE DB, DB-Library는 클라이언트와 서버의 데이터 베이스 환경에서 강력하고 최적화된 커서 기능을 제공한다. ODBC는 서버와 클라이언트 사이의 중간에서 데이터베이스 서버의 커서를 사용할 수 있게 하는 각종 ODBC API를 지원하고, ODBC를 사용하는 클라이언트는 ODBC API를 기반으로 ODBC Direct라는 ODBC API의 상위 수준의 접속 모듈을 통해 RDO,DAO 등의 데이커베이스 접속 인터페이스를 제공하게 되며, 이들은 모두 커서를 사용할 수 있다.(OLE DB도 그렇다) 예를 들면 API 커서는 T-SQL커서가 커서의 선언을 위해 FETCH등의 명령을 사용하는 대신 SQLExtendedFetch나 SQLSetPos 등의 함수를 사용하며 T-SQL에는 없는 최적화를 위한 많은 함수들을 제공한다.

**클라이언트 커서**

SQL Server ODBC 드라이버, DB-Library DLL 및 ADO API를 구현하는 DLL에 의해 내부적으로 구현됩니다. 클라이언트 커서는 모든 결과 집합 행을 클라이언트에 캐시하여 구현합니다. 클라이언트 응용 프로그램이 API 커서 함수를 호출할 때마다 SQL Server ODBC 드라이버, DB-Library DLL 또는 ADO DLL은 클라이언트에 캐시된 결과 집합 행에 대해 커서 작업을 수행합니다.

Transact-SQL 커서 및 API 서버 커서는 서버에서 구현되므로 통칭적으로 서버 커서라고 합니다.

여러 유형의 커서를 혼합하여 사용하지 마십시오. 응용 프로그램에서 DECLARE CURSOR 및 OPEN 문을 실행하는 경우에는 먼저 API 커서 특성을 기본값으로 설정합니다. API 커서 특성을 기본값이 아닌 다른 값으로 설정한 다음 DECLARE CURSOR 및 OPEN 문을 실행하면 SQL Server에게 API 커서를 Transact-SQL 커서로 매핑하라고 요청하는 의미가 됩니다. 예를 들어, 키 집합 커서를 결과 집합에 대해 매핑 요청을 하는 ODBC 특성을 설정한 다음 명령문 핸들을 사용하여 INSENSITIVE 커서를 요청하는 DECLARE CURSOR 및 OPEN을 실행하지 마십시오.

서버 커서의 단점은 현재 모든 Transact-SQL 문을 지원하지 않는다는 점입니다. 서버 커서는 여러 결과 집합을 생성하는 Transact-SQL 문을 지원하지 않으므로 응용 프로그램이 둘 이상의 SELECT 문을 포함하는 일괄 처리 또는 저장 프로시저를 실행할 때는 사용할 수 없습니다. 서버 커서는 COMPUTE, COMPUTE BY, FOR BROWSE, INTO 키워드를 포함하는 SQL 문도 지원하지 않습니다.

**서버 커서 및 클라이언트 커서**

클라이언트 커서 대신 서버 커서를 사용하면 다음과 같은 이점이 있습니다.

●성능

커서의 데이터 일부에 액세스할 경우(예: 브라우징 응용 프로그램)에는 반입된 데이터만 네트워크를 통해 보내므로 서버 커서를 사용할 때 성능이 최적화됩니다. 클라이언트 커서는 전체 결과 집합을 클라이언트에 캐시합니다.

●기타 커서 유형

SQL Server ODBC 드라이버가 클라이언트 커서만 사용할 경우에는 전진 전용 및 정적 커서만 지원할 수 있습니다. API 서버 커서를 사용하면 드라이버가 키 집합 커서 및 동적 커서만 지원할 수 있습니다. SQL Server는 또한 서버 커서를 통해서만 커서 동시성 특성의 전체 범위를 지원합니다. 클라이언트 커서는 지원하는 기능으로만 제한됩니다.

●더 자세한 위치 지정 업데이트

서버 커서는 ODBC **SQLSetPos** 함수 또는 WHERE CURRENT OF 절이 있는 UPDATE 및 DELETE 문 등 위치가 지정된 작업을 직접 지원합니다. 그러나 클라이언트 커서는 Transact-SQL 검색된 UPDATE 문을 생성하여 위치 지정된 커서 업데이트를 시뮬레이션합니다. 그 결과 UPDATE 문의 WHERE 절 조건에 맞는 행이 둘 이상 있는 경우 불필요한 업데이트가 수행됩니다.

●메모리 사용

서버 커서를 사용하면 많은 데이터를 캐시하거나 커서 위치에 대한 정보를 보관하는 작업이 서버에서 이루어지므로 클라이언트는 이러한 작업을 할 필요가 없습니다.

●다중 활성 문

서버 커서를 사용하면 커서 작업 사이에 처리 중인 결과가 연결에 남아 있지 않습니다. 따라서 여러 커서 기반의 명령문을 동시에 활성화할 수 있습니다.

정적 커서나 무반응 커서를 제외한 모든 서버 커서의 작업은 원본 테이블의 스키마에 종속됩니다. 커서를 선언한 다음 이러한 테이블에 대해 스키마를 변경하면 해당 커서에 대한 다음 작업 시 오류가 발생합니다.

**커서 지정**

OLE DB, ODBC 및 ADO 사양에서 커서는 Transact-SQL 문에 의해 반환된 모든 결과 집합에 대해 암시적으로 열립니다. Transact-SQL 문을 실행하기 전에 특징이나 속성을 설정하여 커서 특성을 정의합니다. 그런 다음 API 함수를 호출하여 한 행이나 일괄 행을 한 번에 반입할 수 있습니다. API 커서 특성 또는 속성의 기본 설정은 SQL Server 기본 결과 집합과 동일한 특징을 갖습니다. SQL Server ODBC 드라이버 및 Microsoft OLE DB Provider for SQL Server는 모두 커서 특성 또는 속성이 기본값으로 설정될 때 기본 결과 집합을 구현합니다. 이 경우에는 커서가 사용되지 않습니다. 드라이버와 공급자는 커서 특성이나 속성이 기본값에서 변경될 경우에만 커서를 구현합니다.

OLE DB, ODBC 및 ADO는 이 결과 집합 및 커서 조합에 대해 다른 용어를 사용합니다.

●OLE DB에서는 결과 집합과 관련 커서 동작의 조합을 행 집합이라고 합니다.

●ODBC는 각 결과 집합에 자동으로 매핑되는 커서가 포함되어 있다고 간주하므로 결과 집합과 커서라는 용어를 별 구분 없이 사용합니다. ODBC의 행 집합은 반입 작업에서 반환된 행 수를 특별히 일컫는 용어입니다.

●OLE DB에서의 행 집합을 ADO에서는 레코드 집합이라고 합니다.

DB-Library API는 두 가지 결과 집합 처리 방법을 지원합니다. DB-Library 핵심 함수는 Transact-SQL 문 결과 처리를 기본 결과 집합으로만 지원합니다. 핵심 함수를 사용하면 결과 집합의 행을 전진 전용 방향으로만 한 번에 검색할 수 있으며 결과 집합을 통한 업데이트는 지원하지 않습니다. DB-Library 응용 프로그램에서 커서 처리를 사용하려면 핵심 함수 대신 DB-Library 커서 라이브러리의 특수 함수를 사용해야 합니다.

Microsoft OLE DB Provider for SQL Server, SQL Server ODBC 드라이버 및 DB-Library의 기본값은 API 서버 커서를 사용하여 요청한 커서 함수를 구현하는 것입니다. ODBC 클라이언트 응용 프로그램은 ODBC 커서 라이브러리를 로드하여 서버 커서 대신 클라이언트 커서를 사용할 수 있습니다. DB-Library 응용 프로그램은 DBCLIENTCURSOR 옵션을 설정하여 클라이언트 커서를 사용할 수 있습니다. OLE DB에는 클라이언트 커서의 고유한 구현이 없지만 ADO API에는 있습니다.

다음 그림은 현재 API 커서 설정 및 실행 중인 Transact-SQL 문을 기준으로 구현된 커서 또는 결과 집합 유형을 요약한 것입니다. "커서 유형을 결합하지 마십시오"라는 셀은 Transact-SQL 서버 및 API 커서가 동시에 요청 중이라는 것을 의미합니다. 이 경우 정의되지 않은 동작이 일어납니다



**Transact-SQL 커서**

Transact-SQL 커서는 주로 다른 Transact-SQL 문에서 사용할 수 있는 결과 집합 내용을 만드는 Transact-SQL 스크립트, 트리거, 저장 프로시저에서 사용됩니다.

저장 프로시저나 트리거에서는 일반적으로 다음과 같은 경우에 Transact-SQL 커서를 사용합니다.

1.Transact-SQL 변수를 선언하여 커서에서 반환한 데이터를 포함합니다. 각 결과 집합 열에 대해 하나의 변수를 선언합니다. 열의 데이터 형식에서 암시적으로 변환할 수 있는 데이터 형식과 열에 의해 반환된 값을 모두 포함할 수 있을 정도의 크기로 변수를 선언합니다.

2.DECLARE CURSOR 문을 사용하여 Transact-SQL 커서와 SELECT 문을 연결합니다. 또한 DECLARE CURSOR 문은 커서 이름과 커서가 읽기 전용인지 전진 전용인지의 여부 등 커서 특징도 정의합니다.

3.OPEN 문을 사용하여 SELECT 문을 실행하고 커서를 채웁니다.

4.FETCH INTO 문을 사용하여 각 행을 반입하고 각 열의 데이터를 지정한 변수로 옮깁니다. 그런 다음 다른 Transact-SQL 문이 이 변수를 참조하여 반입된 데이터 값에 액세스할 수 있습니다. Transact-SQL 커서는 행 블록 반입을 지원하지 않습니다.

**※ @@FETCH\_STATUS**

현재 연결로 열린 모든 커서에 대해 실행된 마지막 FETCH 문의 상태를 반환합니다.

|  |  |
| --- | --- |
| **반환 값** | **설명** |
| 0 | FETCH 문은 성공적이었습니다. |
| -1 | FETCH 문은 실패했거나 행이 결과 집합의 범위를 벗어났습니다. |
| -2 | 반입된 행이 없습니다. |

[구문] @@FETCH\_STATUS [반환 형식] **integer**

**비고**

@@FETCH\_STATUS는 연결의 모든 커서에 전역으로 사용되므로 @@FETCH\_STATUS를 신중히 사용하십시오. FETCH 문이 실행된 후 @@FETCH\_STATUS의 테스트는 다른 FETCH 문이 다른 커서에 대해 실행되기 전에 이루어져야 합니다. @@FETCH\_STATUS의 값은 연결에서 반입이 이루어지기 전에 정의되지 않습니다.

예를 들어, 사용자는 커서 하나에서 FETCH 문을 실행한 다음 다른 커서에서 결과를 열고 처리하는 저장 프로시저를 호출합니다. 컨트롤이 호출된 저장 프로시저에서 반환되면 @@FETCH\_STATUS는 저장 프로시저에서 실행된 마지막 FETCH를 반영하고 저장 프로시저가 호출되기 전에 실행된 FETCH 문은 반영하지 않습니다.

다음은 @@FETCH\_STATUS를 사용하여 WHILE 루프에 있는 커서 작업을 제어하는 예제입니다.

DECLARE Employee\_Cursor CURSOR FOR

SELECT LastName, FirstName FROM Northwind.dbo.Employees

OPEN Employee\_Cursor

FETCH NEXT FROM Employee\_Cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

 FETCH NEXT FROM Employee\_Cursor

END

CLOSE Employee\_Cursor

DEALLOCATE Employee\_Cursor

5.커서 작업이 끝나면 CLOSE 문을 사용합니다. 커서를 닫으면 커서의 결과 집합 및 현재 행에 대한 잠금 등의 리소스가 해제되지만 OPEN 문을 다시 실행할 경우 커서 구조를 여전히 처리에 사용할 수 있습니다. 커서가 계속 표시되므로 이 경우에는 커서 이름을 다시 사용할 수 없습니다. DEALLOCATE 문은 커서 이름을 포함하여 커서에 할당된 모드 리소스를 완전히 해제합니다. 커서 할당을 취소한 후에는 DECLARE 문을 실행하여 커서를 다시 작성해야 합니다.

## DECLARE CURSOR

스크롤 동작, 커서가 작동하는 결과 집합을 구축하는 데 사용되는 쿼리 등 Transact-SQL 서버 커서의 속성을 정의합니다. DECLARE CURSOR는 SQL-92 표준 구문과 Transact-SQL 확장 구문을 모두 허용합니다.

**[SQL-92 구문]**

DECLARE *cursor\_name* [ INSENSITIVE ] [ SCROLL ] CURSOR
FOR *select\_statement*
[ FOR { READ ONLY | UPDATE [ OF *column\_name* [ **,**...*n* ] ] } ]

**[Transact-SQL 확장 구문]**

DECLARE *cursor\_name* CURSOR
[ LOCAL | GLOBAL ]
[ FORWARD\_ONLY | SCROLL ]
[ STATIC | KEYSET | DYNAMIC | FAST\_FORWARD ]
[ READ\_ONLY | SCROLL\_LOCKS | OPTIMISTIC ]
[ TYPE\_WARNING ]
FOR *select\_statement*
[ FOR UPDATE [ OF *column\_name* [ **,**...*n* ] ] ]

**[SQL-92 인수]**

🞟 cursor\_name

정의된 Transact-SQL 서버 커서의 이름입니다. *cursor\_name*은 식별자에 대한 규칙을 따라야 합니다. 식별자 규칙에 대한 자세한 내용은 식별자 사용을 참조하십시오.

🞟 INSENSITIVE

커서에서 사용할 데이터를 임시로 복사해 주는 커서를 정의합니다. 커서에 대한 모든 요청은 **tempdb**의 임시 테이블에서 응답하므로 기본 테이블에 대한 수정 사항은 해당 커서에 대한 반입으로 반환된 데이터에는 반영되지 않고 해당 커서는 수정할 수 없습니다. SQL-92 구문을 사용할 때 INSENSITIVE를 생략하면 원본으로 사용하는 테이블에서 커밋된 삭제 및 업데이트 작업이 향후 반입된 데이터에 반영됩니다.

🞟 SCROLL

모든 반입 옵션(FIRST, LAST, PRIOR, NEXT, RELATIVE, ABSOLUTE)을 사용할 수 있도록 지정합니다. SQL-92 DECLARE CURSOR에서 SCROLL을 지정하지 않으면 NEXT 반입 옵션만 지원됩니다. FAST\_FORWARD가 지정된 경우에는 SCROLL을 지정할 수 없습니다.

🞟 select\_statement

커서의 결과 집합을 정의하는 표준 SELECT 문입니다. 커서 선언의 *select\_statement*에서 COMPUTE, COMPUTE BY, FOR BROWSE, INTO 키워드는 사용할 수 없습니다.

Microsoft® SQL Server™는 *select\_statement*의 절이 요청한 커서 형식의 기능과 충돌할 경우 커서를 다른 형식으로 암시적으로 변환합니다. 자세한 내용은 암시적 커서 변환을 참조하십시오.

🞟 READ ONLY

이 커서를 통한 업데이트를 방지합니다. UPDATE 문이나 DELETE 문의 WHERE CURRENT OF 절에서는 이 커서를 참조할 수 없습니다. 이 옵션에서는 업데이트할 커서의 기본 기능을 무시합니다.

UPDATE [OF *column\_name* [**,**...*n*]]

커서 내에서 업데이트할 수 있는 열을 정의합니다. OF *column*\_*name* [**,**...*n*]을 지정하면 표시된 열만 수정할 수 있으나 열 목록 없이 UPDATE를 지정하면 모든 열을 업데이트할 수 있습니다.

**[Transact-SQL 확장 인수]**

🞟 *cursor\_name*

정의된 Transact-SQL 서버 커서의 이름입니다. *cursor\_name*은 식별자에 대한 규칙을 따라야 합니다. 식별자 규칙에 관한 자세한 내용은 식별자 사용을 참조하십시오.

🞟 LOCAL

커서가 만들어진 일괄 처리, 저장 프로시저, 트리거에서 커서 범위를 지역 커서로 지정합니다. 커서 이름은 지정된 범위에서만 유효합니다. 일괄 처리, 저장 프로시저, 트리거의 지역 커서 변수, 또는 저장 프로시저 OUTPUT 매개 변수에서 커서를 참조할 수 있습니다. OUTPUT 매개 변수는 지역 커서를 호출한 일괄 처리, 저장 프로시저, 트리거로 다시 전달하는 데 사용되며 저장 프로시저가 종료된 후 커서 변수에 매개 변수를 할당하여 커서를 참조할 수 있습니다. 커서가 OUTPUT 매개 변수에 다시 전달되지 않을 경우 일괄 처리나 저장 프로시저, 트리거가 종료되면 커서가 암시적으로 할당 해제됩니다. OUTPUT 매개 변수에서 커서가 다시 전달되면, 커서를 참조하는 마지막 변수의 할당이 취소되거나 공간이 부족할 때 커서의 할당이 취소됩니다.

🞟 GLOBAL

커서 범위를 연결의 전역 커서로 지정합니다. 연결되어 실행하는 모든 저장 프로시저 또는 일괄 처리에서 커서 이름을 참조할 수 있습니다. 커서는 연결 해제 시에만 암시적으로 할당이 해제됩니다.

**참고**   GLOBAL이나 LOCAL을 모두 지정하지 않으면 기본적으로 **default to local cursor** 데이터베이스 옵션의 설정에 따라 결정됩니다. SQL Server 7.0에서 이 옵션은 모든 커서가 전역 커서인 이전 버전의 SQL Server와 일치하도록 기본적으로 FALSE가 됩니다. SQL Server의 이후 버전에서는 이 옵션의 기본값이 변경됩니다. 자세한 내용을 보려면 데이터베이스 옵션 설정을 참고하십시오.

🞟 FORWARD\_ONLY

첫 번째 행에서 마지막 행 방향으로만 커서를 스크롤할 수 있도록 지정합니다. 유일하게 지원되는 반입 옵션은 FETCH NEXT입니다. STATIC, KEYSET, DYNAMIC 키워드를 사용하지 않고 FORWARD\_ONLY를 지정하면 DYNAMIC 커서로 작동됩니다. STATIC, KEYSET, DYNAMIC 키워드가 지정된 경우를 제외하면 FORWARD\_ONLY나 SCROLL을 모두 지정하지 않을 경우 FORWARD\_ONLY가 기본값이 됩니다. STATIC, KEYSET, DYNAMIC 커서는 기본적으로 SCROLL이 됩니다. ODBC, ADO 등의 데이터베이스 API와 달리 STATIC, KEYSET, DYNAMIC Transact-SQL 커서에서는 FORWARD\_ONLY가 지원됩니다. FAST\_FORWARD와 FORWARD\_ONLY는 서로 배타적입니다. 하나를 지정하면 다른 하나는 지정할 수 없습니다.

🞟 STATIC

커서에서 사용할 데이터를 임시로 복사해 주는 커서를 정의합니다. 커서에 대한 모든 요청은 **tempdb**의 임시 테이블에서 응답하므로 기본 테이블에 대한 수정 사항은 해당 커서에 대한 반입으로 반환된 데이터에는 반영되지 않고 해당 커서는 수정할 수 없습니다.

🞟 KEYSET

커서를 열 때 커서에 포함된 행의 구성원과 순서가 고정되도록 지정합니다. 행을 고유하게 식별하는 **keyset**라는 키 집합이 **tempdb**의 테이블에 구축됩니다. 커서 소유자나 다른 사용자가 기본 테이블에서 키가 아닌 값을 변경하면 커서 소유자가 커서를 스크롤할 때 변경 사항이 표시됩니다. 그러나 다른 사용자가 삽입한 데이터는 볼 수 없습니다(Transact-SQL 서버 커서를 통해 삽입할 수 없음). 삭제된 행을 반입하려고 하면 @@FETCH\_STATUS가 -2로 반환됩니다. 커서 외부에서 키 값을 업데이트하는 것은 이전 행을 삭제하고 새 행을 삽입하는 것과 같습니다. 새 값을 가진 행을 볼 수 없으므로 이전 값을 가진 행을 반입하려고 하면 @@FETCH\_STATUS가 -2로 반환됩니다. WHERE CURRENT OF 절을 지정하여 커서를 통해 업데이트를 수행한 경우에는 새 값을 볼 수 있습니다.

DYNAMIC

커서를 스크롤할 때 결과 집합에 행에 대한 모든 데이터 변경 사항이 반영되는 커서를 정의합니다. 따라서 반입할 때마다 행의 데이터 값과 순서, 구성원이 변경될 수 있습니다. 동적 커서에서는 ABSOLUTE 반입 옵션이 지원되지 않습니다.

🞟 FAST\_FORWARD

성능 최적화가 설정된 FORWARD\_ONLY, READ\_ONLY 커서를 지정합니다. SCROLL 또는 FOR\_UPDATE가 지정된 경우에는 FAST\_FORWARD를 지정할 수 없습니다. FAST\_FORWARD와 FORWARD\_ONLY는 서로 배타적입니다. 하나를 지정하면 다른 하나는 지정할 수 없습니다.

🞟 READ\_ONLY

이 커서를 통한 업데이트를 방지합니다. UPDATE 문이나 DELETE 문의 WHERE CURRENT OF 절에서는 이 커서를 참조할 수 없습니다. 이 옵션에서는 업데이트할 커서의 기본 기능을 무시합니다.

🞟 SCROLL\_LOCKS

커서를 통해 수행되는 위치 지정 업데이트나 삭제가 성공하도록 지정합니다. Microsoft® SQL Server™는 커서로 행을 읽어들일 때 나중에 수정할 수 있도록 행을 잠급니다. FAST\_FORWARD가 지정된 경우에는 SCROLL\_LOCKS를 지정할 수 없습니다.

🞟 OPTIMISTIC

커서로 행을 읽어들인 후 해당 행이 업데이트 된 경우 커서를 통해 수행되는 위치 지정 업데이트나 삭제가 성공하지 못하도록 지정합니다. 이 경우 SQL Server는 커서로 행을 읽어들일 때 잠그지 않습니다. 대신 **timestamp** 열 값을 비교하거나 테이블에 **timestamp** 열이 없을 경우 검사값을 비교하여 커서로 행을 읽어들인 후 해당 행이 수정되었는지를 확인합니다. 행이 수정된 경우 시도한 위치 지정 업데이트나 삭제가 실패합니다. FAST\_FORWARD가 지정된 경우에는 OPTIMISTIC을 지정할 수 없습니다.

🞟 TYPE\_WARNING

요청한 커서 형식이 다른 형식으로 암시적으로 변환된 경우 클라이언트에게 경고 메시지를 보냅니다.

🞟 *select\_statement*

커서의 결과 집합을 정의하는 표준 SELECT 문입니다. 커서 선언의 *select\_statement*에서 COMPUTE, COMPUTE BY, FOR BROWSE, INTO 키워드는 사용할 수 없습니다.

*select\_statement*의 절이 요청한 커서 형식의 기능과 충돌할 경우 SQL Server는 커서를 다른 형식으로 암시적으로 변환합니다. 자세한 내용은 암시적 커서 변환을 참조하십시오.

🞟 UPDATE [OF *column\_name* [**,**...*n*]]

커서 내에서 업데이트할 수 있는 열을 정의합니다. OF *column*\_*name* [**,**...*n*]을 지정하면 표시된 열만 수정할 수 있습니다. READ\_ONLY 병행성 옵션이 지정된 경우를 제외하면 열 목록 없이 UPDATE를 지정할 경우 모든 열을 업데이트할 수 있습니다.

**비고**

DECLARE CURSOR는 스크롤 동작, 커서가 작동하는 결과 집합을 구축하는 데 사용되는 쿼리 등 Transact-SQL 서버 커서의 속성을 정의합니다. OPEN 문은 결과 집합을 채우고 FETCH는 결과 집합에서 행을 반환합니다. CLOSE 문은 커서와 연결된 현재 결과 집합을 해제하며 DEALLOCATE 문은 커서에서 사용된 자원을 해제합니다.

DECLARE CURSOR 문의 첫 번째 형식은 커서 동작을 선언하기 위해 SQL-92 구문을 사용합니다. DECLARE CURSOR 문의 두 번째 형식은 ODBC, ADO, DB-Library의 데이터베이스 API 커서 함수에서 사용된 것과 동일한 커서 형식을 사용하여 커서를 정의할 수 있는 Transact-SQL 확장을 사용합니다.

위의 두 가지 형식을 함께 사용할 수는 없습니다. CURSOR 키워드 앞에 SCROLL 또는 INSENSITIVE 키워드를 지정하면 CURSOR와 FOR *select\_statement* 키워드 사이에 어떤 키워드도 사용할 수 없습니다. CURSOR와 FOR *select\_statement* 사이에 키워드를 지정하면 CURSOR 키워드 앞에 SCROLL 또는 INSENSITIVE를 지정할 수 없습니다.

Transact-SQL 구문을 사용하는 DECLARE CURSOR에서 READ\_ONLY, OPTIMISTIC, SCROLL\_LOCKS를 지정하지 않을 경우 기본값은 다음과 같습니다.

●권한 부족, 업데이트를 지원하지 않는 원격 테이블 액세스 등의 이유로 SELECT 문이 업데이트를 지원하지 않을 경우 커서는 READ\_ONLY가 됩니다.

●STATIC 및 FAST\_FORWARD 커서는 기본적으로 READ\_ONLY가 됩니다.

●DYNAMIC 및 KEYSET 커서는 기본적으로 OPTIMISTIC이 됩니다.

커서 이름은 다른 Transact-SQL 문에서만 참조될 수 있으며 데이터베이스 API 함수에서는 참조될 수 없습니다. 예를 들어, 커서를 선언한 후 OLE DB, ODBC, ADO, DB-Library 함수나 메서드에서 커서 이름을 참조할 수 없습니다. 커서 행은 API의 반입 함수나 메서드를 사용하여 반입할 수 없고 Transact-SQL FETCH 문에 의해서만 반입할 수 있습니다.

커서를 선언한 후 다음 시스템 저장 프로시저를 사용하여 커서의 특징을 확인할 수 있습니다.

|  |  |
| --- | --- |
| **시스템 저장 프로시저** | **설명** |
| **Sp\_cursor\_list** | 현재 연결에서 볼 수 있는 커서 목록과 그 속성을 반환합니다. |
| **Sp\_describe\_cursor** | 전진 전용 커서, 스크롤 커서 등의 커서 속성을 설명합니다. |
| **Sp\_describe\_cursor\_columns** | 커서 결과 집합에서 열의 속성을 설명합니다. |
| **Sp\_describe\_cursor\_tables** | 커서에 의해 액세스되는 기본 테이블을 설명합니다. |

변수는 커서를 선언하는 *select\_statement*의 일부로 사용될 수 있습니다. 그러나 커서가 선언된 후에 이러한 변수를 수정한 내용은 커서 작업에 영향을 주지 않습니다.

**사용 권한**

DECLARE CURSOR 권한은 커서에 사용된 뷰, 테이블, 열에 SELECT 권한이 있는 모든 사용자에게 기본적으로 부여됩니다.

예제

**A. 단순 커서 및 구문 사용**

다음 커서를 열 때 생성된 결과 집합에는 **pubs** 데이터베이스의 **authors** 테이블에 있는 모든 행과 모든 열이 포함됩니다. 이 커서는 업데이트가 가능하며 이 커서에 대해 수행한 반입에는 모든 업데이트와 삭제 작업이 반영됩니다. SCROLL 옵션을 지정하지 않았으므로 FETCH NEXT 반입 옵션만 사용할 수 있습니다.

DECLARE authors\_cursor CURSOR

 FOR SELECT \* FROM authors

OPEN authors\_cursor

FETCH NEXT FROM authors\_cursor

위와 같이 DECLARE문으로 커서를 선언하고, OPEN문으로 SELECT 로 질의한 결과 집합의 처음 위치로 커서를 가져다 놓고 FETCH NEXT FORM 문으로 커서를 이동시킨다. 다음의 FETCH NEXT문을 계속 사용하게 되면, 커서는 다음 열로 이동하게 되고, 한 개의 열씩 표시하게 된다.

FETCH NEXT FROM authors\_cursor -- 두번째 열

FETCH NEXT FROM authors\_cursor -- 세번째 열

FETCH NEXT FROM authors\_cursor -- 네번째 열

………..

선언된 커서의 사용이 끝나면 커서를 닫고 할당된 시스템 자원을 해제한다.

CLOSE authors\_cursor

DEALLOCATE authors\_cursor

## OPEN

DECLARE CURSOR 또는 SET *cursor\_variable* 문에서 지정된 Transact-SQL 문을 실행하여 Transact-SQL 서버 커서를 열고 커서를 채웁니다.

**[구문]**

OPEN { { [ GLOBAL ] *cursor\_name* } | *cursor\_variable\_name* }

**[인수]**

🞟GLOBAL

*cursor\_name*이 전역 커서를 가리키도록 지정합니다.

🞟 *cursor\_name*

선언된 커서의 이름입니다. 동일한 *cursor\_name*을 가진 전역 커서와 로컬 커서가 모두 존재할 경우 GLOBAL이 지정되어 있으면 *cursor\_name*이 전역 커서를 가리키고 그렇지 않으면 로컬 커서를 가리킵니다.

🞟 *cursor\_variable\_name*

커서를 참조하는 커서 변수의 이름입니다.

**비고**

INSENSITIVE 또는 STATIC 옵션을 사용하여 커서를 선언한 경우, OPEN은 임시 테이블을 만들어 결과 집합을 보관합니다. 결과 집합에 Microsoft® SQL Server™ 테이블의 최대 행 크기를 초과하는 행이 있으면 OPEN이 실패합니다. KEYSET 옵션을 사용하여 커서를 선언하는 경우에는 OPEN이 임시 테이블을 만들어 키 집합을 보관합니다. 임시 테이블은 **tempdb**에 저장됩니다.

커서가 열린 후 @@CURSOR\_ROWS 함수를 사용하여 마지막으로 열린 커서에서 한정하는 행 수를 받습니다. 결과 집합에서 예상되는 행 수에 따라 SQL Server는 별도의 스레드에서 비동기식으로 키 집합 커서를 채우기로 선택할 수 있습니다. 이렇게 하여 키 집합이 완전히 채워지지 않았어도 반입이 즉시 계속되게 할 수 있습니다. 자세한 내용은 비동기 채우기를 참조하십시오.

SQL Server가 키 집합을 비동기식으로 생성하는 임계값을 설정하려면 **cursor threshold** 구성 옵션을 설정하십시오. 자세한 내용은 [sp\_configure](http://isec.pknu.ac.kr/datacenter/old_data/updown5/updown/UPload/ts_sp_ca-cz_0put.htm)를 참조하십시오.

**A.커서를 열고 모든 행을 반입하는 예제**

Northwind 데이터베이스의 Employees 테이블에 커서를 적용하여 커서를 OPEN하는 예이다. 중요한 것은 WHILE문을 사용하여 @@FETCH\_STATUS가 0 이 될 때까지 계속적으로 FETCH를 한다는 점이다.

DECLARE Employee\_Cursor CURSOR FOR

SELECT LastName, FirstName

FROM Northwind.dbo.Employees

WHERE LastName like 'B%'

OPEN Employee\_Cursor

FETCH NEXT FROM Employee\_Cursor

WHILE @@FETCH\_STATUS = 0

BEGIN

 FETCH NEXT FROM Employee\_Cursor

END

CLOSE Employee\_Cursor

DEALLOCATE Employee\_Cursor

## FETCH

Transact-SQL 서버 커서에서 특정 행을 검색합니다.

**[구문]**

FETCH
        [ [ NEXT | PRIOR | FIRST | LAST
                | ABSOLUTE { *n* |**@***nvar* }
                | RELATIVE { *n* |**@***nvar* }
            ]
             FROM
        ]
{ { [ GLOBAL ] *cursor\_name* } | @*cursor\_variable\_name* }
[ INTO**@***variable\_name* [ **,**...*n* ] ]

**[인수]**

🞟NEXT

현재 행 바로 다음의 결과 행을 반환하며 현재 행을 반환되는 행 앞의 행으로 만듭니다. 커서에 대해 FETCH NEXT가 첫 번째 반입인 경우에는 결과 집합의 첫 번째 행을 반환합니다. NEXT는 기본 커서 반입 옵션입니다.

🞟PRIOR

현재 행 바로 앞의 결과 행을 반환하며 현재 행을 반환되는 행 뒤의 행으로 만듭니다. 커서에 대해 FETCH PRIOR가 첫 번째 반입인 경우에는 행이 반환되지 않으며 커서는 첫 번째 행 앞에 위치하게 됩니다.

🞟FIRST

커서의 첫 번째 행을 반환하며 그 행을 현재 행으로 만듭니다.

🞟LAST

커서의 마지막 행을 반환하며 그 행을 현재 행으로 만듭니다.

🞟ABSOLUTE {*n* |**@***nvar*}

*n* 또는 *@nvar*이 양수인 경우에는 커서 맨 앞에서부터 *n* 번째 행에 있는 행을 반환하며 반환되는 행을 새 현재 행으로 만듭니다. *n* 또는 *@nvar*이 음수인 경우에는 커서 맨 뒤에서부터 *n* 번째 행에 있는 행을 반환하며 반환되는 행을 새 현재 행으로 만듭니다. *n* 또는 *@nvar*이 0인 경우에는 행이 반환되지 않습니다. *n*은 반드시 정수 상수여야 하며 *@nvar*은 반드시 **smallint**, **tinyint**, **int** 중 하나여야 합니다.

🞟RELATIVE {*n* |**@***nvar*}

*n* 또는 *@nvar*이 양수인 경우에는 현재 행에서부터 *n* 번째 행에 있는 행을 반환하며 반환되는 행을 새 현재 행으로 만듭니다. *n* 또는 *@nvar*이 음수인 경우에는 현재 행에서부터 앞으로 *n* 번째 행에 있는 행을 반환하며 반환되는 행을 새 현재 행으로 만듭니다. *n* 또는 *@nvar*이 0인 경우에는 현재 행을 반환합니다. 커서에 대해 처음으로 실행된 반입에 음수 또는 0으로 설정된 *n* 또는 *@nvar*를 사용하여 FETCH RELATIVE를 지정한 경우에는 행이 반환되지 않습니다. *n*은 반드시 정수 상수여야 하며 *@nvar*은 **smallint**, **tinyint**, **int** 중 하나여야 합니다.

🞟GLOBAL

*cursor\_name*이 전역 커서를 가리키도록 지정합니다.

🞟*cursor\_name*

반입이 수행되어야 하는 열려진 커서의 이름입니다. 동일한 *cursor\_name*을 가진 전역 커서와 지역 커서가 있을 경우, GLOBAL이 지정되면 *cursor\_name*에 전역 커서를 사용하고 GLOBAL이 지정되지 않으면 지역 커서를 사용합니다.

🞟*@cursor\_variable\_name*

수행할 반입에서 열려진 커서를 참조하는 커서 변수의 이름입니다.

🞟INTO *@variable\_name*[**,***...n*]

데이터를 반입의 열에서 지역 변수로 가져가도록 허용합니다. 목록의 각 변수는 왼쪽에서 오른쪽 순으로 커서 결과 집합의 해당 열과 연관됩니다. 각 변수의 데이터 형식은 반드시 해당 결과 집합 열의 데이터 형식과 일치하거나 암시적 변환이 지원되어야 합니다. 변수의 개수는 커서 선택 목록의 열 수와 일치해야 합니다.

**FETCH 문 사용 시 주의 사항**

🞟 만약 FORWARD\_ONLY나 FAST\_FORWARD 옵션이 설정되어 있다면, FETCH문에 사용할 수 있는 옵션은 오직 NEXT 밖에 없다.(전방 커서이기 때문)

🞟 만약 DYNAMIC, FORWARD\_ONLY, FAST\_FORWARD 옵션이 지정되어 있지 않고 KEYSET, STATIC, SCROLL 옵션 중 하나가 설정되어 있다면, FETCH문에는 사용할 수 있는 모든 옵션의 사용이 가능하다.

🞟 DYNAMIC SCROLL 커서는 ABSOLUTE를 제외한 모든 FETCH 옵션을 사용할 수 있다.

🞟 @@FETCH\_STATUS를 통해 확인할 수 있는 가장 최근의 FETCH 상황은 sp\_describe\_cursor 저장 프로시저의 결과인 fetch\_status 컬럼을 통해 확인할 수도 있다.

**비고**

SQL-92 스타일 DECLARE CURSOR 문에서 SCROLL 옵션을 지정하지 않는 경우, 유일하게 지원되는 FETCH 옵션은 NEXT입니다. SQL-92 스타일 DECLARE CURSOR커서에서 SCROLL을 지정하면 모든 FETCH 옵션이 지원됩니다.

Transact\_SQL DECLARE 커서 확장을 사용하는 경우에는 다음 규칙이 적용됩니다.

🞟FORWARD-ONLY나 FAST\_FORWARD를 지정한 경우에는 유일하게 지원되는 FETCH 옵션이 NEXT입니다.

🞟DYNAMIC, FORWARD\_ONLY 또는 FAST\_FORWARD를 지정하지 않고 KEYSET, STATIC, SCROLL 중 하나를 지정한 경우에는 모든 FETCH 옵션이 지원됩니다.

🞟DYNAMIC SCROLL 커서는 ABSOLUTE를 제외한 모든 FETCH 옵션을 지원합니다.

@@FETCH\_STATUS 함수는 마지막 FETCH 문의 상태를 보고합니다. **sp\_describe\_cursor**에 의해 반환되는 커서의 **fetch\_status** 열에도 동일한 정보가 기록됩니다. 해당 데이터에 대해 어떠한 작업을 수행하려고 시도하기 전에 반드시 이 상태 정보를 사용하여 FETCH 문이 반환하는 데이터의 유효성을 확인해야 합니다. 자세한 내용은 [@@FETCH\_STATUS](http://isec.pknu.ac.kr/datacenter/old_data/updown5/updown/UPload/ts_globals_1c1f.htm)를 참조하십시오.

**사용 권한**

FETCH 사용 권한은 모든 유효한 사용자에게 기본적으로 부여됩니다.

**[예제]**

**A. 단순 커서에서 FETCH 사용**

다음은 성이 B로 시작하는 **authors** 테이블에 있는 행에 대해 단순 커서를 선언하고 FETCH NEXT로 전체 행에 대해 수행하는 예제입니다. 생성된 커서에 대해 FETCH를 실행하여 커서의 위치를 이동한 후 @@FETCH\_STATUS를 통해 커서의 FETCH 상황을 확인한다.

USE pubs

GO

DECLARE authors\_cursor CURSOR FOR

SELECT au\_lname FROM authors

WHERE au\_lname LIKE "B%"

ORDER BY au\_lname

OPEN authors\_cursor

-- Perform the first fetch.

FETCH NEXT FROM authors\_cursor

위의 커서문을 통하여 첫 번째 FETCH가 끝났으면 @@FETCH\_STATUS 의 현재 상태값을 확인하여 0 이 될 때까지 계속적으로 FETCH를 실행한다.

-- Check @@FETCH\_STATUS to see if there are any more rows to fetch.

WHILE @@FETCH\_STATUS = 0

BEGIN

 -- This is executed as long as the previous fetch succeeds.

 FETCH NEXT FROM authors\_cursor

END

CLOSE authors\_cursor

DEALLOCATE authors\_cursor

GO

**B. FETCH를 사용하여 변수에 값 저장**

다음은 지역 변수를 사용하여 커서를 선언하고 FETCH에 적용하는 예이다.

USE pubs

GO

-- Declare the variables to store the values returned by FETCH.

DECLARE @au\_lname varchar(40), @au\_fname varchar(20)

DECLARE authors\_cursor CURSOR FOR

SELECT au\_lname, au\_fname FROM authors

WHERE au\_lname LIKE "B%"

ORDER BY au\_lname, au\_fname

OPEN authors\_cursor

-- Perform the first fetch and store the values in variables.

-- Note: The variables are in the same order as the columns

-- in the SELECT statement.

FETCH NEXT FROM authors\_cursor

INTO @au\_lname, @au\_fname

-- Check @@FETCH\_STATUS to see if there are any more rows to fetch.

WHILE @@FETCH\_STATUS = 0

BEGIN

 -- Concatenate and display the current values in the variables.

 PRINT "Author: " + @au\_fname + " " + @au\_lname

 -- This is executed as long as the previous fetch succeeds.

 FETCH NEXT FROM authors\_cursor

 INTO @au\_lname, @au\_fname

END

CLOSE authors\_cursor

DEALLOCATE authors\_cursor

GO

**C. SCROLL 커서 선언 및 기타 FETCH 옵션 사용**

다음의 예는 스크롤 커서를 만든 후, 여러 가지 FETCH 옵션을 적용하는 예이다.

USE pubs

GO

SELECT au\_lname, au\_fname FROM authors

ORDER BY au\_lname, au\_fname

--스크롤 선언

DECLARE authors\_cursor SCROLL CURSOR FOR

SELECT au\_lname, au\_fname FROM authors

ORDER BY au\_lname, au\_fname

OPEN authors\_cursor

/\*커서의 제일 마지막 열을 FETCH하고, 다음 이전 커서의 위치의 데이터를 리턴하고, 다음 현재의 커서위치로부터 두 번째 까지의 데이터를 리턴하고, 마지막으로 두번째까지의 위치에서 3개의 열을 건너뛴 뒤 그 위치의 데이터를 리턴한다. \*/

FETCH LAST FROM authors\_cursor

FETCH PRIOR FROM authors\_cursor

FETCH ABSOLUTE 2 FROM authors\_cursor

FETCH RELATIVE 3 FROM authors\_cursor

FETCH RELATIVE -2 FROM authors\_cursor

CLOSE authors\_cursor

DEALLOCATE authors\_cursor

GO

## CLOSE

현재 결과 집합을 해제하고 커서가 위치한 행에 보유된 커서 잠금을 해제하여 열린 커서를 닫습니다. CLOSE를 실행해도 데이터 구조를 액세스하여 다시 열 수 있지만 커서를 다시 열 때까지 반입과 위치 지정 업데이트는 허용되지 않습니다. CLOSE는 열려 있는 커서에만 실행할 수 있으며 선언만 되었거나 이미 닫혀 있는 커서에는 사용할 수 없습니다.

**[구문]**

CLOSE { { [ GLOBAL ] *cursor\_name* } | *cursor\_variable\_name* }

**[인수]**

🞟GLOBAL

*cursor\_name*이 전역 커서를 가리키도록 지정합니다.

🞟*cursor\_name*

열린 커서의 이름입니다. 동일한 *cursor\_name*을 가진 전역 커서와 로컬 커서가 존재할 경우 GLOBAL이 지정되어 있으면 전역 커서를 가리키고 그렇지 않으면 로컬 커서를 가리킵니다.

🞟*cursor\_variable\_name*

열린 커서와 연관된 커서 변수의 이름입니다.

다음은 커서 기반 프로세스에서 CLOSE 문의 정확한 위치를 보여 주는 예제입니다.

USE pubs

GO

DECLARE authorcursor CURSOR FOR

SELECT au\_fname, au\_lname

FROM authors

ORDER BY au\_fname, au\_lname

OPEN authorcursor

FETCH NEXT FROM authorcursor

WHILE @@FETCH\_STATUS = 0

BEGIN

 FETCH NEXT FROM authorcursor

END

CLOSE authorcursor

DEALLOCATE authorcursor

GO

## DEALLOCATE

커서 참조를 제거합니다. 마지막 커서 참조가 할당 해제될 경우 Microsoft® SQL Server™에서 커서를 구성하는 데이터 구조가 해제됩니다.

**[구문]**

DEALLOCATE { { [ GLOBAL ] *cursor\_name* } | *@cursor\_variable\_name* }

**[인수]**

*🞟cursor\_name*

이미 선언된 커서의 이름입니다. 동일한 *cursor\_name*을 가진 전역 커서와 지역 커서가 있을 경우, GLOBAL이 지정되면 *cursor\_name*은 전역 커서를 참조하고 GLOBAL이 지정되지 않으면 지역 커서를 참조합니다.

*🞟@cursor\_variable\_name*

**cursor** 변수의 이름입니다. *@cursor\_variable\_name*은 **cursor** 형식이어야 합니다.

## CURSOR\_STATUS

저장 프로시저의 호출자가 프로시저에서 주어진 매개 변수에 대해 커서 및 결과 집합을 반환했는지를 결정할 수 있는 스칼라 함수입니다.

**[구문]**

CURSOR\_STATUS
    **(**
     { **'**local**' , '***cursor\_name***'** }
         | { **'**global**' , '***cursor\_name***'** }
         | { **'**variable**' , '***cursor\_variable***'** }
    **)**

**[인수]**

'local'

커서의 원본이 로컬 커서 이름임을 나타내는 상수를 지정합니다.

'cursor\_name'

커서의 이름입니다. 커서 이름은 식별자에 대한 규칙을 따라야 합니다.

'global'

커서의 원본이 전역 커서 이름임을 나타내는 상수를 지정합니다.

'variable'

커서의 원본이 로컬 변수임을 나타내는 상수를 지정합니다.

'*cursor\_variable*'

커서 변수의 이름입니다. 커서 변수는 **cursor** 데이터 형식을 사용하여 정의되어야 합니다.

**[반환 형식] smallint**

|  |  |  |
| --- | --- | --- |
| **반환 값** | **커서 이름** | **커서 변수** |
| 1 | 커서의 결과 집합에는 행이 최소한 하나 있으며 대/소문자를 구분하지 않는 커서와 키 집합 커서에 대해 결과 집합에는 행이 최소한 하나 있습니다.동적 커서의 경우 결과 집합에는 행이 0, 1 또는 그 이상이 있습니다. | 이 변수에 할당된 커서는 열려 있으며 대/소문자를 구분하지 않는 커서와 키 집합 커서에 대해 결과 집합에는 행이 최소한 하나 있습니다.동적 커서의 경우 결과 집합에는 행이 0, 1 또는 그 이상이 있습니다. |
| 0 | 커서의 결과 집합은 비어 있습니다.\* | 이 변수에 할당된 커서는 열려 있지만 결과 집합은 확실히 비어 있습니다.\* |
| -1 | 커서는 닫혀 있습니다. | 이 변수에 할당된 커서는 닫혀 있습니다. |
| -2 | 적용할 수 없습니다. | 다음 값이 될 수 있습니다. 이전에 호출된 프로시저에서 이 OUTPUT 변수에 할당된 커서가 없습니다.커서는 이전에 호출된 프로시저에서 이 OUTPUT 변수로 할당되었지만 프로시저가 완료되면서 닫혔습니다.  그러므로, 커서는 할당 취소되고 호출하는 프로시저로 반환되지 않습니다.선언된 커서 변수에 할당된 커서가 없습니다. |
| -3 | 지정된 이름의 커서가 없습니다. | 지정된 이름의 커서 변수가 없거나 있는 경우에는 아직 커서가 할당되지 않았습니다. |

\* 동적 커서는 절대 이 결과를 반환하지 않습니다.

**커서 변수 사용**

Microsoft® SQL Server™ 2000은 **cursor** 데이터 형식의 변수도 지원합니다. 다음과 같은 두 가지 방법을 사용하여 커서와 **cursor** 변수를 연결할 수 있습니다.

/\* Use DECLARE @local\_variable, DECLARE CURSOR and SET. \*/

DECLARE @MyVariable CURSOR

DECLARE MyCursor CURSOR FOR

SELECT LastName FROM Northwind.dbo.Employees

SET @MyVariable = MyCursor

/\* Use DECLARE @local\_variable and SET \*/

DECLARE @MyVariable CURSOR

SET @MyVariable = CURSOR SCROLL KEYSET FOR

SELECT LastName FROM Northwind.dbo.Employees

커서를 **cursor** 변수와 연결하고 나면 Transact-SQL 커서 문에서 커서 이름 대신 **cursor** 변수를 사용할 수 있습니다. 저장 프로시저 출력 매개 변수를 **cursor** 데이터 형식에 할당하고 커서와 연결할 수도 있습니다. 이렇게 하면 저장 프로시저가 제어된 방법으로 지역 커서를 표시할 수 있습니다.

**API 서버 커서**

OLE DB, ODBC, ADO 및 DB-Library API는 실행된 SQL 문의 결과 집합에 대해 매핑된 커서를 지원합니다. Microsoft® SQL Server™ OLE DB 공급자, SQL Server ODBC 드라이버 및 DB-Library 동적 연결 라이브러리(DLL)는 API 서버 커서를 통해 이러한 작업을 구현합니다. API 서버 커서는 서버에서 구현되어 API 커서 함수에 의해 관리되는 커서입니다. 응용 프로그램이 API 커서 함수를 호출하면 커서 작업이 OLE DB 공급자, ODBC 드라이버 또는 DB-Library DLL에 의해 서버로 전송됩니다.

OLE DB, ODBC 및 ADO에서 API 서버 커서를 사용할 때는 다음과 같은 경우에 API의 함수나 메서드를 사용합니다.

1.연결을 엽니다.

2.API가 자동으로 각 결과 집합에 대해 매핑하는 커서의 특징을 정의하는 특성 또는 속성을 설정합니다.

3.하나 이상의 Transact-SQL 문을 실행합니다.

4.API 함수 또는 메서드를 사용하여 결과 집합의 행을 반입합니다.

DB-Library의 특수 DB-Library 커서 라이브러리 함수를 사용하여 API 서버 커서 작업을 합니다.

API 커서 특성이나 속성이 기본값으로 설정되면 SQL Server OLE DB 공급자 및 SQL Server ODBC 드라이버가 기본 결과 집합을 사용합니다. API는 기술적으로 커서에 대해 요청하지만 기본 커서 특징은 기본 결과 집합의 동작과 일치합니다. 따라서 OLE DB 공급자와 ODBC 드라이버는 기본 결과 집합을 사용하여 기본 커서 옵션을 구현합니다. 기본 결과 집합을 사용하면 응용 프로그램이 모든 Transact-SQL 문이나 일괄 처리를 실행할 수 있지만 한 연결에 대해 처리 중인 명령문을 하나만 가질 수 있습니다. 즉, 응용 프로그램이 하나의 명령문에서 반환된 결과 집합을 모두 처리하거나 취소해야만 연결에 대해 다른 명령문을 실행할 수 있습니다.

API 커서 특성 또는 속성을 기본값 이외의 값으로 설정하면 SQL Server용 OLE DB 공급자와 SQL Server ODBC 드라이버가 기본 결과 집합 대신 API 서버 커서를 사용합니다. 행을 반입하는 API 함수를 호출할 때마다 서버 왕복이 생성되어 API 서버 커서로부터 행을 반입합니다.

DB-Library 응용 프로그램은 DB-Library 커서 라이브러리 함수를 사용하여 커서를 요청합니다. DBCLIENTCURSOR를 설정하지 않으면 DB-Library 커서 라이브러리 함수가 SQL Server OLE DB 공급자 및 SQL Server ODBC 드라이버와 같은 방법으로 API 서버 커서를 사용합니다.

**API 서버 커서 제한 사항**

●응용 프로그램에서 API 서버 커서를 사용할 때는 다음 명령문을 실행할 수 없습니다.

●SQL Server가 서버 커서에서 지원하지 않는 Transact-SQL 문

●여러 결과 집합을 반환하는 일괄 처리 또는 저장 프로시저

●COMPUTE, COMPUTE BY, FOR BROWSE 또는 INTO 절을 포함하는 SELECT 문

●원격 저장 프로시저를 참조하는 EXECUTE 문

**API 서버 커서 구현**

SQL Server용 OLE DB 공급자, SQL Server ODBC 드라이버 및 DB-Library DLL은 이러한 특수 시스템 저장 프로시저를 사용하여 서버에 대해 커서 작업을 신호합니다.

●**sp\_cursoropen**은 커서 및 커서 옵션에 연결할 SQL 문을 정의한 다음 커서를 채웁니다.

●**sp\_cursorfetch**는 커서로부터 행 또는 행 블록을 반입합니다.

●**sp\_cursorclose**는 커서를 닫고 할당을 취소합니다.

●**sp\_cursoroption**은 여러 가지 커서 옵션을 설정하는 데 사용합니다.

●**sp\_cursor**는 위치 지정 업데이트를 요청하는 데 사용합니다.

●**sp\_cursorprepare**는 커서와 연관된 Transact-SQL 문이나 일괄 처리를 하나의 실행 계획으로 컴파일하지만 커서를 만들지는 않습니다.

●**sp\_cursorexecute**는 **sp\_cursorprepare**가 만든 실행 계획을 사용하여 커서를 만들고 채웁니다.

●**sp\_cursorunprepare**는 **sp\_cursorprepare**에서 실행 계획을 삭제합니다.

이러한 시스템 저장 프로시저는 API 서버 커서를 사용하는 ADO, OLE DB, ODBC 및 DB-Library 응용 프로그램의 SQL 프로필러 추적에 나타납니다. 이 저장 프로시저는 SQL Server용 OLE DB 공급자, SQL Server ODBC 드라이버 및 DB-Library DLL의 내부적 사용만을 위한 것입니다. 이러한 프로시저의 전체 기능은 데이터베이스 API의 커서 기능 사용을 통해서만 응용 프로그램에서 사용할 수 있습니다. 응용 프로그램에서 프로시저를 직접 지정할 수는 없습니다.

SQL Server가 연결에 대해 명령문을 실행할 때는 첫 번째 명령문의 모든 결과를 처리하거나 취소할 때까지 해당 연결에 대해 다른 명령문을 실행할 수 없습니다. API 서버 커서를 사용할 때도 이 규칙이 적용되지만, 응용 프로그램 관점에서는 SQL Server가 연결에 대해 지원하는 여러 활성 문을 시작한 것과 같습니다. 이는 전체 결과 집합은 서버 커서에 저장되고 **sp\_cursor** 시스템 저장 프로시저 실행 명령문만 SQL Server로 전달되기 때문입니다. SQL Server가 저장 프로시저를 실행하면 클라이언트가 결과 집합을 받자마자 다른 명령문을 실행할 수 있게 됩니다. OLE DB 공급자와 ODBC 드라이버는 항상 **sp\_cursor** 저장 프로시저로부터 모든 결과를 수신한 다음 응용 프로그램으로 제어를 반환합니다. 따라서 응용 프로그램 인터리브가 여러 활성 서버 커서에 대해 반입할 수 있습니다.

다음 표는 응용 프로그램이 두 가지 문 핸들을 사용하여 연결에 대해 동시에 두 가지 커서를 처리할 수 있는 방법을 설명한 것입니다.

|  |  |
| --- | --- |
| **문 핸들 1** | **문 핸들 2** |
| API 서버 커서가 사용할 서버 특성을 설정합니다. |   |
| SQL 문에 대해 **SQLExecDirect**를 실행합니다. ODBC 드라이버는 **sp\_cursoropen**을 호출하고 프로시저에서 반환한 결과 집합을 검색합니다. |
|   | API 서버 커서가 사용할 서버 특성을 설정합니다. |
| SQL 문에 대해 **SQLExecDirect**를 실행합니다. ODBC 드라이버는 **sp\_cursoropen**을 호출하고 프로시저에서 반환한 결과 집합을 검색합니다. |
| **SQLFetchScroll**을 실행하여 첫 번째 행 블록을 검색합니다. 드라이버는 **sp\_cursorfetch**를 호출한 다음 프로시저에서 반환한 결과 집합을 검색합니다. |   |
|   | **SQLFetchScroll**을 실행하여 첫 번째 행 블록을 검색합니다. 드라이버는 **sp\_cursorfetch**를 호출한 다음 프로시저에서 반환한 결과 집합을 검색합니다. |
| **SQLFetchScroll**을 실행하여 다른 행 블록을 검색합니다. 드라이버는 **sp\_cursorfetch**를 호출한 다음 프로시저에서 반환한 결과 집합을 검색합니다. |   |
|   | **SQLFetchScroll**을 실행하여 다른 행 블록을 검색합니다. 드라이버는 **sp\_cursorfetch**를 호출한 다음 프로시저에서 반환한 결과 집합을 검색합니다. |
| **SQLFreeStmt** 또는 **SQLCloseCursor**를 호출합니다. 드라이버는 **sp\_cursorclose**를 호출합니다. |   |
|   | **SQLFreeStmt** 또는 **SQLCloseCursor**를 호출합니다. 드라이버는 **sp\_cursorclose**를 호출합니다. |

**sp\_cursor** 저장 프로시저를 호출한 후에는 연결에 처리 중인 결과가 남아 있지 않게 되므로 모두 API 서버 커서로 실행된 경우에는 단일 연결에 대해 여러 Transact-SQL 문을 동시에 실행할 수 있습니다.

**API 서버 커서 지정**

다음은 API에서 API 서버 커서를 사용하는 방법에 대한 요약입니다.

**OLE DB**

a. 세션 개체를 열고, 명령 개체를 열고, 명령 텍스트를 지정합니다.

b. DBPROP\_OTHERINSERT, DBPROP\_OTHERUPDATEDELETE,

DBPROP\_OWNINSERT, DBPROP\_OWNUDPATEDELETE 등의 행 집합 속성을 설정하여 커서 동작을 제어합니다.

c. 명령개체를 실행합니다.

d. IRowset::GetNextRows, IRowsetLocate::GetRowsAt, IRowsetLocate::GetRowsAtBookmark

및 IRowsetScroll::GetRowsAtRatio 등의 메서드를 사용하여 결과 집합에서 행을 반입합니다.

## ODBC

e. 연결을 열고 SQLAllocHandle을 호출하여 문 핸들을 할당합니다.

f. **SQLSetStmtAttr**을 호출하여 SQL\_ATTR\_CURSOR\_TYPE, SQL\_ATTR\_CONCURRENCY

및 SQL\_ATTR\_ROW\_ARRAY\_SIZE 특성을 설정합니다. 또는 SQL\_ATTR\_CURSOR\_SCROLLABLE

및 SQL\_ATTR\_CURSOR\_SENSITIVITY 특성을 설정하여 커서 동작을 지정할 수 있습니다.

g. **SQLExecDirect** 또는 **SQLPrepare** 및 **SQLExecute**를 사용하여 Transact-SQL 문을 실행합니다.

h. **SQLFetch** 또는 **SQLFetchScroll**을 사용하여 행 또는 행 블록을 반입합니다.

## ADO

i. Connection 개체와 Recordset 개체를 정의한 다음 Connection 개체에 대해 Open 메서드를 실행합니다.

*j. CursorType* 및/또는 *LockType* 매개 변수를 지정하여 Recordset 개체에 대해 Open 메서드를 실행합니다.

k. Move, MoveFirst, MoveLast, MoveNext 및 MovePrevious 레코드 집합 메서드를 사용하여 행을 반입합니다.

## DB-Library

l. DB-Library 핵심 함수는 항상 기본 결과 집합을 사용합니다.

m. DBCLIENTCURSOR를 API 서버 커서를 사용하도록 설정하지 않고 DB-Library 커서 라이브러리 함수를 사용합니다.

**클라이언트 커서**

ODBC 및 DB-Library는 모두 클라이언트 커서를 지원하고 커서는 클라이언트에서 구현됩니다. 클라이언트 커서에서는 기본 결과 집합이 클라이언트의 전체 결과 집합을 캐시하는 데 사용되고 모든 커서 작업은 이 클라이언트 캐시에 대해 수행됩니다. Microsoft® SQL Server™ 2000 서버 커서 기능은 전혀 사용되지 않습니다. 클라이언트 커서는 전진 전용 커서 및 정적 커서만 지원하며 키 집합 커서나 동적 커서는 지원하지 않습니다.

DB-Library 클라이언트 커서는 원래 SQL Server가 서버 커서를 지원하기 전에 구현되었습니다. ODBC는 ODBC 커서 라이브러리를 사용하는 클라이언트 커서를 구현합니다. 커서 특징에 대해 기본 설정만 지원하는 ODBC 드라이버에서 사용하기 위한 것이었습니다. DB-Library 및 SQL Server ODBC 드라이버 모두 서버 커서를 통한 커서 작업을 모두 지원하므로 클라이언트 커서 사용을 제한합니다.

클라이언트 커서는 서버 커서가 모든 Transact-SQL 문 또는 일괄 처리를 지원하지 않는 제한을 완화하는 데만 사용되어야 합니다. 서버 커서와 함께 실행할 수 없는 Transact-SQL 문이나 일괄 처리에서 정적 커서가 필요하면 클라이언트 커서를 사용하는 것이 좋습니다.

**커서 유형**

ODBC, ADO 및 DB-Library는 Microsoft® SQL Server™ 2000에서 지원하는 네 가지 커서 유형을 정의합니다. DECLARE CURSOR 문이 확장되었으므로 Transact-SQL 커서에 대해 네 가지 커서 유형을 지정할 수 있습니다. 이러한 커서는 커서가 소비하는 메모리와 **tempdb** 공간 등의 리소스 및 결과 집합에 대한 변경 내용을 발견하는 능력이 다릅니다. 커서는 다음에 같은 행을 반입할 때만 행에 대한 변경 여부를 발견할 수 있습니다. 데이터 원본은 변경의 커서를 현재 반입한 행으로 알릴 수 없습니다. 변경을 발견하는 커서의 능력은 트랜잭션 격리 수준에 의해서도 영향을 받습니다.

SQL Server가 지원하는 네 가지 API 서버 커서 유형은 다음과 같습니다.

● **정적 커서 (Static Cursor)**

: 정적인 커서는 읽기 전용(Read Only)의 커서로서 데이터들은 tempdb에 임시적으로 저장되고, 이러한 데이터 복사본의 임시성 때문에 읽기 전용의 속성을 가지게 된다. 원본과 복사본인 분리되기 때문에 원본에서 복사본을 가져온 후의 원본에 대해 변경된 데이터들은 복사본에 적용되지 않는다. 때문에 보통 정적인 커서는 트랜잭션이 거의 없고(데이터의 변경이 거의 없고) 데이터의 복사본에 대한 비용에 대해 부담이 없을 경우 사용한다.(테이블의 데이터가 많을 경우,tempdb의 많은 영역을 소모할 것이다.) 사실 이러한 점은 정적인 커서의 단점으로 작용하기 때문에 굳이 커서를 만들지 않고 질의를 잘 작성하여 사용하는 편이 더 낳을지도 모른다. 하지만 정적인 커서가 아주 무용한 것은 아니고, 의사결정 지원시스템이나 데이터 마이닝(Data Mining)에서 필요한 What-if 분석에서 유용할 수도 있다. 결국 임시 테이블에서 유형화되어 원본 테이블과 분리될 때의 커서는 정적인 커서라고 보아도 좋을 것이다.

● **동적 커서 (Dynamic Cursor)**

: 동적인 커서는 말 그대로 동적이다. 즉 동적인 커서로 데이터를 가져올 때, 커서는 커서에 의해 변경된 데이터난 다른 사용자에 의해 변경된 데이터를 동적으로 가져온다. 다시 말해서 마치 커서의 FETCH가 일어날 때마다 SELECT문이 다시 실행되는 것과 같은 효과를 얻을 수 있다는 이야기이다. 동적인 커서는 지원 받는 FETCH RELATIVE 옵션은 FETCH RELATIVE n 을 설정할 경우 현재 데이터 집합의 첫번째 위치에서 n개의 행을 건너뛰게 된다.

● **전진 전용 커서 (Forward Cursor)**

: 일반적으로 SQL 서버에서 커서를 사용한다고 하면 바로 이 전방 커서일 것이다. 즉, T-SQL에서 사용하는 커서는 바로 전방 커서이다. 사용빈도가 높고, 효율성이 좋은 전방 커서를 한층 더 최적화 시킨 보다 빠른 전방 커서(Fast Forward-Only Cursor)를 제공한다. 전방 커서는 동적인 커서의 기능에 FETCH 형태의 NEXT문을 가능하게 하는 커서라고 할 수 있다. 전방 커서를 명시적으로 사용하려면 커서문에 INSENSITIVE, SCROLL, FORWARD ONLY가 지정되지 않는 경우이고 만약 질의에 GROUP BY 나 UNION절이 사용되면 커서는 정적 커서로 작성된다.

● **키 집합 커서 (Keyset Cursor)**

: 키셋 커서는 SELECT로 질의한 결과 집합 중 WHERE절에 의해 지정된 조건데이터가 키로 지정되고, 이 키 값만이 tempdb에 저장된다. 키셋 커서가 정적인 커서와 다른 점은 정적인 커서가 읽기 전용이기 때문에 원본의 변경 사항이 복사본에 새로이 적용되지 않지만 키셋의 경우에는 변경사항을 적용 받을 수 있다는 것이다. 즉, 키셋 커서에 의해 결과 집합을 리턴 하고, 그 사이 원본 테이블의 데이터에 변경이 생긴 후 다시 키셋 커서에 의해 데이터를 가져오게 되면 변경된 데이터가 적용 되어 있는 것을 확인할 수 있다. 이러한 기능을 위해 키셋 커서는 SELECT문에서 사용되는 모든 테이블에 대해 유일한 인덱스(UNIQUE INDEX)를 필요로 한다. 다시 말해서 테이블에 PRIMARY 키나 UNIQUE 인덱스를 만들면, 이 테이블은 키셋 커서를 통하여 데이터를 조회 할 수 있음을 의미하는 것이다.

**자동 반입 옵션 사용**

DECLARE CURSOR의 FAST\_FORWARD를 지정하거나 ODBC 응용 프로그램에서 SQL\_CO\_FFO를 지정하여 어느 정도 성능을 향상시킬 수 있지만 ODBC 응용 프로그램에서 SQL\_CO\_FFO\_AF를 지정하여 자동 반입 옵션을 설정하면 성능이 크게 향상됩니다. 자동 반입을 사용하면 네트워크 소통량을 크게 줄일 수 있는 두 가지 최적화가 가능해집니다.

●커서가 열려 있으면 커서에서 첫 번째 행이나 일괄 행이 자동으로 반입됩니다. 따라서 네트워크를 통해 반입 요청을 보낼 필요가 없습니다.

●반입이 커서 끝에 도달하면 커서가 자동으로 닫힙니다. 따라서 네트워크를 통해 닫기 요청을 따로 보낼 필요가 없습니다.

대부분의 경우 응용 프로그램의 메모리에 캐시할 수 있을 정도로 비교적 작은 결과 집합이 있는 커서를 처리할 때 성능이 크게 향상됩니다. 자동 반입 기능이 설정된 빠른 전진 전용 커서를 사용하면 가장 효율적으로 결과 집합을 ODBC 응용 프로그램으로 가져올 수 있습니다. *n*개 행을 포함하는 커서에 대해 자동 반입 옵션을 설정하면 ODBC 응용 프로그램이 다음을 할 수 있습니다.

●행 집합 크기를 *n*+1로 지정합니다.

●*n*+1 변수 배열을 할당하여 결과 집합 열의 데이터를 포함합니다.

●결과 집합 열을 배열에 바인딩합니다.

●커서를 생성하는 SQL 문을 실행합니다.

SQL Server ODBC 드라이버가 문을 실행하면 커서를 열도록 요청합니다. 자동 옵션이 설정되었으므로 서버는 *n*개 행을 반입하여 다시 보냅니다. 서버는 클라이언트로 반환하는 각 네트워크 패킷에 되도록 많은 행을 넣습니다. 서버가 *n*+1의 행을 반입하려고 하면 커서의 끝을 발견하여 자동으로 커서를 닫습니다. 그러면 응용 프로그램이 **SQLCloseCursor** 또는 **SQLFreeStmt**를 실행할 때 ODBC 드라이버가 닫기 요청을 서버로 보낼 필요가 없습니다. 전체 작업은 클라이언트에서 서버로 보내는 한 패킷에서만 수행되며 서버에서 클라이언트로 반환되는 패킷 수는 최소화됩니다.

**빠른 전진 전용 커서의 암시적 변환**

빠른 전진 전용 커서는 다음과 같은 경우 암시적으로 다른 커서 유형으로 변환됩니다.

●SELECT 문이 트리거 없는 하나 이상의 테이블을 트리거와 조인하면 커서가 정적 커서로 변환됩니다.

●SELECT 문이 **text**, **ntext** 또는 **image** 열을 참조하는 경우 SQL Server용 OLE DB 공급자나 SQL Server ODBC 드라이버를 사용하면 커서가 동적 커서로 변환됩니다.

●빠른 전진 전용 커서가 읽기 전용이 아니면 동적 커서로 변환됩니다.

●SELECT 문이 연결된 서버의 원격 테이블을 하나 이상 참조하는 분산 쿼리이면 커서가 키 집합 커서로 변환됩니다.

●SELECT 문이 **text**, **ntext** 또는 **image** 열과 TOP 절을 참조하는 경우에는 커서가 키 집합 커서로 변환됩니다.

**커서 동작**

ODBC와 Transact-SQL 커서 모두 SQL-92 커서 동작인 스크롤 가능 여부와 민감도를 사용한 커서 특징 지정을 지원합니다.

커서 동작은 다음과 같은 방법으로 지정됩니다.

●Transact-SQL 커서는 DECLARE 문의 CURSOR 키워드보다 먼저 SCROLL 및 INSENSITIVE를 지정하는 SQL-92 구문을 사용합니다.

●ODBC API는 SQL\_ATTR\_CURSOR\_SCROLLABLE 및 SQL\_ATTR\_CURSOR\_SENSITIVITY 문 특성을 사용합니다.

**스크롤 가능**

커서의 스크롤 가능 동작은 커서가 지원하는 반입 옵션을 정의합니다.

DECLARE 문에서 SCROLL 키워드를 지정하거나 SQL\_ATTR\_CURSOR\_SCROLLABLE을 SQL\_SCROLLABLE로 설정하면 커서가 모든 반입 옵션을 지원합니다.

SQL-92 스타일의 DECLARE 문에서 SCROLL 키워드를 생략하거나 SQL\_ATTR\_CURSOR\_SCROLLABLE을 SQL\_NONSCROLLABLE로 설정하면 커서가 FETCH NEXT 작업만 지원합니다.

**민감도**

커서의 민감도는 커서를 작성할 때 사용하는 기본 행에 대한 업데이트를 커서를 통해 볼 수 있는지의 여부를 정의합니다. 민감도는 또한 커서를 통해 업데이트할 수 있는지의 여부도 정의합니다.

Transact-SQL DECLARE 문에서 INSENSITIVE 키워드를 지정하거나 ODBC에서 SQL\_ATTR\_CURSOR\_SENSITIVITY를 SQL\_UNSPECIFIED 또는 SQL\_INSENSITIVE로 설정하면 커서가 데이터 수정을 반영하지 않습니다. 커서는 읽기 전용이며 업데이트를 지원하지 않습니다.

Transact-SQL DECLARE 문에서 INSENSITIVE 키워드를 생략하거나 ODBC에서 SQL\_ATTR\_CURSOR\_SENSITIVITY를 SQL\_SENSITIVE로 설정하면 커서가 현재 사용자가 만들거나 다른 사용자가 커밋한 데이터 수정 내용을 반영합니다. 위치 지정 업데이트는 읽기 전용 커서를 사용할 때를 제외하고 커서를 사용하여 수행할 수 없습니다.

**커서 잠금**

Microsoft® SQL Server™에서 커서 정의의 SELECT 문은 다른 SELECT 문에 적용되는 트랜잭션 잠금 규칙과 같은 경우가 많습니다. 그러나 커서에서 다른 스크롤 잠금 집합은 커서 동시성 수준의 사양을 기준으로 얻을 수 있습니다.

커서 정의의 SELECT 문을 포함하여 SELECT 문을 통해 얻은 트랜잭션 잠금은 다음에 의해 제어됩니다.

●연결에 대한 트랜잭션 격리 수준 설정

●FROM 절에서 지정한 잠금 참고

이러한 잠금은 커서와 독립 SELECT 문 모두에 대해 현재 트랜잭션이 끝날 때까지 보유됩니다. SQL Server가 자동 커밋 모드에서 실행될 때는 각 개별 SQL 문이 트랜잭션이 되고 잠금은 문이 끝날 때 해제됩니다. SQL Server가 명시적 또는 암시적 트랜잭션 모드에서 실행 중일 때는 트랜잭션이 커밋되거나 롤백될 때까지 잠금이 보유됩니다.

예를 들어, 다음 두 Transact-SQL 예에 수행된 잠금은 본질적으로 동일합니다.

/\* 예 1 \*/

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

GO

BEGIN TRANSACTION

GO

SELECT \* FROM authors

GO

/\* 예 2 \*/

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ

GO

BEGIN TRANSACTION

GO

DECLARE abc CURSOR STATIC FOR

SELECT \* FROM authors

GO

OPEN abc

GO

트랜잭션 격리 수준을 반복 읽기로 설정한다는 것은 예 1의 독립 SELECT 문과 예 2의 DECLARE CURSOR에 포함된 SELECT 문 모두 읽은 각 행에 대해 공유 잠금을 생성하고, 이 공유 잠금은 트랜잭션이 커밋되거나 롤백될 때까지 보유된다는 것을 의미합니다.

**잠금 얻기**

커서가 독립 SELECT 문과 같은 규칙을 따르기는 하지만 얻은 트랜잭션 잠금 유형에 관계 없이 잠금은 다른 시간에 얻어집니다. 독립 SELECT나 커서에 의해 생성된 잠금은 항상 행을 검색할 때 얻어집니다. 독립 SELECT의 경우 문을 실행하면 모든 행이 검색됩니다. 그러나 커서는 커서 유형에 따라 행을 검색하는 시기가 다릅니다.

정적 커서는 커서가 열릴 때 전체 결과 집합을 검색합니다. 이 경우 열릴 때 결과 집합의 각 행이 잠깁니다.

키 집합 커서는 커서가 열릴 때 결과 집합의 각 행의 키를 검색합니다. 이 경우 열릴 때 결과 집합의 각 행이 잠깁니다.

일반 전진 전용 커서를 포함하는 동적 커서는 반입될 때까지 행을 검색하지 않습니다. 잠금도 반입될 때까지 얻어지지 않습니다.

빠른 전진 전용 커서는 쿼리 최적화 프로그램이 선택한 실행 계획에 따라 잠금을 얻는 시기가 달라집니다. 동적 커서를 선택하면 행이 반입될 때까지 잠기지 않습니다. 작업 테이블이 생성되면 행이 작업 테이블로 읽혀지고 열릴 때 잠깁니다.

커서는 고유한 동시성 사양도 지원하며 그 중 일부는 각 반입 시 행에 대해 추가 잠금을 생성합니다. 이러한 스크롤 잠금은 다음 반입 작업 또는 커서가 잠길 때 중 먼저 일어나는 시기까지 보유됩니다. 커밋 시 커서를 열어 두는 연결 옵션이 설정되면 이러한 잠금이 커밋 또는 롤백 작업 중에도 보유됩니다.

**커서 및 트랜잭션**

Microsoft® SQL Server™ 2000은 커밋 및 롤백 시 커서를 닫을 것인지 열어 둘 것인지를 제어하는 연결 또는 데이터베이스 옵션 설정을 지원합니다.

커밋이나 롤백 시 커서를 닫도록 옵션을 설정하면 커서를 닫을 때 모든 스크롤 잠금이 자동으로 해제됩니다. 커밋 시 커서를 계속 열어 두도록 옵션을 설정하면 활성 스크롤 잠금이 다음 반입 또는 커서를 닫을 때까지 보유됩니다. 커서의 행을 포함하여 모든 트랜잭션 잠금은 커서의 개방 여부에 관계 없이 트랜잭션 커밋 또는 롤백 시 해제됩니다.

**커서 동시성**

Microsoft® SQL Server™ 2000은 서버 커서에 대한 네 가지 동시성 옵션을 지원합니다.

●READ\_ONLY

●OPTIMISTIC WITH VALUES

●OPTIMISTIC WITH ROW VERSIONING

●SCROLL LOCKS

**READ\_ONLY**

커서를 통한 위치 지정 업데이트는 지원되지 않으며 결과 집합을 구성하는 행에 대해 잠금이 보유되지 않습니다.

**OPTIMISTIC WITH VALUES**

낙관적 동시성 제어는 트랜잭션 제어 이론의 표준입니다. 낙관적 동시성 제어는 커서가 열린 후 행이 업데이트되는 동안 다른 사용자가 행을 업데이트할 가능성이 거의 없는 경우에 사용합니다. 이 옵션으로 커서를 열면 원본 행에 대한 잠금이 보유되지 않아 처리량이 최대화됩니다. 사용자가 행을 수정하려고 하면 행을 마지막 반입할 때 검색된 값과 현재 행 값이 비교됩니다. 변경된 값이 있으면 다른 사용자가 행을 이미 업데이트했다는 것을 인식하여 오류를 반환합니다. 값이 같으면 서버가 수정을 수행합니다.

이 동시성 옵션을 설정하면 사용자나 프로그래머가 때때로 발생하는 다른 사용자의 행 수정 오류를 처리해야 합니다. 이 오류를 받은 응용 프로그램은 일반적으로 커서를 새로 고치고 새 값을 얻은 다음 사용자가 새 값을 수정할 것인지를 결정하게 합니다. SQL Server 6.5 이전 버전에서는 **text**, **ntext** 및 **image** 열은 동시성 비교에 사용되지 않습니다.

**OPTIMISTIC WITH ROW VERSIONING**

이 낙관적 동시성 제어 옵션은 행 버전 지정을 기반으로 합니다. 행 버전을 지정하면 행을 커서로 읽은 후 행이 변경되었는지의 여부를 서버가 결정할 때 사용하는 일부 유형의 버전 식별자가 원본 테이블에 있어야 합니다. SQL Server에서는 이 기능이 데이터베이스에서 상대적 수정 시퀀스를 나타내는 이진 숫자인 **timestamp** 데이터 형식에 의해 제공됩니다. 각 데이터베이스에는 전역적 현재 타임스탬프 값인 **@@**DBTS가 있습니다. **timestamp** 열이 있는 행이 어떤 방법으로든 수정될 때마다 SQL Server는 현재 **@@**DBTS 값을 **timestamp** 열에 저장하고 **@@**DBTS를 증가시킵니다. 테이블에 **timestamp** 열이 있으면 타임스탬프가 행 수준에 적용됩니다. 그러면 서버가 행이 행의 현재 타임스탬프 값과 행이 마지막으로 반입될 때 저장된 타임스탬프 값을 비교하여 행의 업데이트 여부를 결정합니다. 서버는 모든 열의 값을 비교할 필요 없이 **timestamp** 열만 비교하면 됩니다. 응용 프로그램이 **timestamp** 열이 없는 테이블의 행 버전을 지정하여 낙관적 동시성을 요청하면 커서가 기본적으로 값을 기반으로 낙관적 동시성을 제어합니다.

**SCROLL LOCKS**

이 옵션은 응용 프로그램이 커서 결과 집합으로 읽을 때 원본 데이터베이스 행을 잠그는 비관적 동시성 제어를 구현합니다. 서버 커서를 사용하면 커서로 읽을 때 행에 대해 업데이트 잠금이 얻어집니다. 트랜잭션 내에서 커서를 열면 트랜잭션이 커밋되거나 롤백될 때까지 트랜잭션 업데이트 잠금이 보유됩니다. 커서 잠금은 다음 행을 반입할 때 삭제됩니다. 커서가 트랜잭션 밖에서 열려 있으면 다음 행을 반입할 때 잠금이 삭제됩니다. 따라서 사용자가 전체 비관적 동시성 제어를 원할 때마다 트랜잭션에서 커서를 열어야 합니다. 업데이트 잠금은 다른 작업이 업데이트 또는 단독 잠금을 얻지 못하게 하여 다른 작업이 행을 업데이트하지 못하게 합니다. 그러나 업데이트 잠금은 공유 잠금을 차단하지 못하므로 다른 작업이 업데이트 잠금과 읽기를 함께 요청하지 않는 한 행을 읽지 못하게 할 수 없습니다.

**스크롤 잠금**

이러한 커서 동시성 옵션은 커서 정의의 SELECT 문에서 지정한 잠금 참고에 따라 스크롤 잠금을 생성할 수 있습니다. 스크롤 잠금은 반입 시 각 행에 대해 얻어지며 다음 반입 작업 또는 커서가 잠길 때 중 먼저 일어나는 시기까지 보유됩니다. 다음 반입 시 서버는 새 반입에서 행에 대해 스크롤 잠금을 얻은 다음 이전 반입에서 행에 대해 얻은 스크롤 잠금을 해제합니다. 스크롤 잠금은 트랜잭션 잠금과 별개의 것이며 커밋 또는 롤백 작업 후까지 지속될 수 있습니다. 커밋 시 커서를 닫는 옵션을 해제하면 COMMIT을 수행해도 열려 있는 커서가 닫히지 않으며 커밋 후에도 스크롤 잠금이 보유되어 반입된 데이터의 격리가 유지됩니다.

얻어지는 스크롤 잠금의 유형은 커서 동시성 옵션과 커서 SELECT 문의 잠금 참고에 따라 달라집니다.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **잠금 참고** | **읽기 전용** | **값이 있는 낙관적 동시성** | **행 버전을 지정한 낙관적 동시성** | **잠금** |
| 참고 없음 | 잠금 없음 | 잠금 없음 | 잠금 없음 | 업데이트 |
| |NOLOCK | 잠금 없음 | 잠금 없음 | 잠금 없음 | 잠금 없음 |
| |HOLDLOCK | 공유 | 공유 | 공유 | 업데이트 |
| | UPDLOCK | 오류 | 업데이트 | 업데이트 | 업데이트 |
| | TABLOCKX | 오류 | 잠금 없음 | 잠금 없음 | 업데이트 |
| 나머지 | 잠금 없음 | 잠금 없음 | 잠금 없음 | 업데이트 |

\*NOLOCK 참고를 지정하면 이 참고가 지정된 테이블이 커서를 통해 읽기 전용이 됩니다.

**커서 동시성 옵션 지정**

동시성 옵션은 각 커서 환경에 따라 다르게 지정됩니다.

●Transact-SQL 커서

DECLARE CURSOR 문에서 READ\_ONLY, SCROLL\_LOCK 및 OPTIMISTIC 키워드를 지정합니다. OPTIMISTIC 키워드는 행 버전을 지정한 낙관적 동시성을 지원하며 Transact-SQL 커서는 값이 있는 낙관적 동시성 옵션을 지원하지 않습니다.

●ADO 응용 프로그램

**Recordset** 개체의 **LockType** 속성에서 **adLockReadOnly**, **adLockPessimistic**, **adLockOptimistic** 또는 **adLockBatchOptimistic**을 지정합니다.

●ODBC 응용 프로그램

문 속성 SQL\_ATTR\_CONCURRENCY to SQL\_CONCUR\_READ\_ONLY, SQL\_CONCUR\_ROWVER, SQL\_CONCUR\_VALUES 또는 SQL\_CONCUR\_LOCK을 설정합니다.

●DB-Library 응용 프로그램

**dbcursoropen** 매개 변수 *concuropt*를 CUR\_READONLY, CUR\_OPTCC(행 버전을 지정한 낙관적 동시성용), CUR\_OPTCCVAL 또는 CUR\_LOCKCC로 설정합니다.

**커서 트랜잭션 격리 수준**

특정 커서의 트랜잭션 잠금 동작은 커서 동시성 설정의 잠금 동작과 커서 SELECT에서 지정한 잠금 참고 및 트랜잭션 격리 수준 옵션을 통해 결정됩니다.

Microsoft® SQL Server™ 2000은 다음과 같은 커서 트랜잭션 격리 수준을 지원합니다.

**커밋된 읽기**

SQL Server는 행을 커서로 읽는 동안 공유 잠금을 얻지만 행을 읽고 난 즉시 잠금을 해제합니다. 공유 잠금 요청은 단독 잠금에 의해 차단되므로 커서는 다른 작업이 업데이트했지만 아직 커밋하지 않은 행을 읽을 수 없습니다. 커밋된 읽기는 SQL Server 및 ODBC에 대한 기본 격리 수준 설정입니다.

**커밋되지 않은 읽기**

SQL Server는 행을 커서로 읽는 동안 잠금을 요청하지 않으며 단독 잠금을 보장하지 않습니다. 커서는 업데이트되었지만 아직 커밋되지 않은 값으로 채울 수 있습니다. SQL Server에서는 사용자의 모든 잠금 트랜잭션 제어 메커니즘이 무시됩니다.

**반복 읽기 또는 순차 가능**

SQL Server는 READ COMMITTED 등을 통해 커서로 행을 읽을 때 각 행에 대해 공유 잠금을 요청하지만 트랜잭션 내에서 커서가 열려 있으면 행을 읽은 다음 해제하지 않고 트랜잭션이 끝날 때까지 공유 잠금이 보유됩니다. 이렇게 하면 SELECT 문에서 HOLDLOCK을 지정하는 것과 같은 결과가 나타납니다.

**커서 프로그래밍 정보**

Microsoft® SQL Server™ 2000 응용 프로그램을 개발할 때 정확한 커서 옵션을 선택하는 것은 매우 중요합니다.

블록 커서를 사용하면 클라이언트와 SQL Server 간의 네트워크 왕복 수가 줄기 때문에 성능이 향상됩니다. 요청한 커서 유형에서 지원하지 않는 Transact-SQL 문을 실행하면 SQL Server가 커서 유형을 암시적으로 변환합니다. SQL Server는 큰 키 집합 커서를 비동기식으로 채워서 커서가 열린 후 첫 번째 행을 반입할 수 있을 때까지의 시간을 단축시킵니다.

**커서 유형 선택**

선택할 커서 유형은 다음과 같은 몇 가지 변수에 따라 결정됩니다.

●결과 집합의 크기

●필요한 데이터의 비율

●커서 열기 성능

●스크롤 또는 위치 지정 업데이트 등 커서 작업에 필요한 항목

●다른 사용자의 데이터 수정에 대한 가시성 수준

업데이트가 수행되지 않은 경우 결과 집합이 작을 때는 기본 설정을 사용해도 상관 없지만 행을 모두 검색하기 전에 응답을 찾고자 하는 큰 결과 집합에서는 동적 커서를 사용하는 것이 좋습니다.

커서 유형 선택 규칙

커서 유형을 선택할 때는 다음과 같은 간단한 규칙이 적용됩니다.

●한 행을 반환하는 단일 선택이나 기타 작은 결과 집합에 대해서는 기본 설정을 사용합니다. 작은 결과 집합을 클라이언트에 캐시한 다음 서버에 커서 구현을 요청하는 대신 캐시를 통해 스크롤하는 것이 더 효율적입니다.

●보고서를 작성하는 등 전체 결과 집합을 클라이언트에 반입할 때는 기본 설정을 사용합니다. 기본 결과 집합을 사용하면 데이터를 클라이언트로 신속하게 전달할 수 있습니다.

●응용 프로그램에서 위치 지정 업데이트를 사용할 때는 기본 결과 집합을 사용할 수 없습니다.

●응용 프로그램이 여러 개의 활성 문을 사용할 때는 기본 결과 집합을 사용할 수 없습니다. 여러 활성 문을 지원하는 용도로만 커서를 사용할 때는 빠른 전진 전용 커서를 선택하십시오.

●여러 결과 집합을 생성할 Transact-SQL 또는 일괄 Transact-SQL 문에 대해서는 기본 결과 집합을 사용해야 합니다.

●동적 커서는 정적 커서나 키 집합 커서보다 빨리 열립니다. 정적 커서와 키 집합 커서를 열 때는 내부 임시 작업 테이블을 작성해야 하지만 동적 커서에는 필요하지 않습니다.

●조인 시 키 집합 커서 및 정적 커서는 동적 커서보다 빠를 수 있습니다.

●절대 반입을 수행하려면 키 집합 커서나 정적 커서를 사용해야 합니다.

●정적 커서와 키 집합 커서는 **tempdb**의 사용량이 늘어납니다. 정적 서버 커서는 **tempdb**에 전체 커서를 작성하며 키 집합 커서는 **tempdb**에 키 집합을 작성합니다.

●롤백 작업 중 커서가 계속 열려 있으면 동기적 정적 커서를 사용하고 CURSOR\_CLOSE\_ON\_COMMIT를 OFF로 설정하십시오.

서버 커서를 사용하면 API 반입 함수나 메서드를 호출할 때마다 서버로의 왕복이 일어납니다. 응용 프로그램은 블록 커서를 사용하고 각 반입 시 비교적 큰 행 수를 반환하여 이러한 왕복을 최소화해야 합니다.