FileMaker® 15

SQL 참조

FileMaker, Inc.
5201 Patrick Henry Drive
Santa Clara, California 95054

자세한 정보를 보려면 다음 웹 사이트(http://www.filemaker.com/kr)를 방문하십시오.

Edition: 01

차례

제 1장
소개

데이터베이스 개발자로서 여러분은 SQL에 대한 지식 없이도 FileMaker Pro를 사용하여 데이터베이스 솔루션을 생성할 수 있습니다. 그러나 SQL에 대한 일부 지식이 있는 경우, FileMaker 데이터베이스 파일을 ODBC나 JDBC 데이터 원본으로 사용하고 ODBC와 JDBC를 사용하여 데이터를 다른 응용 프로그램과 공유할 수 있습니다. FileMaker Pro ExecuteSQL 함수를 사용하여 FileMaker Pro 데이터베이스 내 테이블 일치 항목에서 데이터를 검색할 수도 있습니다.

이 참조에서는 FileMaker에서 지원되는 SQL 문 및 표준을 설명합니다. FileMaker ODBC 및 JDBC 클라이언트 드라이버는 이 참조에서 설명한 모든 SQL 문을 지원합니다. FileMaker Pro ExecuteSQL 함수는 SELECT 문만 지원합니다.

이 참조 정보

- 이전 버전의 FileMaker Pro와 ODBC 및 JDBC 사용에 관한 정보는 http://www.filemaker.com/documentation/ko 사이트를 참조하십시오.
- 이 참조에서는 FileMaker Pro 함수 사용, ODBC와 JDBC 응용 프로그램 코딩 및 SQL 쿼리 구성의 기본에 익숙해 있다고 가정합니다. 이 주제에 관한 자세한 정보는 기타 업체 책을 참조하십시오.
- 이 참조는 특정 FileMaker Pro Advanced 기능을 설명하는 경우를 제외하면, "FileMaker Pro"를 사용하여 FileMaker Pro 및 FileMaker Pro Advanced를 참조합니다.

FileMaker 설명서 위치

- FileMaker Pro에서 **도움말** 메뉴 > **제품 설명서**를 선택합니다.
- FileMaker Server 관리 콘솔에서 **도움말** 메뉴 > **FileMaker Server 제품 설명서**를 선택합니다.
- 추가 FileMaker 설명서를 보고 다운로드하거나 더 알아보려면 http://www.filemaker.com/documentation/ko 사이트를 방문하십시오.

SQL 정보

SQL 또는 Structured Query Language는 관계형 데이터베이스에서 데이터를 쿼리하도록 설계했던 프로그래밍 언어입니다. 데이터베이스를 쿼리하는 데 사용된 기본 명령문은 SELECT 문입니다.

데이터베이스를 쿼리하기 위한 언어 외에, SQL은 데이터를 추가, 업데이트 및 삭제할 수 있는 데이터 조작을 수행하기 위한 명령문을 제공합니다.

SQL은 데이터 정의를 수행하기 위한 명령문도 제공합니다. 이 명령문을 사용하여 테이블과 인덱스를 생성하고 수정할 수 있습니다.

FileMaker에서 지원되는 SQL 문 및 표준은 제 2장, "지원 표준."에서 설명됩니다.

FileMaker 데이터베이스를 데이터 원본으로 사용하기

FileMaker 데이터베이스를 ODBC 또는 JDBC 데이터 원본으로 호스트하면, FileMaker 데이터를 ODBC 및 JDBC 호환 응용 프로그램과 공유할 수 있습니다. 응용 프로그램은 FileMaker 클라이언트 드라이버를 사용하여 FileMaker 데이터 원본에 연결하며, ODBC나 JDBC를 사용하여 SQL 쿼리를 구성하고 실행하며 FileMaker 데이터베이스 솔루션에서 검색된 데이터를 처리합니다.

FileMaker 소프트웨어를 ODBC 및 JDBC 응용 프로그램용 데이터 원본으로 사용할 수 있는 방법에 관한 추가 정보는 FileMaker ODBC 및 JDBC 설명서를 참조하십시오.

FileMaker ODBC 및 JDBC 클라이언트 드라이버는 이 참조에서 설명한 모든 SQL 문을 지원합니다.

ExecuteSQL 함수 사용하기

FileMaker Pro ExecuteSQL 함수를 사용하여 관계형 그래프에서 이름이 지정된 테이블 일치 항목에서 검색할 수 있지만 정의된 관계와는 별도입니다. 테이블 연결 또는 테이블 간 관계를 생성하지 않고 다중 테이블에서 데이터를 검색할 수 있습니다. 일부 경우에 ExecuteSQL 함수를 사용하여 사용자의 관계형 그래프를 간소화할 수 있습니다.

ExecuteSQL 함수와 쿼리한 필드가 레이아웃에 있지 않아도 되므로 ExecuteSQL 함수를 사용하여 레이아웃 문맥과 별개로 데이터를 검색할 수 있습니다. 이러한 독립적인 문맥 때문에 ExecuteSQL 함수를 스크립트에서 사용하면 스크립트의 이동성을 향상할 수 있습니다. 차트와 보고를 비롯하여 계산을 지정할 수 있는 어디에서나 ExecuteSQL 함수를 사용할 수 있습니다.

ExecuteSQL 함수는 8 페이지의 "SELECT 문". 부분에서 설명하는 SELECT 문만 지원합니다.

ExecuteSQL 함수는 중괄호({})가 없는 SQL-92 구문 ISO 날짜 및 시간 포맷만 승인합니다. ExecuteSQL 함수는 중괄호 안의 ODBC/JDBC 포맷 날짜, 시간 및 타임스탬프 상수를 승인하지 않습니다.

ExecuteSQL 함수의 사용과 구문에 관한 정보는 FileMaker Pro 도움말을 참조하십시오.

제 2장
지원 표준

이 참조에서는 FileMaker에서 지원되는 SQL 문 및 구성을 설명합니다. FileMaker ODBC 및 JDBC 클라이언트 드라이버는 이 장에서 설명한 모든 SQL 문을 지원합니다. FileMaker Pro ExecuteSQL 함수는 SELECT 문만 지원합니다.

클라이언트 드라이버를 사용하여 ODBC 또는 JDBC 호환 응용 프로그램에서 FileMaker 데이터베이스 솔루션에 접근합니다. FileMaker 데이터베이스 솔루션을 FileMaker Pro 또는 FileMaker Server에서 호스트할 수 있습니다.

- ODBC 클라이언트 드라이버는 ODBC 3.0 레벨 1을 지원합니다.
- JDBC 클라이언트 드라이버는 JDBC 3.0 스펙에 대해 일부 지원을 제공합니다.
- ODBC 및 JDBC 클라이언트 드라이버는 일부 SQL-92 중간 기능이 있는 SQL-92 항목 단계별 규칙을 모두 지원합니다.

유니코드 문자 지원

ODBC 및 JDBC 클라이언트 드라이버는 유니코드 API를 지원합니다. 하지만 클라이언트 드라이버를 사용하는 사용자 설정 응용 프로그램을 생성하려면 필드 이름, 테이블 이름 및 파일 이름으로 ASCII를 사용하십시오(유니코드가 아닌 쿼리 도구나 응용 프로그램이 사용되는 경우).

메모 유니코드 데이터를 삽입하고 검색하려면 `SQL_C_WCHAR`를 사용하십시오.

SQL 문

ODBC 및 JDBC 클라이언트 드라이버는 다음 SQL 문에 대한 지원을 제공합니다.

- SELECT(페이지 8)
- DELETE(페이지 17)
- INSERT(페이지 17)
- UPDATE(페이지 19)
- CREATE TABLE(페이지 20)
- TRUNCATE TABLE(페이지 21)
- ALTER TABLE(페이지 22)
- CREATE INDEX(페이지 22)
- DROP INDEX(페이지 23)

클라이언트 드라이버는 ODBC SQL 및 JDBC SQL 데이터 유형으로 맵핑된 FileMaker 데이터 유형도 지원합니다. 데이터 유형 변환은 FileMaker ODBC 및 JDBC 설명서를 참조하십시오. SQL 쿼리 구성에 관한 추가 정보는 기타 업체 설명서를 참조하십시오.

메모 ODBC 및 JDBC 클라이언트 드라이버는 FileMaker 포털을 지원하지 않습니다.

SELECT 문

SELECT 문을 사용하여 요청할 열을 지정합니다. SELECT 문 다음에 검색할(예: last_name) 열 표현식(필드 이름과 유사)을 작성하십시오. 표현식은 수학 연산 또는 문자열 조작(예: SALARY * 1.05)을 포함할 수 있습니다.

SELECT 문은 다양한 절을 사용할 수 있습니다.

```
SELECT [DISTINCT] {* | column_expression [[AS] column_alias],...}
FROM table_name [table_alias], ...
[ WHERE expr1 rel_operator expr2 ]
[ GROUP BY {column_expression, ...} ]
[ HAVING expr1 rel_operator expr2 ]
[ UNION [ALL] (SELECT...) ]
[ ORDER BY {sort_expression [DESC | ASC]}, ... ]
[ OFFSET n {ROWS | ROW} ]
[ FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
[ FOR UPDATE [OF {column_expression, ...}] ]
```

괄호 안의 항목은 옵션입니다.

column_alias는 열에 추가 설명이 포함된 이름을 지정하거나 긴 열 이름을 줄여 사용할 수 있습니다. 예를 들어, 별칭 department를 dept열에 지정합니다.

```
SELECT dept AS department FROM emp
```

필드 이름이 테이블 이름 또는 테이블 별칭 앞에 있을 수 있습니다. 예를 들어, EMP.LAST_NAME 또는 E.LAST_NAME이며, 여기에서 E는 테이블 EMP의 별칭입니다.

DISTINCT 연산자는 첫 번째 열 표현식 앞에 올 수 있습니다. 이 연산자는 쿼리 결과에서 중복되는 행을 제거합니다. 예를 들어 다음과 같습니다.

```
SELECT DISTINCT dept FROM emp
```

SQL 절

ODBC 및 JDBC 클라이언트 드라이버는 다음 SQL 절에 대한 지원을 제공합니다.

다음 SQL 절 사용	작업
FROM(페이지 9)	SELECT 문에서 사용되는 테이블을 표시합니다.
WHERE(페이지 10)	레코드를 검색하기 위해 충족되어야 하는 조건을 지정합니다(예: File Maker Pro 찾기 요청).
GROUP BY(페이지 11)	반환된 값을 그룹화해야 하는 하나 이상의 필드 이름을 지정합니다. 이 절은 각 가룹에 대해 하나의 행을 반환하여 집계 값 세트를 반환하는 데 사용됩니다(예: FileMaker Pro 하위 요약).
HAVING(페이지 11)	레코드 그룹에 대한 조건을 지정합니다(예: 월급이 총 $200,000 이상인 부서만 표시).
UNION(페이지 12)	둘 이상의 SELECT 문의 결과를 단일 결과로 결합합니다.
ORDER BY(페이지 12)	레코드 정렬 방법을 표시합니다.
OFFSET(페이지 13)	행 검색 시작 전에 건너 뛸 행 수를 언급합니다.
FETCH FIRST(페이지 13)	검색할 행 수를 지정합니다. 쿼리가 지정한 행의 수 미만을 표시하는 경우 더 적은 수의 행이 반환될 수 있더라도 지정된 수 이상의 행은 반환되지 않습니다.
FOR UPDATE(페이지 14)	SQL 커서를 통해 현재 위치 업데이트 또는 현재 위치 삭제를 수행합니다.

메모 열이 없는 테이블에서 데이터를 검색하려는 경우, SELECT 문은 어느 것도 반환하지 않습니다.

FROM 절

FROM 절은 SELECT 문에서 사용된 테이블을 표시합니다. 포맷은 다음과 같습니다.

```
FROM table_name [table_alias] [, table_name [table_alias]]
```

table_name은 현재 데이터베이스의 테이블 이름입니다. 테이블 이름은 알파벳 문자로 시작되어야 합니다. 테이블 이름이 알파벳이 아닌 다른 문자로 시작된 경우, 따옴표(따옴표 붙은 식별자)로 둘러쌉니다.

table_alias는 추가 설명이 포함된 이름을 테이블에 지정하고, 긴 테이블 이름을 줄이거나 또는 둘 이상의 쿼리에 동일 테이블을 포함할 수 있습니다(예: 셀프 조인).

필드 이름은 알파벳 문자로 시작되어야 합니다. 필드 이름이 알파벳이 아닌 다른 문자로 시작된 경우, 따옴표(따옴표 붙은 식별자)로 둘러쌉니다. 예를 들어, _LASTNAME이라는 필드에 대한 ExecuteSQL 문은 다음과 같습니다.

```
SELECT "_LASTNAME" from emp
```

필드 이름이 테이블 이름 또는 테이블 별칭 앞에 있을 수 있습니다. 예를 들어, 테이블 스펙을 FROM employee E라고 지정하면, LAST_NAME 필드를 E.LAST_NAME으로 참조할 수 있습니다. SELECT 문이 테이블을 테이블 자체에 연결하면 테이블 별칭이 사용되어야 합니다. 예를 들어 다음과 같습니다.

```
SELECT * FROM employee E, employee F WHERE E.manager_id =
F.employee_id
```

등호(=)에는 결과에서 일치된 행만 포함합니다.

둘 이상의 테이블을 연결하고 원형 테이블의 해당 행에 없는 모든 행을 버리려면, INNER JOIN을 사용할 수 있습니다. 예를 들어 다음과 같습니다.

```
SELECT *
   FROM Salespeople INNER JOIN Sales_Data
   ON Salespeople.Salesperson_ID = Sales_Data.Salesperson_ID
```

두 테이블을 연결하지만 첫 번째 테이블("왼쪽" 테이블)의 행을 버리지 않으려면, LEFT OUTER JOIN을 사용할 수 있습니다.

```
SELECT *
   FROM Salespeople LEFT OUTER JOIN Sales_Data
   ON Salespeople.Salesperson_ID = Sales_Data.Salesperson_ID
```

"Salespeople" 테이블의 모든 행은 연결된 테이블에 표시됩니다.

참고

- RIGHT OUTER JOIN은 현재 지원되지 않습니다.
- FULL OUTER JOIN은 현재 지원되지 않습니다.

WHERE 절

WHERE 절은 레코드를 검색하기 위해 충족해야 하는 조건을 지정합니다. WHERE 절은 다음 양식의 조건을 포함합니다.

```
WHERE expr1 rel_operator expr2
```

expr1 및 expr2는 필드 이름, 상수 값 또는 표현식일 수 있습니다.

rel_operator는 두 표현식을 링크하는 관계형 연산자입니다. 예를 들어, 다음 SELECT 문은 \20,000 이상의 수입이 있는 직원의 이름을 검색합니다.

```
SELECT last_name,first_name FROM emp WHERE salary >= 20000
```

WHERE 절은 다음과 같은 표현식을 사용할 수도 있습니다.

```
WHERE expr1 IS NULL
WHERE NOT expr2
```

메모 SELECT(계획) 목록에서 정규화된 이름을 사용하는 경우, 관련 WHERE 절에서도 정규화된 이름을 사용해야 합니다.

GROUP BY 절

GROUP BY 절은 반환된 값을 그룹화해야 하는 하나 이상의 필드 이름을 지정합니다. 이 절은 집계 값 세트를 반환하는 데 사용됩니다. 다음과 같은 형식입니다.

```
GROUP BY columns
```

GROUP BY 절의 범위는 FROM 절의 테이블 표현식입니다. 결과적으로 columns에서 지정된 열 표현식은 FROM 절에서 지정된 테이블로부터 와야 합니다. 열 표현식은 쉼표로 분리된 하나 이상의 데이터베이스 테이블의 필드 이름일 수 있습니다.

예제

다음 예제는 각 부서의 월급을 총계합니다.

```
SELECT dept_id, SUM (salary) FROM emp GROUP BY dept_id
```

명령문은 별도의 각 부서 ID에 대해 한 행을 반환합니다. 각 행은 부서 ID 및 해당 부서의 직원 월급 총계를 포함합니다.

HAVING 절

HAVING 절은 레코드 그룹에 대한 조건을 지정할 수 있습니다(예: 월급이 총 \200,000 이상인 부서만 표시). 다음과 같은 형식입니다.

```
HAVING expr1 rel_operator expr2
```

expr1 및 expr2는 필드 이름, 상수 값 또는 표현식일 수 있습니다. 이 표현식은 SELECT 절의 열 표현식과 일치하지 않아도 됩니다.

rel_operator는 두 표현식을 링크하는 관계형 연산자입니다.

예제

다음 예제는 총 월급이 \200,000보다 많은 부서만을 반환합니다.

```
SELECT dept_id, SUM (salary) FROM emp
   GROUP BY dept_id HAVING SUM (salary) > 200000
```

UNION 연산자

UNION 연산자는 둘 이상의 SELECT 문의 결과를 단일 결과로 결합합니다. 단일 결과는 SELECT 문에서 반환된 모든 레코드입니다. 기본적으로 중복 레코드는 반환되지 않습니다. 중복 레코드를 반환하려면, ALL 키워드(UNION ALL)를 사용하십시오. 포맷은 다음과 같습니다.

```
SELECT statement UNION [ALL] SELECT statement
```

UNION 연산자를 사용하면 각 SELECT 문에 대한 선택 목록에 동일한 수의 열 표현식이 있어야 하며 동일한 데이터 유형으로 동일한 순서로 지정되어야 합니다. 예를 들어 다음과 같습니다.

```
SELECT last_name, salary, hire_date FROM emp UNION SELECT name, pay,
birth_date FROM person
```

이 예제에는 동일한 수의 열 표현식이 있으며 차례대로 각 열 표현식에는 동일한 데이터 유형이 있습니다.

열 표현식의 데이터 유형이 다르기 때문에(EMP의 SALARY에는 RAISES의 LAST_NAME과 다른 데이터 유형이 있음) 다음 예제는 유효하지 않습니다. 이 예제에서 각 SELECT 문에 동일한 수의 열 표현식이 있지만 표현식은 데이터 유형별 동일 순서가 아닙니다.

```
SELECT last_name, salary FROM emp UNION SELECT salary, last_name FROM
raises
```

ORDER BY 절

ORDER BY 절은 레코드의 정렬 방법을 표시합니다. SELECT 문이 ORDER BY 절을 포함하지 않는 경우 임의의 순서로 레코드를 반환할 수도 있습니다.

포맷은 다음과 같습니다.

```
ORDER BY {sort_expression [DESC | ASC]}, ...
```

sort_expression은 필드 이름 또는 사용할 열 표현식의 위치 번호일 수 있습니다. 기본값은 오름차순(ASC) 정렬을 수행하는 것입니다.

예를 들어 last_name, first_name 순서로 정렬하려면, 다음 SELECT 문 중 하나를 사용할 수 있습니다.

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name
```

또는

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY 2,3
```

두 번째 예제는 위치 번호 2와 3을 사용하여 last_name 및 first_name을 명시적으로 지정한 이전 예제와 동일한 순서로 가져옵니다.

메모 FileMaker SQL은 유니코드 이진 정렬 순서를 사용합니다. 언어 정렬 또는 기본 언어 중립적 정렬 순서로 사용되는 FileMaker Pro 정렬 순서와는 다릅니다.

OFFSET 및 FETCH FIRST 절

OFFSET 및 FETCH FIRST 절은 결과 세트의 특정 시작점에서 시작되는 지정된 행의 범위를 반환하는 데 사용됩니다. 큰 결과 세트에서 검색된 행을 제한하는 기능을 사용하면 데이터를 "페이징"하여 효율성을 향상시킬 수 있습니다.

OFFSET 절은 데이터를 반환하기 전에 건너뛸 행 수를 표시합니다. OFFSET 절이 SELECT 문에서 사용되지 않으면 시작 행은 0입니다. FETCH FIRST 절은 반환할 행 수를 OFFSET 절에서 표시한 시작점에서부터 1 이상의 부호 없는 정수나 백분율로 지정합니다. OFFSET 및 FETCH FIRST 모두 SELECT 문에서 사용되는 경우, OFFSET 절이 먼저 와야 합니다.

OFFSET 및 FETCH FIRST 절은 하위 쿼리에서 지원되지 않습니다.

OFFSET 포맷

OFFSET 포맷은 다음과 같습니다.

```
OFFSET n {ROWS | ROW} ]
```

n은 부호 없는 정수입니다. n이 결과 세트에서 반환된 행 수보다 큰 경우, 반환되는 것이 없어 오류 메시지가 나타나지 않습니다.

ROWS는 ROW와 동일합니다.

FETCH FIRST 포맷

FETCH FIRST 포맷은 다음과 같습니다.

```
FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
```

n은 반환될 행 수입니다. n이 생략된 경우 기본 값은 1입니다 .

PERCENT가 뒤에 오지 않으면 n은 1 이상의 부호 없는 정수입니다. n 다음에 PERCENT가 오면 , 값은 양의 분수 값 또는 부호 없는 정수입니다.

ROWS는 ROW와 동일합니다.

WITH TIES는 ORDER BY 절과 함께 사용되어야 합니다.

WITH TIES는 피어 행 때문에 FETCH에서 지정한 개수 값보다 더 많은 행을 반환할 수 있으며, ORDER BY 절을 기반으로 하는 분명하지 않은 이 행도 반환됩니다.

예제

예를 들어, last_name, first_name 순서로 정렬된 결과 세트의 26번째 행에서 정보를 반환하려면, 다음 SELECT 문을 사용하십시오.

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS
```

10개의 행만 반환하도록 지정하려면 다음을 입력하십시오.

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS FETCH FIRST 10 ROWS ONLY
```

10개의 행 및 피어 행(ORDER BY 절을 기반으로 한 분명하지 않은 행)을 반환하려면 다음을 입력하십시오.

```
SELECT emp_id, last_name, first_name FROM emp ORDER BY last_name,
first_name OFFSET 25 ROWS FETCH FIRST 10 ROWS WITH TIES
```

FOR UPDATE 절

FOR UPDATE 절은 SQL 커서를 통해 현재 위치 업데이트 또는 현재 위치 삭제를 위한 레코드를 잠급니다. 포맷은 다음과 같습니다.

```
FOR UPDATE [OF column_expressions]
```

column_expressions는 쉼표로 분리된 업데이트할 데이터베이스의 필드 이름 목록입니다. column_expressions는 옵션이며 무시됩니다.

예제

다음 예제는 SALARY 필드의 값이 \20,000보다 많은 직원 데이터베이스의 모든 레코드를 반환합니다. 각 레코드를 가져오면 잠겨 있습니다. 레코드가 업데이트되거나 삭제되면 변경사항을 커밋할 때까지 잠금이 적용됩니다. 그렇지 않으면 다음 레코드를 가져올 때 잠금이 해제됩니다.

```
SELECT * FROM emp WHERE salary > 20000
   FOR UPDATE OF last_name, first_name, salary
```

추가 예제:

사용	샘플 SQL
텍스트 상수	`SELECT 'CatDog' FROM Salespeople`
숫자 상수	`SELECT 999 FROM Salespeople`
날짜 상수	`SELECT DATE '2016-06-05' FROM Salespeople`
시간 상수	`SELECT TIME '02:49:03' FROM Salespeople`
타임스탬프 상수	`SELECT TIMESTAMP '2016-06-05 02:49:03' FROM Salespeople`
텍스트 열	`SELECT Company_Name FROM Sales_Data` `SELECT DISTINCT Company_Name FROM Sales_Data`
숫자 열	`SELECT Amount FROM Sales_Data` `SELECT DISTINCT Amount FROM Sales_Data`
날짜 열	`SELECT Date_Sold FROM Sales_Data` `SELECT DISTINCT Date_Sold FROM Sales_Data`
시간 열	`SELECT Time_Sold FROM Sales_Data` `SELECT DISTINCT Time_Sold FROM Sales_Data`
타임스탬프 열	`SELECT Timestamp_Sold FROM Sales_Data` `SELECT DISTINCT Timestamp_Sold FROM Sales_Data`
BLOB[a] 열	`SELECT Company_Brochures FROM Sales_Data` `SELECT GETAS(Company_Logo, 'JPEG') FROM Sales_Data`
Wildcard *	`SELECT * FROM Salespeople` `SELECT DISTINCT * FROM Salespeople`

a. BLOB 는 FileMaker 데이터베이스 파일 컨테이너 필드입니다 .

예제에서 참고

`column`은 FileMaker 데이터베이스 파일의 필드에 대한 참조입니다 (필드에는 여러 고유 값이 포함되어 있을 수 있습니다.)

별표(*) 와일드카드 문자는 "모든 것"의 약칭입니다. 예를 들어, `SELECT * FROM Salespeople`의 결과는 `Salespeople` 테이블의 모든 열입니다. 예를 들어, `SELECT DISTINCT * FROM Salespeople`의 결과는 `Salespeople` 테이블(중복 없음)의 고유한 모든 행입니다.

- FileMaker는 빈 문자열에 데이터를 저장하지 않으므로 다음 쿼리는 항상 레코드를 반환하지 않습니다.

```
SELECT * FROM test WHERE c =' '
SELECT * FROM test WHERE c <>' '
```

- 이진 데이터를 포함한 `SELECT`를 사용하는 경우, `GetAs()` 함수를 사용하여 반환할 스트림을 지정해야 합니다. 추가 정보는 다음 "컨테이너 필드의 콘텐츠 검색하기: CAST() 함수 및 GetAs() 함수," 부분을 참조하십시오.

컨테이너 필드의 콘텐츠 검색하기: CAST() 함수 및 GetAs() 함수

이진 데이터, 파일 참조 정보 또는 특정 파일 유형의 데이터를 컨테이너 필드에서 검색할 수 있습니다.

파일 데이터 또는 JPEG 이진 데이터가 존재하는 경우, `GetAS(field name, 'JPEG')`를 포함한 `SELECT` 문은 이진 양식의 데이터를 검색합니다. 그렇지 않은 경우 필드 이름이 포함된 `SELECT` 문은 `NULL`을 반환합니다.

파일, 그림 또는 QuickTime 동영상에 대한 파일 경로와 같이 컨테이너 필드에서 파일 참조 정보를 검색하려면 `CAST()` 함수와 `SELECT` 문을 사용하십시오. 예를 들어 다음과 같습니다.

```
SELECT CAST(Company_Brochures AS VARCHAR) FROM Sales_Data
```

이 예제에서는 다음과 같습니다.

- FileMaker Pro를 사용하여 파일을 컨테이너 필드에 삽입했지만 파일에 대한 참조만 저장한 경우, `SQL_VARCHAR`을 입력하면 `SELECT` 문은 파일 참조 정보를 검색합니다.
- FileMaker Pro를 사용하여 파일의 콘텐츠를 컨테이너 필드에 삽입한 경우, `SELECT` 문은 파일 이름을 검색합니다.
- 다른 응용 프로그램의 컨테이너 필드로 파일을 가져오면 `SELECT` 문은 '?'를 표시합니다 (파일은 FileMaker Pro에 **Untitled.dat**으로 표시됩니다).

컨테이너 필드에서 데이터를 검색하려면 `GetAs()` 함수를 사용하십시오. `DEFAULT` 옵션을 사용하거나 파일 유형을 지정할 수 있습니다. `DEFAULT` 옵션은 스트림 유형을 명시적으로 정의하지 않아도 컨테이너의 마스터 스트림을 검색합니다.

```
SELECT GetAs(Company_Brochures, DEFAULT) FROM Sales_Data
```

컨테이너에서 개별 스트림 유형을 검색하려면 데이터를 FileMaker Pro의 컨테이너 필드로 삽입하는 방법을 기반으로 하는 파일의 유형으로 `GetAs()` 함수를 사용하십시오. 예를 들어 다음과 같습니다.

- **삽입** > **파일** 명령을 사용하여 데이터를 삽입했던 경우, `GetAs()` 함수에 `'FILE'`을 지정하십시오. 예를 들어 다음과 같습니다.

```
SELECT GetAs(Company_Brochures, 'FILE') FROM Sales_Data
```

- **삽입** > **사운드** 명령(표준 사운드 — MAC OS X 원시 포맷)을 사용하여 데이터를 삽입했던 경우, `GetAs()` 함수에 `'snd'`를 지정하십시오. 예를 들어 다음과 같습니다.

```
SELECT GetAs(Company_Meeting, 'snd ') FROM Company_Newsletter
```

- **삽입 > 그림** 명령을 사용하여 데이터를 삽입했던 경우, 드래그 앤 드롭하거나 클립보드에서 붙이고, 다음 테이블에 나열된 파일 유형 중 하나를 지정하십시오. 예를 들어 다음과 같습니다.

```
SELECT GetAs(Company_Logo, 'JPEG') FROM Company_Icons
```

파일 유형	설명	파일 유형	설명
'GIFf'	GIF(Graphics Interchange Format)	'PNTG'	MacPaint
'JPEG'	사진 이미지	'.SGI'	일반 비트맵 포맷
'JP2 '	JPEG 2000	'TIFF'	디지털 이미지용 Raster 파일 포맷
'PDF '	PDF(Portable Document Format)	'TPIC'	Targa
'PNGf'	비트맵 이미지 포맷	'8BPS'	PhotoShop(PSD)

DELETE 문

DELETE 문을 사용하여 데이터베이스 테이블에서 레코드를 삭제합니다. DELETE 문의 포맷은 다음과 같습니다.

```
DELETE FROM table_name [ WHERE { conditions } ]
```

메모 WHERE 절은 삭제할 레코드를 결정합니다. WHERE 키워드를 포함하지 않는 경우, 테이블의 모든 레코드가 삭제됩니다(하지만 테이블은 그대로 남음).

예제

직원 테이블에서의 DELETE 문 예제는 다음과 같습니다.

```
DELETE FROM emp WHERE emp_id = 'E10001'
```

각 DELETE 문은 WHERE 절에서 조건을 충족하는 모든 레코드를 제거합니다. 이 경우 직원 ID가 E10001인 모든 레코드가 삭제됩니다. 직원 ID가 직원 테이블에서 고유하기 때문에 하나의 레코드만 삭제됩니다.

INSERT 문

INSERT 문을 사용하여 데이터베이스 테이블에 레코드를 생성합니다. 다음 중 하나를 지정할 수 있습니다.

- 새로운 레코드로 삽입할 값의 목록
- 새로운 레코드 세트로 삽입할 다른 테이블에서 복사하는 SELECT 문

INSERT 문의 포맷은 다음과 같습니다.

```
INSERT INTO table_name [(column_name, ...)] VALUES (expr, ...)
```

column_name은 VALUES 절에서 지정된 값의 열 이름과 순서를 제공하는 열 이름의 옵션 목록입니다. column_name을 생략하는 경우, 값 표현식(expr)은 테이블에서 정의된 모든 열의 값을 제공해야 하며 테이블에 정의된 열과 동일한 순서여야 합니다. column_name은 필드 반복도 지정할 수 있습니다(예: lastDates[4]).

expr은 새로운 레코드의 열에 대한 값을 제공하는 표현식 목록입니다. 보통 표현식은 열에 대해 상수 값입니다(하지만 하위 쿼리일 수 있음). 작은 따옴표(') 쌍 안에 문자열 값을 둘러싸야 합니다. 작은 따옴표로 둘러싼 문자열 값에 작은 따옴표를 포함하려면, 두 개의 작은 따옴표를 같이 사용하십시오(예: 'Don''t').

하위 쿼리는 괄호로 둘러싸야 합니다.

다음 예제는 표현식 목록을 삽입합니다.

```
INSERT INTO emp (last_name, first_name, emp_id, salary, hire_date)
   VALUES ('Smith', 'John', 'E22345', 27500, DATE '2016-06-05')
```

각 INSERT 문은 하나의 레코드를 데이터베이스 테이블에 추가합니다. 이 경우 레코드는 직원 데이터베이스 테이블, EMP에 추가됩니다. 값은 5개의 열에 대해 지정됩니다. 테이블의 나머지 열은 빈 값을 지정하며 이는 Null을 의미합니다.

메모 매개변수가 있는 명령문을 준비하고 사용자의 응용 프로그램에서 데이터를 스트림하지 않는 경우 컨테이너 필드에서 텍스트만 삽입할 수 있습니다. 이진 데이터를 사용하려면 간단하게 작은 따옴표로 둘러싸서 파일 이름을 지정하거나 PutAs() 함수를 사용할 수 있습니다. 파일 이름을 지정하면 파일 유형은 파일 확장자로 추정됩니다.

```
INSERT INTO table_name (container_name) VALUES(? AS 'filename.file
extension')
```

지원되지 않는 파일 유형은 유형 FILE로 삽입됩니다.

PutAs() 함수 사용 시 유형을 PutAs(col, 'type')으로 지정하십시오. 여기서 유형 값은 16 페이지의 "컨테이너 필드의 콘텐츠 검색하기: CAST() 함수 및 GetAs() 함수".에서 설명한 대로 지원되는 파일 유형입니다.

SELECT 문은 열 이름 목록에서 지정된 각 column_name 값에 대한 값을 반환하는 쿼리입니다. 값 표현식 목록 대신 SELECT 문을 사용하면 하나의 테이블에서 행 세트를 선택하고 단일 INSERT 문을 사용하여 다른 테이블로 삽입할 수 있습니다.

다음은 SELECT 문을 사용하는 INSERT 문의 예제입니다.

```
INSERT INTO emp1 (first_name, last_name, emp_id, dept, salary)
   SELECT first_name, last_name, emp_id, dept, salary from emp
   WHERE dept = 'D050'
```

이 유형의 INSERT 문에서 삽입할 열의 수는 SELECT 문의 열 수와 일치해야 합니다. 다른 유형의 INSERT 문에서 값 표현식 목록의 경우과 같이 삽입할 열 목록은 SELECT 문의 열에 해당되어야 합니다. 예를 들어, 삽입된 첫 번째 열은 선택한 첫 번째 열에 해당되며, 두 번째 열은 두 번째에 해당됩니다.

해당 열의 크기와 데이터 유형은 호환되어야 합니다. SELECT 목록의 각 열에는 INSERT 목록의 해당 열의 일반적인 INSERT/UPDATE에서 ODBC 또는 JDBC 클라이언트 드라이버가 승인하는 데이터 유형이 있어야 합니다. SELECT 목록 열의 값 크기가 해당 INSERT 목록 열의 크기보다 크면 값이 잘립니다.

SELECT 문은 값을 삽입하기 전에 평가됩니다.

UPDATE 문

UPDATE 문을 사용하여 데이터베이스 테이블에서 레코드를 변경합니다. UPDATE 문의 포맷은 다음과 같습니다.

```
UPDATE table_name SET column_name = expr, ... [ WHERE { conditions } ]
```

column_name은 값을 변경할 열의 이름입니다. 몇몇 열은 하나의 명령문에서 변경될 수 있습니다.

expr은 열의 새로운 값입니다.

보통 표현식은 열에 대해 상수 값입니다(하지만 하위 쿼리일 수 있음). 작은 따옴표(') 쌍 안에 문자열 값을 둘러싸야 합니다. 작은 따옴표로 둘러싼 문자열 값에 작은 따옴표를 포함하려면, 두 개의 작은 따옴표를 같이 사용하십시오(예: 'Don''t').

하위 쿼리는 괄호로 둘러싸야 합니다.

WHERE 절은 유효한 절입니다. 업데이트된 레코드를 결정합니다.

예제

직원 테이블에서 UPDATE 문 예제는 다음과 같습니다.

```
UPDATE emp SET salary=32000, exempt=1 WHERE emp_id = 'E10001'
```

UPDATE 문은 WHERE 절에서 조건을 충족하는 모든 레코드를 변경합니다. 이 경우 월급 및 예외 상태는 직원 ID가 E10001인 모든 직원에 대해 변경됩니다. 직원 ID가 직원 테이블에서 고유하기 때문에 하나의 레코드만 업데이트됩니다.

다음은 하위 쿼리를 사용한 예제입니다.

```
UPDATE emp SET salary = (SELECT avg(salary) from emp) WHERE emp_id =
'E10001'
```

이 경우 월급은 직원 ID가 E10001인 직원에 대해 회사 평균 월급으로 변경됩니다.

메모 매개변수가 있는 명령문을 준비하고 사용자의 응용 프로그램에서 데이터를 스트림하지 않는 경우 컨테이너 필드에서 텍스트만 업데이트할 수 있습니다. 이진 데이터를 사용하려면 간단하게 작은 따옴표로 둘러싸서 파일 이름을 지정하거나 PutAs() 함수를 사용할 수 있습니다. 파일 이름을 지정하면 파일 유형은 파일 확장자로 추정됩니다.

```
UPDATE table_name SET (container_name) = ? AS 'filename.file
extension'
```

지원되지 않는 파일 유형은 유형 FILE로 삽입됩니다.

PutAs() 함수 사용 시 유형을 PutAs(col, 'type')으로 지정하십시오. 여기서 유형 값은 16 페이지의 "컨테이너 필드의 콘텐츠 검색하기: CAST() 함수 및 GetAs() 함수"에서 설명한 대로 지원되는 파일 유형입니다.

CREATE TABLE 문

CREATE TABLE 문을 사용하여 데이터베이스 파일에 테이블을 생성합니다. CREATE TABLE 문의 포맷은 다음과 같습니다.

```
CREATE TABLE table_name ( table_element_list [,
table_element_list...] )
```

명령문 내에서 각 열의 이름과 데이터 유형을 지정합니다.

- table_name은 테이블의 이름입니다. table_name에는 100개의 문자 제한이 있습니다. 이름이 동일한 테이블은 정의되지 않아야 합니다. 테이블 이름은 알파벳 문자로 시작되어야 합니다. 테이블 이름이 알파벳이 아닌 다른 문자로 시작된 경우, 따옴표(따옴표 붙은 식별자)로 둘러쌉니다.
- table_element_list의 포맷은 다음과 같습니다.

```
field_name field_type [[repetitions]]
[DEFAULT expr] [UNIQUE | NOT NULL | PRIMARY KEY | GLOBAL]
[EXTERNAL relative_path_string [SECURE | OPEN calc_path_string]]
```

  - field_name은 필드의 이름입니다. 필드 이름은 고유값이어야 합니다. 필드 이름은 알파벳 문자로 시작되어야 합니다. 필드 이름이 알파벳이 아닌 다른 문자로 시작된 경우, 따옴표(따옴표 붙은 식별자)로 둘러쌉니다. 예를 들어, _LASTNAME이라는 필드에 대한 CREATE TABLE 문은 다음과 같습니다.

```
CREATE TABLE "_EMPLOYEE" (ID INT PRIMARY KEY, "_FIRSTNAME"
VARCHAR(20), "_LASTNAME" VARCHAR(20))
```

  - CREATE TABLE 문 repetitions의 경우 필드 유형 뒤에 대괄호 안에 1~32000 사이의 숫자를 사용하여 필드 반복을 지정합니다. 예를 들어 다음과 같습니다.

```
EMPLOYEE_ID INT[4]
LASTNAME VARCHAR(20)[4]
```

  - field_type은 다음 중 하나일 수 있습니다. NUMERIC, DECIMAL, INT, DATE, TIME, TIMESTAMP, VARCHAR, CHARACTER VARYING, BLOB, VARBINARY, LONGVARBINARY 또는 BINARY VARYING. NUMERIC 및 DECIMAL의 경우, 자릿수와 크기를 지정할 수 있습니다. 예를 들어 다음과 같습니다. DECIMAL(10,0). TIME 및 TIMESTAMP의 경우, 자릿수를 지정할 수 있습니다. 예를 들어 다음과 같습니다. TIMESTAMP(6). VARCHAR 및 CHARACTER VARYING의 경우, 문자열 길이를 지정할 수 있습니다. 예를 들어 다음과 같습니다. VARCHAR(255).
  - DEFAULT 키워드를 사용하면 열에 대해 기본값을 설정할 수 있습니다. expr의 경우, 상수나 표현식을 사용할 수 있습니다. 허용 가능한 표현식은 USER, USERNAME, CURRENT_USER, CURRENT_DATE, CURDATE, CURRENT_TIME, CURTIME, CURRENT_TIMESTAMP, CURTIMESTAMP 및 NULL입니다.
  - 열을 UNIQUE로 정의하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 자동으로 **고유** 유효성 옵션을 선택합니다.
  - 열을 NOT NULL로 정의하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 자동으로 **비어 있지 않음** 유효성 옵션을 선택합니다. 필드는 FileMaker Pro의 데이터베이스 관리 대화상자의 **필드** 탭에서 **필수 값**으로 플래그 지정됩니다.

- 열을 컨테이너 필드로 정의하려면 BLOB, VARBINARY 또는 BINARY VARYING을 field_type으로 사용하십시오.
- 외부적으로 데이터를 저장하는 컨테이너 필드로 열을 정의하려면 EXTERNAL 키워드를 사용하십시오. relative_path_string은 FileMaker 데이터베이스의 위치와 상대적으로 외부적으로 데이터를 저장하는 폴더를 정의합니다. 이 경로는 FileMaker Pro 관리 컨테이너 대화상자의 기본 디렉토리로 지정되어야 합니다. 보안 저장 장치는 SECURE 또는 공개 저장 장치는 OPEN을 지정해야 합니다. 공개 저장 장치를 사용하는 경우 calc_path_string은 컨테이너 대상체를 저장할 relative_path_string 폴더 내의 폴더입니다. 경로는 폴더 이름에 슬래시(/)를 사용해야 합니다.

예제

사용	샘플 SQL
텍스트 열	`CREATE TABLE T1 (C1 VARCHAR, C2 VARCHAR (50), C3 VARCHAR (1001), C4 VARCHAR (500276))`
텍스트 열, NOT NULL	`CREATE TABLE T1NN (C1 VARCHAR NOT NULL, C2 VARCHAR (50) NOT NULL, C3 VARCHAR (1001) NOT NULL, C4 VARCHAR (500276) NOT NULL)`
숫자 열	`CREATE TABLE T2 (C1 DECIMAL, C2 DECIMAL (10,0), C3 DECIMAL (7539,2), C4 DECIMAL (497925,301))`
날짜 열	`CREATE TABLE T3 (C1 DATE, C2 DATE, C3 DATE, C4 DATE)`
시간 열	`CREATE TABLE T4 (C1 TIME, C2 TIME, C3 TIME, C4 TIME)`
타임스탬프 열	`CREATE TABLE T5 (C1 TIMESTAMP, C2 TIMESTAMP, C3 TIMESTAMP, C4 TIMESTAMP)`
컨테이너 필드 열	`CREATE TABLE T6 (C1 BLOB, C2 BLOB, C3 BLOB, C4 BLOB)`
외부 저장 장치 컨테이너 필드 열	`CREATE TABLE T7 (C1 BLOB EXTERNAL 'Files/MyDatabase/' SECURE)` `CREATE TABLE T8 (C1 BLOB EXTERNAL 'Files/MyDatabase/' OPEN 'Objects')`

TRUNCATE TABLE 문

TRUNCATE TABLE 문을 사용하면 지정된 테이블에서 모든 레코드를 빠르게 삭제하여 모든 데이터 테이블을 비웁니다.

```
TRUNCATE TABLE table_name
```

WHERE 절을 TRUNCATE TABLE 문으로 지정할 수 없습니다. TRUNCATE TABLE 문은 모든 레코드를 삭제합니다.

table_name에서 지정된 테이블의 레코드만 삭제합니다. 관련 테이블의 레코드에는 적용되지 않습니다.

TRUNCATE TABLE 문은 레코드 데이터를 삭제하기 위해 테이블에서 모든 레코드를 잠글 수 있어야 합니다. 다른 사용자가 테이블에서 레코드를 잠근 경우, FileMaker는 오류 코드 301("다른 사용자가 레코드를 사용 중입니다")을 반환합니다.

ALTER TABLE 문

ALTER TABLE 문을 사용하여 데이터베이스 파일에서 기존 테이블의 구조를 변경합니다. 각 명령문에서 하나의 열만 수정할 수 있습니다. ALTER TABLE 문의 포맷은 다음과 같습니다.

```
ALTER TABLE table_name ADD [COLUMN] column_definition

ALTER TABLE table_name DROP [COLUMN] unqualified_column_name

ALTER TABLE table_name ALTER [COLUMN] column_definition SET DEFAULT
expr

ALTER TABLE table_name ALTER [COLUMN] column_definition DROP DEFAULT
```

ALTER TABLE 문을 사용하기 전에 테이블의 구조와 수정할 방법을 알아야 합니다.

예제

작업	샘플 SQL
열 추가	ALTER TABLE Salespeople ADD C1 VARCHAR
열 제거	ALTER TABLE Salespeople DROP C1
열의 기본값 설정	ALTER TABLE Salespeople ALTER Company SET DEFAULT 'FileMaker'
열의 기본값 제거	ALTER TABLE Salespeople ALTER Company DROP DEFAULT

메모 SET DEFAULT 및 DROP DEFAULT는 테이블의 기존 행에 영향을 주지 않지만 다음에 테이블에 추가된 행의 기본값을 변경합니다.

CREATE INDEX 문

CREATE INDEX 문을 사용하여 데이터베이스 파일에서 빠르게 검색합니다. CREATE INDEX 문의 포맷은 다음과 같습니다.

```
CREATE INDEX ON table_name.column_name
CREATE INDEX ON table_name (column_name)
```

CREATE INDEX는 단일 열에 대해 지원됩니다(다중 열 인덱스가 지원되지 않음). 컨테이너 필드 유형, 요약 필드, 전역 저장 장치 옵션이 있는 필드 또는 FileMaker 데이터베이스 파일에 저장되지 않은 계산 필드에 해당하는 열에서 인덱스가 허용되지 않습니다.

텍스트 열에 대해 인덱스를 생성하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 **인덱싱**에서 **최소**의 저장 장치 옵션을 자동으로 선택합니다. 텍스트가 아닌 열(또는 일본어 텍스트로 포맷된 열)에 대해 인덱스를 생성하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 **인덱싱**에서 **모두**의 저장 장치 옵션을 자동으로 선택합니다.

모든 열에 대해 인덱스를 생성하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 **인덱싱**에서 **필요한 경우 인덱스 자동 생성**의 저장 장치 옵션을 자동으로 선택합니다.

필요한 경우 FileMaker는 자동으로 인덱스를 생성합니다. CREATE INDEX를 사용하면 명령할 때가 아니라 즉시 인덱스가 빌드됩니다.

예제

```
CREATE INDEX ON Salespeople.Salesperson_ID
```

DROP INDEX 문

DROP INDEX 문을 사용하여 데이터베이스 파일에서 인덱스를 제거합니다. DROP INDEX 문의 포맷은 다음과 같습니다.

```
DROP INDEX ON table_name.column_name
DROP INDEX ON table_name (column_name)
```

사용자의 데이터베이스 파일이 너무 크거나, 또는 쿼리에서 필드를 자주 사용하지 않으면 인덱스를 제거하십시오.

사용자의 쿼리 성능이 저하되고 아주 큰 FileMaker 데이터베이스 파일을 인덱스된 여러 텍스트 필드로 작업하는 경우, 일부 필드에서 인덱스를 드롭해 보십시오. 또한, SELECT 문에서 거의 사용하지 않는 필드에서 인덱스를 드롭해 보십시오.

모든 열에 대해 인덱스를 드롭하면 FileMaker 데이터베이스 파일의 해당 필드에 대해 **인덱싱**에서 **없음** 및 **필요한 경우 인덱스 자동 생성**의 저장 장치 옵션을 자동으로 선택합니다.

PREVENT INDEX CREATION 속성이 지원되지 않습니다.

예제

```
DROP INDEX ON Salespeople.Salesperson_ID
```

SQL 표현식

SELECT 문의 WHERE, HAVING 및 ORDER BY 절에서 표현식을 사용하여 상세하고 복잡한 데이터베이스 쿼리를 만드십시오. 유효한 표현식 요소는 다음과 같습니다.

- 필드 이름
- 상수
- 지수/과학적 표기법
- 숫자 연산자
- 문자 연산자
- 날짜 연산자
- 관계형 연산자
- 논리 연산자
- 함수

필드 이름

대부분의 일반 표현식은 calc 또는 Sales_Data.Invoice_ID와 같은 단순 필드 이름입니다.

상수

상수는 변경되지 않는 값입니다. 예를 들어, 표현식 `PRICE * 1.05`에서 값 1.05는 상수입니다. 또는 30의 값을 상수 `Number_Of_Days_In_June`에 지정할 수 있습니다.

작은 따옴표(') 쌍 안에 문자 상수를 둘러싸야 합니다. 작은 따옴표로 둘러싼 문자 상수에 작은 따옴표를 포함하려면, 두 개의 작은 따옴표를 같이 사용하십시오(예: `'Don''t'`).

ODBC 및 JDBC 응용 프로그램의 경우, FileMaker는 대괄호({}) 안의 ODBC/JDBC 포맷 날짜, 시간 및 타임스탬프 상수를 허용합니다. 예제:

- `{D '2016-06-05'}`
- `{T '14:35:10'}`
- `{TS '2016-06-05 14:35:10'}`

FileMaker에서는 형식 지정자(`D`, `T`, `TS`)가 대문자나 소문자일 수 있습니다. 형식 지정자 뒤에 공백을 사용하거나 공백을 생략할 수 있습니다.

FileMaker는 중괄호가 없는 SQL-92 구문 ISO 날짜 및 시간 포맷도 허용합니다.

- `DATE 'YYYY-MM-DD'`
- `TIME 'HH:MM:SS'`
- `TIMESTAMP 'YYYY-MM-DD HH:MM:SS'`

FileMaker Pro ExecuteSQL 함수는 중괄호가 없는 SQL-92 구문 ISO 날짜 및 시간 포맷만 승인합니다.

상수	허용 가능한 구문 (예)
텍스트	`'Paris'`
숫자	`1.05`

날짜	`DATE '2016-06-05'`
	`{ D '2016-06-05' }`
	`{06/05/2016}`
	`{06/05/16}`
	참고 두 자리의 연도 구문은 ODBC/JDBC 포맷이나 SQL-92 포맷에서 지원되지 않습니다.

시간	`TIME '14:35:10'`
	`{ T '14:35:10' }`
	`{14:35:10}`

타임스탬프	`TIMESTAMP '2016-06-05 14:35:10'`
	`{ TS '2016-06-05 14:35:10'}`
	`{06/05/2016 14:35:10}`
	`{06/05/16 14:35:10}`
	데이터 유형 제한:을 확인하십시오. **4자리 연도 날짜**는 이 2자리 연도 구문을 사용하는 필드에 대한 FileMaker 데이터베이스에서 유효한 옵션으로 선택되지 않습니다.
	참고 두 자리의 연도 구문은 ODBC/JDBC 포맷이나 SQL-92 포맷에서 지원되지 않습니다.

날짜와 시간 값을 입력하면 데이터베이스 파일 로케일의 포맷을 일치시키십시오. 예를 들어, 데이터베이스가 이탈리아어 시스템에서 생성된 경우 이탈리아어 날짜와 시간 포맷을 사용하십시오.

지수/과학적 표기법

숫자는 과학적 표기법을 사용하여 표현될 수 있습니다.

예제

```
SELECT column1 / 3.4E+7 FROM table1 WHERE calc < 3.4E-6 * column2
```

숫자 연산자

숫자 표현식에 다음 연산자를 포함할 수 있습니다. +, -, *, / 및 ^ 또는 ** (지수).

숫자 표현식 앞에 단항 더하기(+) 또는 빼기(-)가 있을 수 있습니다.

문자 연산자

문자를 연결할 수 있습니다.

예제

다음 예제에서 `last_name`은 `'JONES '`이며 first_name은 `'ROBERT '`: 입니다.

연산자	연결	예제	결과
+	후행 공백 유지	first_name + last_name	'ROBERT JONES '
-	후행 공백을 끝으로 이동	first_name - last_name	'ROBERTJONES '

날짜 연산자

날짜를 수정할 수 있습니다.

예제

다음 예제에서 `hire_date`는 `DATE '2016-01-30'`입니다.

연산자	적용 날짜	예제	결과
+	날짜에 일 수 더하기	`hire_date` + 5	DATE '2016-02-04'
-	두 날짜 사이의 일 수 찾기	`hire_date - DATE '2016-01-01'`	`29`
	날짜에서 일 수 빼기	`hire_date - 10`	DATE '2016-01-20'

추가 예제:

```
SELECT Date_Sold, Date_Sold + 30 AS agg FROM Sales_Data
SELECT Date_Sold, Date_Sold - 30 AS agg FROM Sales_Data
```

관계형 연산자

연산자	의미
=	다음과 같음
<>	다음과 같지 않음
>	다음보다 큼
>=	다음보다 크거나 같음
<	보다 작음
<=	다음보다 작거나 같음
LIKE	패턴 일치
NOT LIKE	패턴 일치하지 않음
IS NULL	Null과 같지 않음
IS NOT NULL	Null과 같지 않음
BETWEEN	하한과 상한 사이의 값 범위
IN	지정된 값 세트 멤버 또는 하위 쿼리 멤버
NOT IN	지정된 값 세트 멤버 또는 하위 쿼리 멤버가 아님
EXISTS	하위 쿼리가 하나 이상의 레코드를 반환한 경우 'True'
ANY	값을 하위 쿼리(연산자 앞에 =, <>, >, >=, < 또는 <=가 표시)에서 반환한 각 값과 비교, =는 동일함
ALL	값을 하위 쿼리(연산자 앞에 =, <>, >, >=, < 또는 <=가 표시)에서 반환한 각 값과 비교

예제

```
SELECT Sales_Data.Invoice_ID FROM Sales_Data
   WHERE Sales_Data.Salesperson_ID = 'SP-1'

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Invoice_ID
<> 125

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount > 3000

SELECT Sales_Data.Time_Sold FROM Sales_Data
   WHERE Sales_Data.Time_Sold < '12:00:00'

SELECT Sales_Data.Company_Name FROM Sales_Data
   WHERE Sales_Data.Company_Name LIKE '%University'

SELECT Sales_Data.Company_Name FROM Sales_Data
   WHERE Sales_Data.Company_Name NOT LIKE '%University'

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount IS NULL

SELECT Sales_Data.Amount FROM Sales_Data WHERE Sales_Data.Amount IS NOT
NULL
```

```
SELECT Sales_Data.Invoice_ID FROM Sales_Data
   WHERE Sales_Data.Invoice_ID BETWEEN 1 AND 10

SELECT COUNT(Sales_Data.Invoice_ID) AS agg
   FROM Sales_Data WHERE Sales_Data.INVOICE_ID IN (50,250,100)

SELECT COUNT(Sales_Data.Invoice_ID) AS agg
   FROM Sales_Data WHERE Sales_Data.INVOICE_ID NOT IN (50,250,100)

SELECT COUNT(Sales_Data.Invoice_ID) AS agg FROM Sales_Data
   WHERE Sales_Data.INVOICE_ID NOT IN (SELECT Sales_Data.Invoice_ID
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID = 'SP-4')

SELECT *
   FROM Sales_Data WHERE EXISTS (SELECT Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID IS NOT NULL)

SELECT *
   FROM Sales_Data WHERE Sales_Data.Amount = ANY (SELECT
Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID = 'SP-1')

SELECT *
   FROM Sales_Data WHERE Sales_Data.Amount = ALL (SELECT
Sales_Data.Amount
   FROM Sales_Data WHERE Sales_Data.Salesperson_ID IS NULL)
```

논리 연산자

둘 이상의 조건을 결합할 수 있습니다. 조건은 AND 또는 OR로 연결되어야 합니다.

```
salary = 40000 AND exempt = 1
```

논리 NOT 연산자는 다음과 같이 의미를 반전하는 데 사용됩니다.

```
NOT (salary = 40000 AND exempt = 1)
```

예제

```
SELECT * FROM Sales_Data WHERE Sales_Data.Company_Name
   NOT LIKE '%University' AND Sales_Data.Amount > 3000

SELECT * FROM Sales_Data WHERE (Sales_Data.Company_Name
   LIKE '%University' OR Sales_Data.Amount > 3000)
   AND Sales_Data.Salesperson_ID = 'SP-1'
```

연산자 우선 순위

표현식이 보다 복잡해지면 표현식 평가 순서도 중요합니다. 이 표는 연산자가 평가되는 순서를 표시합니다. 첫 번째 행의 연산자가 먼저 평가되는 식입니다. 동일한 행의 연산자는 표현식에서 왼쪽에서 오른쪽으로 평가됩니다.

우선 순위	연산자
1	Unary '-', Unary '+'
2	^, **
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	AND
8	OR

다음 예제는 우선 순위의 중요성을 표시합니다.

```
WHERE salary > 40000 OR hire_date > (DATE '2008-01-30') AND dept =
'D101'
```

AND가 먼저 평가되기 때문에 이 쿼리는 2008년 1월 30일 후에 고용된 D101 부서의 직원 및 부서나 입사일에 상관 없이 수입이 \40,000보다 많은 모든 직원을 검색합니다.

강제로 해당 절을 다른 순서로 평가하려면, 괄호를 사용하여 먼저 평가할 조건을 둘러쌉니다. 예를 들어 다음과 같습니다.

```
WHERE (salary > 40000 OR hire_date > DATE '2008-01-30') AND dept =
'D101'
```

수입이 \40,000보다 많거나 2008년 1월 30일 후에 고용된 D101 부서의 직원을 검색합니다.

SQL 함수

FileMaker SQL은 표현식에서 사용할 수 있는 여러 함수를 지원합니다. 일부 함수는 문자열을 반환하고, 일부는 숫자를 반환하고, 일부는 날짜를 반환하며, 일부는 함수 인수로 충족되는 조건에 따른 값을 반환합니다.

집계 함수

집계 함수는 레코드 세트로부터 단일 값을 반환합니다. 집계 함수를 SELECT 문의 일부로 필드 이름(예: AVG(SALARY))과 함께 사용하거나 열 표현식(예: AVG(SALARY * 1.07))과 결합하여 사용할 수 있습니다.

DISTINCT 연산자와 함께 열 표현식 앞에 지정하여 중복 값을 제거할 수 있습니다. 예를 들어 다음과 같습니다.

```
COUNT (DISTINCT last_name)
```

이 예제에서 고유한 마지막 이름 값만 계산됩니다.

집계 함수	반환 결과
SUM	숫자 필드 표현식의 값 총계. 예를 들어, SUM(SALARY) 는 모든 월급 필드 값의 총계를 반환합니다.
AVG	숫자 필드 표현식의 값 평균. 예를 들어, AVG(SALARY) 는 모든 월급 필드 값의 평균을 반환합니다.
COUNT	모든 필드 표현식의 값 수. 예를 들어, COUNT(NAME) 은 이름 값의 수를 반환합니다. 필드 이름과 함께 COUNT를 사용하면, COUNT는 null이 아닌 필드 값의 수를 반환합니다. 특수 예제는 COUNT(*) 이며, null 값이 있는 레코드를 포함한 세트로 레코드 수를 반환합니다.
MAX	모든 필드 표현식의 최대값. 예를 들어, MAX(SALARY) 는 모든 월급 필드 값의 최대값을 반환합니다.
MIN	모든 필드 표현식의 최소값. 예를 들어, MIN(SALARY) 는 모든 월급 필드 값의 최소값을 반환합니다.

예제

```
SELECT SUM (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT AVG (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT COUNT (Sales_Data.Amount) AS agg FROM Sales_Data

SELECT MAX (Sales_Data.Amount) AS agg FROM Sales_Data
   WHERE Sales_Data.Amount < 3000

SELECT MIN (Sales_Data.Amount) AS agg FROM Sales_Data
   WHERE Sales_Data.Amount > 3000
```

집계 함수를 다른 함수에서 인수로 사용할 수 없습니다. 사용하는 경우, FileMaker에서 오류 코드 8309("Expressions involving aggregations are not supported")를 반환합니다. 예를 들어, 집계 함수 SUM을 ROUND 함수의 인수로 사용할 수 없기 때문에 다음 명령문은 유효하지 않습니다.

```
SELECT ROUND(SUM(Salary), 0) FROM Payroll
```

하지만 집계 함수는 숫자를 반환하는 함수를 인수로 사용할 수 있습니다. 다음은 유효한 명령문의 예입니다.

```
SELECT SUM(ROUND(Salary, 0)) FROM Payroll
```

문자열을 반환하는 함수

문자열을 반환하는 함수	설명	예제
CHR	ASCII 코드를 하나의 문자열로 반환	CHR(67)은 **C**를 반환
CURRENT_USER	연결 시 지정된 로그인 ID를 반환	
DAYNAME	지정된 날짜에 해당되는 날짜의 이름을 반환	
RTRIM	문자열에서 후행 공백 제거	RTRIM(' ABC ')는 ' ABC'를 반환
TRIM	문자열에서 선행 및 후행 공백 제거	TRIM(' ABC ')는 'ABC'를 반환
LTRIM	문자열에서 선행 공백 제거	LTRIM(' ABC')는 'ABC'를 반환
UPPER	문자열의 각 문자를 대문자로 변경	UPPER('Allen')은 'ALLEN'을 반환
LOWER	문자열의 각 문자를 소문자로 변경	LOWER('Allen')은 'allen'을 반환
LEFT	문자열에서 가장 왼쪽 문자를 반환	LEFT('Mattson',3)은 'Mat'을 반환
MONTHNAME	캘린더 월의 이름을 반환	
RIGHT	문자열에서 가장 오른쪽 문자를 반환	RIGHT('Mattson',4)는 'tson'을 반환
SUBSTR SUBSTRING	문자열의 매개변수, 추출할 첫 번째 문자 및 추출할 문자 수(옵션)와 함께 문자열의 하위 문자열을 반환	SUBSTR('Conrad',2,3)은 'onr'을 반환 SUBSTR('Conrad',2)는 'onrad'를 반환
SPACE	빈 문자열 생성	SPACE(5)는 ' '를 반환
STRVAL	임의의 유형의 값을 문자열로 변환	STRVAL('Woltman')은 'Woltman'을 반환 STRVAL(5 * 3)은 '15'를 반환 STRVAL(4 = 5)는 'False'를 반환 STRVAL(DATE '2008-12-25') '2008-12-25'를 반환
TIME TIMEVAL	시간을 문자열로 반환	오후 9:49에서, TIME()은 **21:49:00**을 반환
USERNAME USER	연결 시 지정된 로그인 ID를 반환	

메모 TIME() 함수는 사용되지 않습니다. 대신 SQL 표준 CURRENT_TIME을 사용합니다.

예제

```
SELECT CHR(67) + SPACE(1) + CHR(70) FROM Salespeople

SELECT RTRIM(' ' + Salespeople.Salesperson_ID) AS agg FROM Salespeople

SELECT TRIM(SPACE(1) + Salespeople.Salesperson_ID) AS agg FROM
Salespeople

SELECT LTRIM(' ' + Salespeople.Salesperson_ID) AS agg FROM Salespeople

SELECT UPPER(Salespeople.Salesperson) AS agg FROM Salespeople

SELECT LOWER(Salespeople.Salesperson) AS agg FROM Salespeople

SELECT LEFT(Salespeople.Salesperson, 5) AS agg FROM Salespeople

SELECT RIGHT(Salespeople.Salesperson, 7) AS agg FROM Salespeople

SELECT SUBSTR(Salespeople.Salesperson_ID, 2, 2) +
SUBSTR(Salespeople.Salesperson_ID, 4, 2) AS agg FROM Salespeople

SELECT SUBSTR(Salespeople.Salesperson_ID, 2) +
SUBSTR(Salespeople.Salesperson_ID, 4) AS agg FROM Salespeople

SELECT SPACE(2) + Salespeople.Salesperson_ID AS Salesperson_ID FROM
Salespeople

SELECT STRVAL('60506') AS agg FROM Sales_Data WHERE Sales_Data.Invoice
= 1
```

숫자를 반환하는 함수

숫자를 반환하는 함수	설명	예제
ABS	숫자 표현식의 절대값을 반환	
ATAN	라디언으로 표현되는 각도로 인수의 아크탄젠트를 반환	
ATAN2	라디언으로 표현되는 각도로 x축 또는 y축의 아크탄젠트를 반환	
CEIL CEILING	인수와 같거나 큰 가장 작은 정수 값을 반환	
DEG DEGREES	라디언으로 표현되는 각도인 인수의 자유도를 반환	
DAY	날짜의 일을 반환	DAY(DATE '2016-01-30')은 **30**을 반환
DAYOFWEEK	날짜 표현식의 주(1-7)를 반환	DAYOFWEEK(DATE '2004-05-01') **7**을 반환
MOD	두 수를 나누고 나눗셈의 나머지를 반환	MOD(10,3)은 **1**을 반환

숫자를 반환하는 함수	설명	예제
EXP	인수에서 지정한 거듭제곱으로 반올림된 밑이 자연 로그(e)인 값을 반환	
FLOOR	인수와 같거나 작은 가장 큰 정수 값을 반환	
HOUR	값의 시간 부분을 반환	
INT	숫자의 정수 부분을 반환	`Int(6.4321)` 은 **6**을 반환
LENGTH	문자열의 길이를 반환	`LENGTH('ABC')` 는 **3**을 반환
MONTH	날짜의 월 부분을 반환	`MONTH(DATE '2016-01-30')` 은 **1**을 반환
LN	인수의 자연 로그를 반환	
LOG	인수의 공통 로그를 반환	
MAX	두 숫자 중 큰 수를 반환	`MAX(66,89)` 는 **89**를 반환
MIN	두 숫자 중 작은 수를 반환	`MIN(66,89)` 는 **66**을 반환
MINUTE	값의 분 부분을 반환	
NUMVAL	문자열을 숫자로 변환. 문자열이 유효한 숫자가 아닌 경우 함수가 실패합니다.	`NUMVAL('123')` 은 **123**을 반환
PI	수학 상수 pi의 상수 값을 반환	
RADIANS	도로 표현되는 인수의 라디언 수를 반환	
ROUND	숫자를 반올림	`ROUND(123.456,0)` 은 **123**을 반환 `ROUND(123.456,2)` 는 **123.46**을 반환 `ROUND(123.456,-2)` 는 **100**을 반환
SECOND	값의 초 부분을 반환	
SIGN	인수의 부호 표시기: 음수의 경우 -1, 0은 0, 양수의 경우 1	
SIN	인수의 사인을 반환	
SQRT	인수의 제곱근을 반환	
TAN	인수의 탄젠트를 반환	
YEAR	날짜의 연도 부분을 반환	`YEAR(DATE '2016-01-30')` 은 **2016**을 반환

날짜를 반환하는 함수

날짜를 반환하는 함수	설명	예제
CURDATE CURRENT_DATE	오늘 날짜를 반환	
CURTIME CURRENT_TIME	현재 시간을 반환	
CURTIMESTAMP CURRENT_TIMESTAMP	현재 타임스탬프 값을 반환	
TIMESTAMPVAL	문자열을 타임스탬프로 변환	TIMESTAMPVAL('2016-01-30 14:00:00')은 타임스탬프 값을 반환합니다.
DATE TODAY	오늘 날짜를 반환	오늘이 11/21/2016인 경우, DATE()는 **2016-11-21**을 반환
DATEVAL	문자열을 날짜로 변환	DATEVAL('2016-01-30')은 **2016-01-30**을 반환

메모 DATE() 함수는 사용되지 않습니다. 대신 SQL 표준 CURRENT_DATE를 사용합니다.

조건 함수

조건 함수	설명	예제
CASE WHEN	단순 **CASE** 포맷 *input_exp*의 값을 *value_exp* 인수와 비교하여 결과를 결정합니다. CASE *input_exp* {WHEN *value_exp* THEN *result*...} [ELSE *result*] END	SELECT Invoice_ID, CASE Company_Name WHEN 'Exports UK' THEN 'Exports UK Found' WHEN 'Home Furniture Suppliers' THEN 'Home Furniture Suppliers Found' ELSE 'Neither Exports UK nor Home Furniture Suppliers' END, Salesperson_ID FROM Sales_Data
	검색된 **CASE** 포맷 WHEN 표현식에서 지정한 조건이 true인지 여부를 기반으로 결과를 반환합니다. CASE {WHEN *boolean_exp* THEN *result...*} [ELSE *result*] END	SELECT Invoice_ID, Amount, CASE WHEN Amount > 3000 THEN 'Above 3000' WHEN Amount < 1000 THEN 'Below 3000' ELSE 'Between 1000 and 3000' END, Salesperson_ID FROM Sales_Data

조건 함수	설명	예제
COALESCE	NULL이 아닌 첫 번째 값을 반환	SELECT Salesperson_ID, COALESCE(Sales_Manager, Salesperson) FROM Salespeople
NULLIF	두 값을 비교하고 두 값이 같은 경우 NULL을 반환하며 같지 않은 경우 첫 번째 값을 반환합니다.	SELECT Invoice_ID, NULLIF(Amount, -1), Salesperson_ID FROM Sales_Data

FileMaker 시스템 대상체

FileMaker 데이터베이스 파일은 SQL 쿼리를 사용하여 접근할 수 있는 다음의 시스템 대상체를 포함합니다.

FileMaker 시스템 테이블

모든 FileMaker 데이터베이스 파일에는 두 개의 시스템 테이블이 포함되어 있습니다. 바로 FileMaker_Tables 및 FileMaker_Fields입니다. ODBC 응용 프로그램의 경우, 이들 테이블은 카탈로그 함수 SQLTables에서 반환된 정보에 포함되어 있습니다. JDBC 응용 프로그램의 경우, 이들 테이블은 DatabaseMetaData 메소드 getTables에서 반환된 정보에 포함되어 있습니다. ExecuteSQL 함수에서도 해당 테이블을 사용합니다.

FileMaker_Tables

FileMaker_Tables 테이블에는 FileMaker 파일에서 정의된 데이터베이스 테이블에 대한 정보가 있습니다.

FileMaker_Tables 테이블에는 다음의 열과 함께 관계형 그래프에서 각 테이블 일치 항목에 대한 행이 있습니다.

- TableName - 테이블 일치 항목의 이름.
- TableId - 테이블 일치 항목의 고유 ID.
- BaseTableName - 테이블 일치 항목이 생성된 기본 테이블의 이름.
- BaseFileName - 기본 테이블을 포함하는 데이터베이스 파일의 FileMaker 파일 이름.
- ModCount - 이 테이블의 정의가 커밋된 총 변경 횟수.

예제

```
SELECT TableName FROM FileMaker_Tables WHERE TableName LIKE ' Sales%'
```

FileMaker_Fields 테이블

FileMaker_Fields 테이블에는 FileMaker 파일에서 정의된 필드에 대한 정보가 있습니다.

FileMaker_Fields 테이블에는 다음의 열이 있습니다.

- TableName - 필드를 포함하는 테이블의 이름.
- FieldName - 필드의 이름.
- FieldType - 필드의 SQL 데이터 유형.
- FieldId - 필드의 고유 ID.
- FieldClass - 다음 세 값 중 하나: Summary(요약 필드의 경우), Calculated(계산된 결과의 경우), Normal.
- FieldReps - 필드의 반복 횟수.
- ModCount - 이 테이블의 정의가 커밋된 총 변경 횟수.

예제:

```
SELECT * FROM FileMaker_Fields WHERE TableName=' Sales'
```

FileMaker 시스템 열

FileMaker는 시스템 열(필드)을 FileMaker 파일에서 정의된 모든 테이블의 모든 행(레코드)에 추가합니다. ODBC 응용 프로그램의 경우, 이들 열은 카탈로그 함수 SQLSpecialColumns에서 반환된 정보에 포함되어 있습니다. JDBC 응용 프로그램의 경우, 이들 열은 DatabaseMetaData 메소드 getVersionColumns에서 반환된 정보에 포함되어 있습니다. ExecuteSQL 함수에서도 해당 열을 사용합니다.

ROWID 열

`ROWID` 시스템 열은 레코드의 고유 ID 번호를 포함합니다. FileMaker Pro의 Get(레코드 ID) 함수가 반환하는 값과 동일합니다.

ROWMODID 열

`ROWMODID` 시스템 열은 현재 레코드가 커밋된 총 변경 횟수를 포함합니다. FileMaker Pro의 Get(레코드 수정 수) 함수가 반환하는 값과 동일합니다.

예제:

```
SELECT ROWID, ROWMODID FROM MyTable WHERE ROWMODID > 3
```

예약 SQL 키워드

이 부분은 열, 테이블, 별칭 또는 기타 사용자가 정의한 대상체로 사용되어서는 안 되는 예약 키워드를 나열합니다. 구문 오류가 발생하면 이 오류는 예약 단어 중 하나를 사용하기 때문에 발생할 수 있습니다. 이 키워드 중 하나를 사용하려면 따옴표를 사용하여 단어가 키워드로 처리되지 않도록 해야 합니다.

예를 들어 다음 `CREATE TABLE` 문은 `DEC` 키워드가 데이터 요소 이름으로 사용되는 방법을 표시합니다.

```
create table t ("dec" numeric)
```

ABSOLUTE
ACTION
ADD
ALL
ALLOCATE
ALTER
AND
ANY
ARE
AS
ASC
ASSERTION
AT
AUTHORIZATION
AVG
BEGIN
BETWEEN
BINARY
BIT
BIT_LENGTH
BLOB
BOOLEAN
BOTH
BY
CASCADE
CASCADED
CASE
CAST
CATALOG
CHAR
CHARACTER
CHARACTER_LENGTH
CHAR_LENGTH
CHECK
CHR
CLOSE
COALESCE
COLLATE
COLLATION
COLUMN
COMMIT
CONNECT
CONNECTION
CONSTRAINT
CONSTRAINTS
CONTINUE
CONVERT
CORRESPONDING
COUNT
CREATE
CROSS
CURDATE
CURRENT
CURRENT_DATE
CURRENT_TIME
CURRENT_TIMESTAMP
CURRENT_USER
CURSOR
CURTIME
CURTIMESTAMP
DATE
DATEVAL
DAY
DAYNAME
DAYOFWEEK
DEALLOCATE
DEC
DECIMAL
DECLARE
DEFAULT
DEFERRABLE
DEFERRED
DELETE
DESC
DESCRIBE
DESCRIPTOR
DIAGNOSTICS
DISCONNECT
DISTINCT
DOMAIN
DOUBLE
DROP
ELSE
END
END_EXEC
ESCAPE
EVERY

EXCEPT
EXCEPTION
EXEC
EXECUTE
EXISTS
EXTERNAL
EXTRACT
FALSE
FETCH
FIRST
FLOAT
FOR
FOREIGN
FOUND
FROM
FULL
GET
GLOBAL
GO
GOTO
GRANT
GROUP
HAVING
HOUR
IDENTITY
IMMEDIATE
IN
INDEX
INDICATOR
INITIALLY
INNER
INPUT
INSENSITIVE
INSERT
INT
INTEGER
INTERSECT
INTERVAL
INTO
IS
ISOLATION
JOIN
KEY
LANGUAGE
LAST
LEADING
LEFT
LENGTH
LEVEL
LIKE
LOCAL
LONGVARBINARY
LOWER
LTRIM
MATCH
MAX
MIN
MINUTE
MODULE
MONTH
MONTHNAME
NAMES
NATIONAL
NATURAL
NCHAR
NEXT
NO
NOT
NULL
NULLIF
NUMERIC
NUMVAL
OCTET_LENGTH
OF
OFFSET
ON
ONLY
OPEN
OPTION
OR
ORDER
OUTER
OUTPUT
OVERLAPS
PAD
PART
PARTIAL
PERCENT
POSITION
PRECISION
PREPARE
PRESERVE
PRIMARY
PRIOR
PRIVILEGES
PROCEDURE
PUBLIC
READ
REAL
REFERENCES
RELATIVE
RESTRICT
REVOKE
RIGHT
ROLLBACK
ROUND
ROW
ROWID
ROWS
RTRIM
SCHEMA
SCROLL
SECOND
SECTION
SELECT
SESSION
SESSION_USER

SET
SIZE
SMALLINT
SOME
SPACE
SQL
SQLCODE
SQLERROR
SQLSTATE
STRVAL
SUBSTRING
SUM
SYSTEM_USER
TABLE
TEMPORARY
THEN
TIES
TIME
TIMESTAMP
TIMESTAMPVAL
TIMEVAL
TIMEZONE_HOUR
TIMEZONE_MINUTE
TO
TODAY
TRAILING
TRANSACTION
TRANSLATE
TRANSLATION
TRIM
TRUE
TRUNCATE
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USERNAME
USING
VALUE
VALUES
VARBINARY
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

인덱스

M

N

O

P

R

S