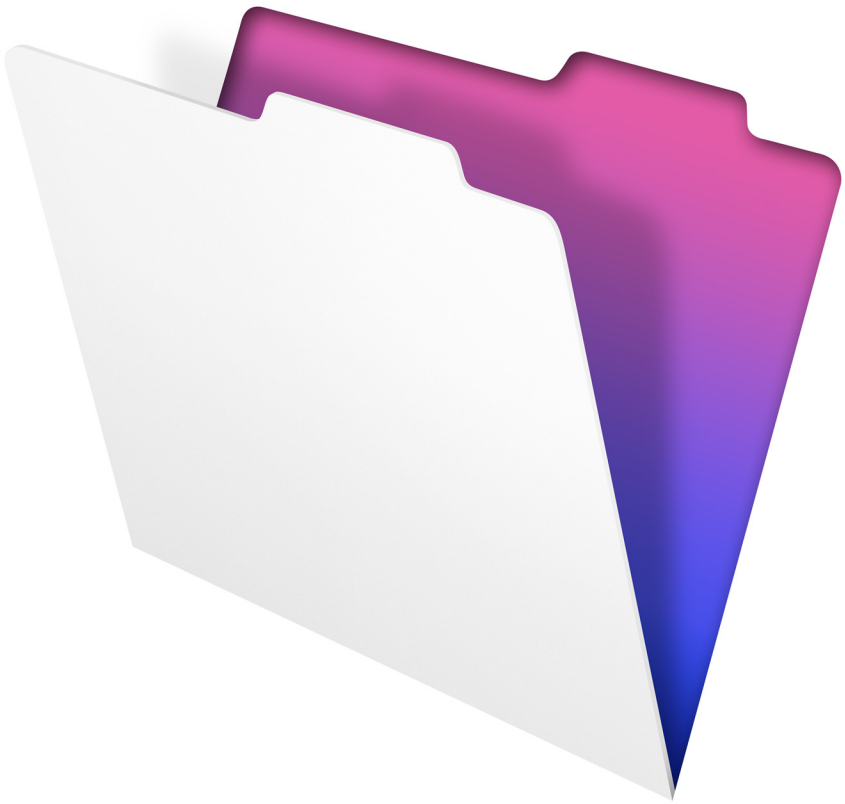
FileMaker® 13

Guide de référence SQL



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

Edition : 01

Table des matières

Chapitre 1
Introduction

En tant que développeur de bases de données, vous pouvez utiliser FileMaker Pro pour créer des solutions de bases de données sans connaître le langage SQL. En revanche, si vous connaissez le langage SQL, vous pouvez utiliser un fichier de base de données FileMaker comme source de données ODBC ou JDBC, en partageant vos données avec d'autres applications à l'aide d'ODBC et de JDBC. Vous pouvez également utiliser la fonction ExecuteSQL de FileMaker Pro pour récupérer des données de n'importe quelle occurrence de table dans une base de données FileMaker Pro.

Le présent guide de référence décrit les instructions et les normes SQL prises en charge par FileMaker. Les pilotes clients ODBC et JDBC de FileMaker prennent en charge toutes les instructions SQL décrites dans ce guide de référence. La fonction ExecuteSQL de FileMaker Pro prend uniquement en charge l'instruction SELECT.

A propos de ce guide

- Pour plus d'informations sur l'utilisation d'ODBC et de JDBC avec les versions antérieures de FileMaker Pro, consultez le site à l'adresse suivante : http://www.filemaker.fr/support/product/documentation.html.
- Ce guide de référence suppose que vous connaissez bien les principes d'utilisation de base des fonctions de FileMaker Pro, le codage des applications ODBC et JDBC et l'élaboration de requêtes SQL. Pour plus d'informations sur ces sujets, consultez un livre spécialisé.
- Ce guide de référence mentionne « FileMaker Pro » pour faire référence à FileMaker Pro et à FileMaker Pro Advanced, sauf quand il décrit des fonctions propres à FileMaker Pro Advanced.

Emplacement de la documentation au format PDF

Pour accéder aux PDF de la documentation FileMaker :

- Dans FileMaker Pro, sélectionnez le menu **Aide** > **Documentation produit**.
- Dans FileMaker Server, sélectionnez le menu **Aide** > **Documentation produit**.
- Pour consulter de la documentation supplémentaire, visitez le site à l'adresse suivante : http://www.filemaker.fr/support/product/documentation.html. Toutes les mises à jour de ce document sont également disponibles sur le site Web.

A propos de SQL

SQL, ou Structured Query Language, est un langage de programmation permettant d'interroger des données dans une base de données relationnelle. La principale instruction utilisée pour interroger une base de données est l'instruction SELECT.

Outre le langage permettant d'interroger une base de données, SQL fournit des instructions afin de manipuler des données pour ajouter, mettre à jour et supprimer des données.

SQL fournit également des instructions pour définir des données. Ces instructions vous permettent de créer et de modifier des tables et des index.

Les instructions et normes SQL prises en charge par FileMaker sont décrites dans chapitre 2, « Normes prises en charge ».

Utilisation d'une base de données FileMaker comme source de données

Lorsque vous hébergez une base de données FileMaker en tant que source de données ODBC ou JDBC, les données FileMaker peuvent être partagées avec des applications compatibles avec ODBC et JDBC. Les applications se connectent à la source de données FileMaker à l'aide du pilote client FileMaker, crée et exécute des requêtes SQL à l'aide d'ODBC ou de JDBC et traite les données récupérées dans la solution de base de données FileMaker.

Pour plus d'informations sur la manière dont vous pouvez utiliser le logiciel FileMaker en tant que source de données pour les applications ODBC et JDBC, consultez le *Guide ODBC et JDBC de FileMaker*.

Les pilotes clients ODBC et JDBC de FileMaker prennent en charge toutes les instructions SQL décrites dans ce guide de référence.

Utilisation de la fonction ExecuteSQL

La fonction ExecuteSQL de FileMaker Pro vous permet de récupérer des données dans les occurrences de tables dont le nom est affiché dans le graphe de liens, mais indépendamment de toute relation définie. Vous pouvez récupérer des données dans plusieurs tables sans créer de lien de table ou une quelconque relation entre les tables. Dans certains cas, vous pouvez réduire la complexité de votre graphe de liens grâce à la fonction ExecuteSQL.

Les rubriques que vous interrogez avec la fonction ExecuteSQL ne doivent pas nécessairement figurer sur un modèle, vous pouvez donc utiliser cette fonction pour récupérer des données indépendamment de tout contexte de modèle. En raison de cette indépendance du contexte, l'utilisation de la fonction ExecuteSQL dans des scripts peut améliorer la portabilité des scripts. Vous pouvez utiliser cette fonction partout où vous pouvez spécifier des calculs, notamment la création de graphiques et de rapports.

Elle prend uniquement en charge l'instruction SELECT, décrite dans la section « SELECT, instruction », page 7.

Aussi, cette fonction n'accepte que les formats de date et d'heure ISO syntaxe SQL-92 sans accolades ({}). Elle n'accepte pas les constantes de date, d'heure et d'horodatage au format ODBC/JDBC entre accolades.

Pour plus d'informations sur la syntaxe et l'utilisation de la fonction ExecuteSQL, consultez l'aide FileMaker Pro.

Chapitre 2
Normes prises en charge

Le présent guide de référence décrit les instructions et les normes SQL prises en charge par FileMaker. Les pilotes clients ODBC et JDBC de FileMaker prennent en charge toutes les instructions SQL décrites dans ce chapitre. La fonction ExecuteSQL de FileMaker Pro prend uniquement en charge l'instruction SELECT.

Les pilotes clients permettent d'accéder à une solution de base de données FileMaker à partir d'une application compatible ODBC ou JDBC. Cette solution peut être hébergée par FileMaker Pro ou FileMaker Server

- Le pilote client ODBC prend en charge ODBC 3.5 niveau 1, ainsi que quelques fonctions du niveau 2.
- Le pilote client JDBC prend partiellement en charge la spécification JDBC 3.0.
- Les pilotes clients ODBC et JDBC sont compatibles avec le niveau d'entrée de SQL-92, ainsi qu'avec certaines fonctions SQL-92 intermédiaires.

Prise en charge des caractères Unicode

Les pilotes clients ODBC et JDBC prennent en charge l'API Unicode. Toutefois, si vous créez une application personnalisée qui utilise les pilotes clients, utilisez le format ASCII pour les noms de rubriques, de tables et de fichiers (si vous utilisez un outil de requête ou une application non Unicode).

Remarque Pour insérer et extraire des données Unicode, utilisez `SQL_C_WCHAR`.

Instructions SQL

Les pilotes clients ODBC et JDBC prennent en charge les instructions SQL suivantes :

- SELECT (page 7)
- DELETE (page 16)
- INSERT (page 16)
- UPDATE (page 18)
- CREATE TABLE (page 19)
- ALTER TABLE (page 21)
- CREATE INDEX (page 22)
- DROP INDEX (page 22)

Les pilotes clients prennent aussi en charge la conversion du type de données FileMaker en types de données ODBC SQL et JDBC SQL. Pour plus d'informations sur les conversions de types de données, consultez le *Guide ODBC et JDBC de FileMaker*. Pour plus d'informations sur la création de requêtes SQL, consultez un ouvrage spécialisé.

Remarque Les pilotes clients ODBC et JDBC ne prennent pas en charge les tables externes FileMaker.

SELECT, instruction

L'instruction `SELECT` vous permet de spécifier les colonnes qui vous intéressent. Faites suivre l'instruction `SELECT` des expressions de colonne (comparables aux noms de rubriques) que vous voulez extraire (par exemple, `nom_famille`). Les expressions peuvent contenir des opérations mathématiques ou des instructions de manipulation de chaînes de texte (par exemple, `SALAIRE * 1.05`).

L'instruction `SELECT` peut être accompagnée de diverses clauses :

```
SELECT [DISTINCT] {* | expression_colonne [[AS] alias_colonne],...}
FROM nom_table [alias_table], ...
[ WHERE expr1 opérateur_rel expr2 ]
[ GROUP BY {expression_colonne, ...} ]
[ HAVING expr1 opérateur_rel expr2 ]
[ UNION [ALL] (SELECT...) ]
[ ORDER BY {expression_tri [DESC | ASC]}, ... ]
[ OFFSET n {ROWS | ROW} ]
[ FETCH FIRST [ n [ PERCENT ] ] { ROWS | ROW } {ONLY | WITH TIES } ]
[ FOR UPDATE [OF {expression_colonne, ...}] ]
```

Les éléments entre crochets sont facultatifs.

`alias_colonne` peut servir à attribuer à la colonne un nom plus descriptif ou à abréger un nom de colonne long. Par exemple, pour attribuer l'alias `service` à la colonne `serv` :

```
SELECT serv AS service FROM emp
```

Vous pouvez faire précéder le nom des rubriques par le nom ou de l'alias de la table. Par exemple, `EMP.NOM_FAMILLE` ou `E.NOM_FAMILLE`, où `E` est l'alias de la table `EMP`.

Vous pouvez faire précéder la première expression de colonne de l'opérateur `DISTINCT`. Cet opérateur supprime les rangées en double du résultat d'une requête. Par exemple :

```
SELECT DISTINCT serv FROM emp
```

Clauses SQL

Les pilotes clients ODBC et JDBC prennent en charge les clauses SQL suivantes.

Utilisez cette clause SQL	Pour
FROM (page 8)	Indiquer les tables utilisées dans l'instruction `SELECT`.
WHERE (page 9)	Indiquer les conditions que doivent remplir les enregistrements à extraire (comme dans une recherche sous FileMaker Pro).
GROUP BY (page 10)	Indiquer le nom d'une ou de plusieurs rubriques en fonction desquelles les valeurs renvoyées doivent être groupées. Cette clause sert à renvoyer un ensemble de valeurs statistiques en renvoyant une rangée par groupe (comme dans un sous-récapitulatif FileMaker Pro).
HAVING (page 10)	Indiquer les conditions s'appliquant à des groupes d'enregistrements (par exemple, afficher uniquement les services dont le total des salaires est supérieur à 200 000 euros).
UNION (page 11)	Combiner les résultats de deux instructions `SELECT` ou plus en un seul résultat.
ORDER BY (page 11)	Indiquer le mode de tri appliqué aux enregistrements.
OFFSET (page 12)	Indiquer le nombre de rangées à ignorer avant de commencer la récupération de rangées.
FETCH FIRST (page 12)	Indiquer le nombre de rangées à récupérer. Le nombre maximum de rangées spécifiées est renvoyé, il est possible que moins de rangées soient renvoyées si la requête fournit un nombre inférieur de rangées spécifiées.
FOR UPDATE (page 13)	Procéder à des mises à jour positionnées ou à des suppressions positionnées via les curseurs SQL

Remarque Si vous essayez de récupérer des données depuis une table sans colonne, l'instruction `SELECT` ne renvoie rien.

Clause FROM

La clause `FROM` indique quelles tables sont utilisées dans l'instruction `SELECT`. La syntaxe de cette clause est la suivante :

```
FROM nom_table [alias_table], [, nom_table [alias_table]]
```

`nom_table` correspond au nom d'une table dans la base de données active. Le nom de la table doit commencer par des caractères alphabétiques. Si le nom de la table commence par un caractère autre qu'un caractère alphabétique, placez-le entre des guillemets doubles (identifiant cité).

`alias_table` peut être utilisé pour attribuer à la table un nom plus descriptif, pour abréger un nom de table plus long ou pour inclure la même table dans la requête à plusieurs reprises (par exemple dans les liens internes).

Les noms des rubriques doivent commencer par des caractères alphabétiques. Si le nom d'une rubrique commence par un caractère autre qu'un caractère alphabétique, placez-le entre des guillemets doubles (identifiant cité). Par exemple, l'instruction `ExecuteSQL` de la rubrique intitulée `_LASTNAME` est :

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

Vous pouvez faire précéder le nom des rubriques par le nom ou de l'alias de la table. Par exemple, à partir de la spécification de table `FROM employés E`, vous pouvez faire référence à la rubrique `NOM_FAMILLE` sous la forme `E.NOM_FAMILLE`. Les alias de tables doivent être utilisés si l'instruction `SELECT` joint une table à elle-même. Par exemple :

```
SELECT * FROM employés E, employés F WHERE E.id_directeur =
F.id_employé
```

Le signe égal (=) n'inclut que les rangées correspondantes dans le résultat.

Si vous joignez plusieurs tables et que vous souhaitez ignorer toutes les rangées qui n'ont pas de rangées correspondantes dans les deux tables sources, vous pouvez utiliser une instruction `INNER JOIN`. Par exemple :

```
SELECT *
   FROM Vendeurs INNER JOIN Informations_Ventes
   ON Vendeurs.ID_Vendeur = Informations_Ventes.ID_Vendeur
```

Si vous liez deux tables, mais ne souhaitez pas ignorer les rangées de la première table (table de «gauche»), vous pouvez utiliser `LEFT OUTER JOIN`.

```
SELECT *
   FROM Vendeurs LEFT OUTER JOIN Informations_Ventes
   ON Vendeurs.ID_Vendeur = Informations_Ventes.ID_Vendeur
```

Chaque rangée de la table « Vendeurs » apparaît dans la table liée.

Remarques

- `RIGHT OUTER JOIN` n'est pas pris en charge actuellement.
- `FULL OUTER JOIN` n'est pas pris en charge actuellement.

Clause WHERE

La clause `WHERE` indique les conditions que les enregistrements doivent remplir pour être extraits. Elle spécifie ces conditions sous la forme suivante :

```
WHERE expr1 opérateur_rel expr2
```

`expr1` et `expr2` peuvent être des noms de rubriques, des valeurs constantes ou des expressions.

`opérateur_rel` est l'opérateur relationnel qui lie les deux expressions. Par exemple, l'instruction `SELECT` suivante extrait le nom des employés dont le salaire est supérieur ou égal à 20 000 euros.

```
SELECT nom_famille,prénom FROM emp WHERE salaire = 20000
```

La clause WHERE peut également utiliser des expressions telles que les suivantes :

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

Remarque Si vous utilisez des noms entièrement qualifiés dans la liste (de projection) SELECT, vous devez également utiliser les noms entièrement qualifiés dans la clause WHERE liée.

Clause GROUP BY

La clause GROUP BY indique les noms d'une ou de plusieurs rubriques devant servir à grouper les valeurs renvoyées. Cette clause sert à renvoyer un jeu de valeurs statistiques. La syntaxe de cette clause est la suivante :

```
GROUP BY colonnes
```

colonnes doit correspondre à l'expression de colonne utilisée dans la clause SELECT. Une expression de colonne peut être composée d'un ou de plusieurs noms de rubriques de la table de base de données, séparés par des virgules.

Exemple

L'exemple suivant calcule la somme des salaires de chaque service.

```
SELECT id_serv, SUM (salaire) FROM emp GROUP BY id_serv
```

Cette instruction renvoie une rangée pour chaque ID de service distinct. Chaque rangée contient l'ID de service et calcule la somme des salaires des employés du service.

Clause HAVING

La clause HAVING vous permet de spécifier des conditions pour des groupes d'enregistrements (par exemple, afficher uniquement les services sont le total des salaires est supérieur à 200 000 euros). La syntaxe de cette clause est la suivante :

```
HAVING expr1 opérateur_rel expr2
```

expr1 et expr2 peuvent être des noms de rubriques, des valeurs constantes ou des expressions. Ces expressions n'ont pas besoin de correspondre à une expression de colonne dans la clause SELECT.

opérateur_rel est l'opérateur relationnel qui lie les deux expressions.

Exemple

L'exemple suivant renvoie uniquement les services dont la somme des salaires est supérieure à 200 000 euros :

```
SELECT id_serv, SUM (salaire) FROM emp
   GROUP BY id_serv HAVING SUM (salaire) > 200000
```

Opérateur UNION

L'opérateur UNION combine les résultats de deux instructions SELECT ou plus en un seul et même résultat. Ce résultat correspond à l'ensemble des enregistrements renvoyés par les instructions SELECT. Par défaut, le système ne renvoie pas les enregistrements en double. Si vous les voulez également, employez le mot-clé ALL (UNION ALL). La syntaxe de cette clause est la suivante :

```
instruction_SELECT UNION [ALL] instruction_SELECT
```

Avec l'opérateur UNION, les listes de sélection de chaque instruction SELECT doivent avoir le même nombre d'expressions de colonne, avec les mêmes types de données et le même ordre. Par exemple :

```
SELECT nom_famille, salaire, date_embauche FROM emp UNION SELECT nom,
paie, date_naissance FROM personne
```

Cet exemple utilise le même nombre d'expressions de colonne et chaque expression de colonne, placée dans le même ordre, utilise le même type de données.

L'exemple suivant n'est pas valide car les types de données des expressions de colonne sont différents (la rubrique SALAIRE de la table EMP n'emploie pas le même type de données que la rubrique NOM_FAMILLE de la table AUGMENTATIONS). Cet exemple utilise le même nombre d'expressions de colonne dans chaque instruction SELECT, mais ces expressions ne sont pas dans le même ordre par type de données.

```
SELECT nom_famille, salaire FROM emp UNION SELECT salaire,
nom_famille FROM augmentations
```

Clause ORDER BY

La clause ORDER BY indique comment les enregistrements doivent être triés. La syntaxe de cette clause est la suivante :

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

expression_tri peut être des noms de rubriques, des expressions ou le numéro de position de l'expression de colonne à utiliser. Par défaut, le tri est croissant (ASC).

Par exemple, pour trier par nom_famille, puis par prénom, vous pouvez utiliser l'une des instructions SELECT suivantes :

```
SELECT id_emp, nom_famille, prénom FROM emp ORDER BY nom_famille,
prénom
```

ou

```
SELECT id_emp, nom_famille, prénom FROM emp ORDER BY 2,3
```

Dans le second exemple, nom_famille est la deuxième expression de colonne après SELECT. ORDER BY 2 trie donc par nom_famille.

Clauses OFFSET et FETCH FIRST

Les clauses `OFFSET` et `FETCH FIRST` permettent de renvoyer une plage de rangées en commençant par un point de départ donné dans un ensemble de résultats. La capacité à limiter les rangées récupérées dans des ensembles de résultats étendus vous permet de passer les données en revue et d'améliorer l'efficacité.

La clause `OFFSET` indique le nombre de rangées à ignorer avant de commencer à renvoyer des données. Si la clause `OFFSET` n'est pas utilisée dans une instruction `SELECT`, la rangée de départ sera 0. La clause `FETCH FIRST` indique le nombre de rangées à renvoyer, sous la forme d'un entier non signé supérieur ou égal à 1 ou sous la forme d'un pourcentage, depuis le point de départ indiqué dans la clause `OFFSET`. Si les deux clauses `OFFSET` et `FETCH FIRST` sont utilisées dans une instruction `SELECT`, la clause `OFFSET` doit être prioritaire.

Les clauses `OFFSET` et `FETCH FIRST` ne sont pas prises en charge dans les sous-requêtes.

Format OFFSET

Le format `OFFSET` est le suivant :

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

`n` est un entier non signé. Si `n` est supérieur au nombre de rangées renvoyées dans l'ensemble de résultats, alors rien ne sera renvoyé et aucun message d'erreur n'apparaîtra.

`ROWS` est l'équivalent de `ROW`.

Format FETCH FIRST

Le format `FETCH FIRST` est le suivant :

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

`n` indique le nombre de rangées à renvoyer. La valeur par défaut est 1 si `n` est omis, `n` est un entier non signé supérieur ou égal à 1, à moins qu'il ne soit suivi par `PERCENT`. Si `n` est suivi par `PERCENT`, la valeur peut être une valeur fractionnelle positive ou un entier non signé.

`ROWS` est l'équivalent de `ROW`.

`WITH TIES` doit être utilisé avec la clause `ORDER BY`.

`WITH TIES` permet de renvoyer davantage de rangées contrairement à la valeur spécifiée dans `FETCH` car les rangées homologues, ces rangées qui ne sont pas distinctes et basées sur la clause `ORDER BY`, sont également renvoyées.

Exemples

Par exemple, pour renvoyer des informations depuis la 26ème rangée de l'ensemble de résultats triés par `nom_famille`, puis par `prénom`, utilisez l'instruction `SELECT` suivante :

```
SELECT id_emp, nom_famille, prénom FROM emp ORDER BY nom_famille,
prénom OFFSET 25 ROWS
```

Pour indiquer que voulez uniquement renvoyer dix rangées :

```
SELECT id_emp, nom_famille, prénom FROM emp ORDER BY nom_famille,
prénom OFFSET 25 ROWS FETCH FIRST 10 ROWS ONLY
```

Pour renvoyer dix rangées et leurs rangées homologues (rangées qui ne sont pas distinctes et basées sur la clause `ORDER BY`) :

```
SELECT id_emp, nom_famille, prénom FROM emp ORDER BY nom_famille,
prénom OFFSET 25 ROWS FETCH FIRST 10 ROWS WITH TIES
```

Clause FOR UPDATE

La clause `FOR UPDATE` verrouille les enregistrements pour des mises à jour positionnées ou des suppressions positionnées via les curseurs SQL. La syntaxe de cette clause est la suivante :

```
FOR UPDATE [OF expressions_colonne]
```

`expressions_colonne` est une liste de noms de rubriques de la table de base de données que vous souhaitez mettre à jour, séparés par une virgule. `expressions_colonne` est facultatif et est ignoré.

Exemple

L'exemple suivant renvoie tous les enregistrements de la base de données d'employés dont la valeur de la rubrique `SALAIRE` est supérieure à 20 000 euros. Lorsque chaque enregistrement est récupéré, il est verrouillé. Si vous mettez à jour l'enregistrement ou que vous le supprimez, le verrou est maintenu jusqu'à ce que vous validiez la modification. Sinon, le verrou disparaît lorsque vous passez à l'enregistrement suivant.

```
SELECT * FROM emp WHERE salaire > 20000
   FOR UPDATE OF nom, prénom, salaire
```

Exemples supplémentaires :

Utilisation de	Exemple de code SQL
Constante de type texte	`SELECT 'ChienChat' FROM Vendeurs`
Constante de type numérique	`SELECT 999 FROM Vendeurs`
Constante de type date	`SELECT DATE '05/06/2012' FROM Vendeurs`
Constante de type heure	`SELECT TIME '02:49:03' FROM Vendeurs`
Constante de type horodatage	`SELECT TIMESTAMP '05/06/2012 02:49:03' FROM Vendeurs`
Colonne de texte	`SELECT Nom_Société FROM Informations_Ventes` `SELECT DISTINCT Nom_Société FROM Informations_Ventes`
colonne de type numérique	`SELECT Quantité FROM Informations_Ventes` `SELECT DISTINCT Quantité FROM Informations_Ventes`
Colonne de type date	`SELECT Date_Vente FROM Informations_Ventes` `SELECT DISTINCT Date_Vente FROM Informations_Ventes`
Colonne de type heure	`SELECT Heure_Vente FROM Informations_Ventes` `SELECT DISTINCT Heure_Vente FROM Informations_Ventes`
Colonne de type horodatage	`SELECT Horodatage_Vente FROM Informations_Ventes` `SELECT DISTINCT Horodatage_Vente FROM Informations_Ventes`
Colonne[a] Conteneur	`SELECT Brochures_Société FROM Informations_Ventes` `SELECT GETAS(Logo_Société, 'JPEG') FROM Informations_Ventes`
Caractère joker *	`SELECT * FROM Vendeurs` `SELECT DISTINCT * FROM Vendeurs`

a. Un conteneur de fichier de base de données est une rubrique qui contient un fichier de base de données FileMaker.

Notes à propos des exemples

Une `colonne` est une référence à une rubrique dans un fichier de base de données FileMaker cette rubrique peut contenir plusieurs valeurs distinctes).

Le caractère générique (*) est une manière plus courte de dire « tout ». L'exemple `SELECT * FROM Vendeurs` fait apparaître toutes les colonnes de la table `Vendeurs`. L'exemple `SELECT DISTINCT * FROM Vendeurs` fait apparaître toutes les rangées uniques de la table `Vendeurs` (sans doublons).

- FileMaker ne stocke pas de données pour les chaînes vides. Les requêtes suivantes ne renvoient donc jamais d'enregistrement :

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

- Si vous utilisez `SELECT` avec des données binaires, vous devez utiliser la fonction `GetAs()` pour indiquer le flux à renvoyer. Consultez la section «Récupération du contenu d'une rubrique Conteneur : fonctions CAST() et GetAs()», pour plus d'informations.

Récupération du contenu d'une rubrique Conteneur : fonctions CAST() et GetAs()

Depuis une rubrique Conteneur, vous pouvez récupérer des données binaires, des informations de référence de fichier ou des données appartenant à un type de fichier particulier.

S'il existe des données de fichier ou des données binaires JPEG, l'instruction `SELECT` avec la fonction `GetAS(nom rubrique, 'JPEG')` récupère des données au format binaire ; sinon, l'instruction `SELECT` avec un nom de rubrique renvoie la valeur `NULL`.

Pour récupérer des informations de référence de fichier depuis une rubrique Conteneur, comme le chemin d'accès à un fichier, à une image ou à une séquence Quicktime, utilisez la fonction `CAST()` avec une instruction `SELECT`. Par exemple :

```
SELECT CAST(Brochures_Société AS VARCHAR(NNN)) FROM
Informations_Ventes
```

Dans cet exemple :

- Si vous avez inséré un fichier dans la rubrique Conteneur en utilisant FileMaker Pro, mais que vous avez enregistré uniquement une référence dans le fichier, l'instruction `SELECT` récupère les informations de référence du fichier comme étant du type `SQL_VARCHAR`.
- Si vous avez inséré le contenu d'un fichier dans la rubrique Conteneur en utilisant FileMaker Pro, l'instruction `SELECT` récupère le nom de ce fichier.
- Si vous avez importé un fichier dans la rubrique Conteneur depuis une autre application, l'instruction `SELECT` affiche « ? » (le fichier apparaît sous la forme **Untitled.dat** dans FileMaker Pro).

Pour récupérer des données depuis une rubrique Conteneur, utilisez la fonction `GetAs()`. Vous pouvez utiliser l'option `DEFAULT` ou indiquer le type de fichier. L'option `DEFAULT` récupère le flux maître du conteneur sans devoir définir explicitement le type de flux :

```
SELECT GetAs(Brochures_Société, DEFAULT) FROM Informations_Ventes
```

Pour récupérer un type de flux individuel depuis un conteneur, utilisez la fonction `GetAs()` et indiquez le type du fichier en fonction du mode d'insertion des données dans la rubrique Conteneur dans FileMaker Pro. Par exemple :

- Si les données ont été insérées à l'aide de la commande **Insérer** > **Fichier**, indiquez `'FILE'` dans la fonction `GetAs()`. Par exemple :

  ```
  SELECT GetAs(Brochures_Société, 'FILE') FROM Informations_Ventes
  ```

- Si les données ont été insérées à l'aide de la commande **Insérer** > **Son** (audio standard — format brut Mac OS X), indiquez `'snd'` dans la fonction `GetAs()`. Par exemple :

  ```
  SELECT GetAs(Réunion_Société, 'snd') FROM LettreInfo_Société
  ```

- Si les données ont été insérées à l'aide de la commande **Insertion** > **Image**, faites glisser et déposez le fichier ou copiez-le depuis le Presse-papiers et indiquez l'un des types de fichiers répertorié dans le tableau ci-dessous. Par exemple :

```
SELECT GetAs(Logo_Société, 'JPEG') FROM Icônes_Société
```

Type de fichier	Description	Type de fichier	Description
'GIFf'	Graphics Interchange Format	'PNTG'	MacPaint
'JPEG'	Photos	'.SGI'	Format bitmap générique
'JP2 '	JPEG 2000	'TIFF'	Format de fichier trame pour images numériques
'PDF '	Portable Document Format	'TPIC'	Targa
'PNGf'	Format d'image Bitmap	'8BPS'	PhotoShop (PSD)

Instruction DELETE

L'instruction `DELETE` permet de supprimer des enregistrements d'une table de base de données. La syntaxe de cette instruction est la suivante :

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

Remarque La clause `WHERE` indique quels enregistrements doivent être supprimés. Si vous n'incluez pas le mot-clé `WHERE`, tous les enregistrements de la table seront supprimés (mais la table restera intacte).

Exemple

Voici un exemple d'instruction `DELETE` appliquée à la table Employés :

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

Chaque instruction `DELETE` supprime chaque enregistrement réunissant les conditions définies dans la clause `WHERE`. Ici, chaque enregistrement contenant l'ID d'employé `E10001` est supprimé. Les ID d'employés étant uniques dans la table Employés, un seul enregistrement est supprimé.

Instruction INSERT

L'instruction `INSERT` permet de créer des enregistrements dans une table de base de données. Vous pouvez spécifier, au choix :

- Une liste de valeurs à insérer sous la forme d'un nouvel enregistrement
- Une instruction `SELECT` permettant de copier les données d'une autre table et de les insérer sous forme de nouveaux enregistrements

La syntaxe de cette instruction est la suivante :

```
INSERT INTO nom_table [(nom_col, ...)] VALUES (expr, ...)
```

nom_col est une liste facultative de noms de colonnes qui indique le nom et l'ordre des colonnes dont les valeurs sont spécifiées dans la clause VALUES. Si vous omettez nom_col, les expressions de valeur (expr) doivent fournir des valeurs pour toutes les colonnes définies dans la table et elles doivent figurer dans le même ordre que les colonnes définies pour la table. nom_col peut également indiquer la répétition d'une rubrique, par exemple : lastDates[4].

expr représente la liste d'expressions donnant les valeurs des colonnes du nouvel enregistrement. Généralement, les expressions sont les valeurs constantes des colonnes (mais il peut également s'agir d'une sous-requête). Les valeurs de type chaîne de caractères doivent être placées entre guillemets simples ('). Pour inclure un guillemet simple (correspondant à une apostrophe) dans une valeur de type chaîne de caractères placée entre guillemets simples, utilisez deux guillemets simples consécutifs (par exemple 'L''instance').

Les sous-requêtes doivent être entre parenthèses.

L'exemple suivant insère une liste d'expressions :

```
INSERT INTO emp (nom_famille, prénom, id_emp, salaire, date_embauche)
   VALUES ('Martin', 'Robert', 'E22345', 27500, DATE '05-06-2013')
```

Chaque instruction INSERT ajoute un enregistrement à la table de base de données. Ici, un enregistrement a été ajouté à la table EMP (employés) de la base de données, avec des valeurs pour cinq colonnes. Les autres colonnes de la table n'ont pas reçu de valeur. Autrement dit, leur valeur est nulle.

Remarque Dans les rubriques Conteneur, vous ne pouvez utiliser INSERT que pour du texte, à moins de préparer une déclaration paramétrée et de générer le flux de données depuis votre application. Pour utiliser des données binaires, vous pouvez simplement assigner le nom de fichier en l'insérant entre des guillemets simples ou utiliser la fonction PutAs(). Lorsque vous spécifiez un nom de fichier, le type de fichier est déduit de l'extension de fichier :

```
INSERT INTO nom_table (nom_conteneur) VALUES(? AS
'nomfichier.extension fichier')
```

Les types de fichiers non pris en charge seront insérés en tant que type FILE.

Lorsque vous utilisez la fonction PutAs(), indiquez le type : PutAs(col, 'type'), où la valeur du type est un type de fichier pris en charge comme décrit dans la section « Récupération du contenu d'une rubrique Conteneur : fonctions CAST() et GetAs() », page 15.

L'instruction SELECT est une requête qui renvoie des valeurs pour chaque valeur nom_col spécifiée dans la liste de noms de colonnes. L'utilisation d'une instruction SELECT plutôt qu'une liste d'expressions de valeurs vous permet de sélectionner un ensemble de rangées dans une table et de l'insérer dans une autre table à l'aide d'une instruction INSERT unique.

Voici un exemple d'instruction INSERT utilisant une instruction SELECT :

```
INSERT INTO emp1 (prénom, nom_famille, id_emp, service, salaire)
   SELECT prénom, nom_famille, id_emp, service, salaire from emp
   WHERE service = 'D050'
```

Dans ce type d'instruction INSERT, le nombre de colonnes à insérer doit être le même que celui des colonnes spécifiées dans l'instruction SELECT La liste de colonnes à insérer doit être identique aux colonnes de l'instruction SELECT comme c'est le cas avec une liste d'expressions de valeurs dans l'autre type d'instruction INSERT. Par exemple, la première colonne insérée correspond à la première colonne sélectionnée, la deuxième à la deuxième colonne sélectionnée, et ainsi de suite.

La taille et le type de données de ces colonnes correspondantes doivent être compatibles. Chaque colonne de la liste SELECT doit avoir un type de données accepté par le pilote client ODBC ou JDBC pour une opération INSERT/UPDATE habituelle sur la colonne correspondante de la liste INSERT. Les valeurs sont tronquées si la taille de la valeur de la colonne SELECT est supérieure à celle de la colonne INSERT correspondante.

L'instruction SELECT est évaluée avant l'insertion de valeurs.

Instruction UPDATE

L'instruction UPDATE permet de modifier les enregistrements d'une table de base de données. La syntaxe de cette instruction est la suivante :

```
UPDATE nom_table SET nom_col = expr, ... [ WHERE { conditions } ]
```

nom_col est le nom d'une colonne dont la valeur doit être modifiée. Vous pouvez modifier plusieurs colonnes dans la même instruction.

expr est la nouvelle valeur de la colonne.

Généralement, les expressions sont les valeurs constantes des colonnes (mais il peut également s'agir d'une sous-requête). Les valeurs de type chaîne de caractères doivent être placées entre guillemets simples ('). Pour inclure un guillemet simple (correspondant à une apostrophe) dans une valeur de type chaîne de caractères placée entre guillemets simples, utilisez deux guillemets simples consécutifs (par exemple 'L''instance').

Les sous-requêtes doivent être entre parenthèses.

La clause WHERE peut être toute clause valide. Elle détermine les enregistrements à modifier.

Exemples

Voici un exemple d'instruction UPDATE appliquée à la table Employés :

```
UPDATE emp SET salaire=32000, exempt=1 WHERE id_emp = 'E10001'
```

Cette instruction UPDATE a pour effet de modifier chaque enregistrement réunissant les conditions définies dans la clause WHERE. Ici, il s'agit de changer le salaire et le statut d'exemption pour tous les employés dont l'ID est E10001. Les ID des employés étant uniques dans la table Employés, un seul enregistrement est mis à jour.

Voici un exemple comportant une sous-requête :

```
UPDATE emp SET salaire = (SELECT avg(salaire) from emp) WHERE
id_emp = 'E10001'
```

Ici, le salaire dont l'ID d'employé est `E10001` est remplacé par le salaire moyen de la société.

Remarque Dans les rubriques Conteneur, vous ne pouvez utiliser `UPDATE` qu'avec du texte, à moins de préparer une déclaration paramétrée et de générer le flux de données depuis votre application. Pour utiliser des données binaires, vous pouvez simplement assigner le nom de fichier en l'insérant entre des guillemets simples ou utiliser la fonction `PutAs()`. Lorsque vous spécifiez un nom de fichier, le type de fichier est déduit de l'extension de fichier :

```
UPDATE nom_table SET (nom_conteneur) = ? AS 'nomfichier.extension
fichier'
```

Les types de fichiers non pris en charge seront insérés en tant que type FILE.

Lorsque vous utilisez la fonction `PutAs()`, indiquez le type : `PutAs(col, 'type')`, où la valeur du type est un type de fichier pris en charge comme décrit dans la section « Récupération du contenu d'une rubrique Conteneur : fonctions CAST() et GetAs() », page 15.

Instruction CREATE TABLE

L'instruction `CREATE TABLE` permet de créer une table dans un fichier de base de données. La syntaxe de cette instruction est la suivante :

```
CREATE TABLE nom_table ( liste_éléments_de_table [,
liste_éléments_de_table...] )
```

Dans cette instruction, vous devez indiquer le nom et le type de données de chaque colonne.

- `nom_table` correspond au nom de la table. `nom_table` est limité à 100 caractères. Aucune table portant ce nom ne doit avoir été définie. Le nom de la table doit commencer par un caractère alphabétique. Si le nom de la table commence par un caractère autre qu'un caractère alphabétique, placez-le entre des guillemets doubles (identifiant cité).

- Le format de `liste_éléments_table` est le suivant :

```
nom_rubrique type_rubrique [DEFAULT expr]
[UNIQUE | NOT NULL | PRIMARY KEY | GLOBAL]
[EXTERNAL chaîne_chemin_relatif [SECURE | OPEN chaîne_chemin_calc]]
```

  - `nom_rubrique` correspond au nom de la rubrique. Aucune rubrique de la même table ne doit porter ce même nom. Pour définir une rubrique multivaluée, vous devez utiliser un chiffre entre crochets. Par exemple : `lastDates[4]`. Les noms des rubriques commencent par un caractère alphabétique. Si le nom d'une rubrique commence par un caractère autre qu'un caractère alphabétique, placez-le entre des guillemets doubles (identifiant cité). Par exemple, l'instruction `CREATE TABLE` de la rubrique intitulée `_LASTNAME` est :

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

  - `type_rubrique` peut être l'un des éléments suivants : `NUMERIC`, `DECIMAL`, `INT`, `DATE`, `TIME`, `TIMESTAMP`, `VARCHAR`, `CHARACTER VARYING`, `BLOB`, `VARBINARY`, `LONGVARBINARY` ou `BINARY VARYING`. Pour `NUMERIC` et `DECIMAL`, vous pouvez définir la précision et l'échelle. Par exemple : `DECIMAL(10,0)`. Pour `TIME` et `TIMESTAMP`, vous pouvez définir la précision. Par exemple : `TIMESTAMP(6)`. Pour `VARCHAR` et `CHARACTER VARYING`, vous pouvez définir la longueur de la chaîne. Par exemple : `VARCHAR(255)`.
  - Le mot-clé `DEFAULT` vous permet de définir une valeur par défaut pour une colonne. Pour `expr`, vous pouvez utiliser une valeur constante ou une expression. Les expressions autorisées sont les suivantes : `USER`, `USERNAME`, `CURRENT_USER`, `CURRENT_DATE`, `CURDATE`, `CURRENT_TIME`, `CURTIME`, `CURRENT_TIMESTAMP`, `CURTIMESTAMP` et `NULL`.
  - Quand vous définissez une colonne comme étant `UNIQUE`, l'option de validation **Unique** est sélectionnée automatiquement pour la rubrique correspondante dans le fichier de base de données FileMaker.
  - Quand vous définissez une colonne en tant que `NOT NULL`, l'option de validation **Non vide** est sélectionnée automatiquement pour la rubrique correspondante dans le fichier de base de données FileMaker. La rubrique est marquée en tant que **Valeur requise** dans l'onglet **Rubriques** de la boîte de dialogue Gérer la base de données de FileMaker Pro.
  - Pour définir une colonne en tant que rubrique Conteneur, utilisez `BLOB`, `VARBINARY` ou `BINARY VARYING` en guise de `type_rubrique`.
  - Pour définir une colonne en tant que rubrique Conteneur stockant les données en externe, utilisez le mot-clé `EXTERNAL`. L'élément `chaîne_chemin_relatif` définit le dossier dans lequel les données sont stockées en externe, par rapport à l'emplacement de la base de données FileMaker. Ce chemin doit être défini en tant que répertoire de base dans la boîte de dialogue Gérer les conteneurs de FileMaker Pro. Vous devez préciser `SECURE` pour un stockage sécurisé ou `OPEN` pour un stockage ouvert. Si vous utilisez un stockage ouvert, l'élément `chaîne_chemin_calc` correspond au dossier figurant dans le dossier `chaîne_chemin_relatif` où les objets Conteneur doivent être stockés. Le chemin doit utiliser des barres obliques (/) dans le nom de dossier.

Exemples

Utilisation de	Exemple de code SQL
Colonne de texte	`CREATE TABLE T1 (C1 VARCHAR, C2 VARCHAR (50), C3 VARCHAR (1001), C4 VARCHAR (500276))`
colonne de type texte, `NOT NULL`	`CREATE TABLE T1NN (C1 VARCHAR NOT NULL, C2 VARCHAR (50) NOT NULL, C3 VARCHAR (1001) NOT NULL, C4 VARCHAR (500276) NOT NULL)`
colonne de type numérique	`CREATE TABLE T2 (C1 DECIMAL, C2 DECIMAL (10,0), C3 DECIMAL (7539,2), C4 DECIMAL (497925,301))`
Colonne de type date	`CREATE TABLE T3 (C1 DATE, C2 DATE, C3 DATE, C4 DATE)`
Colonne de type heure	`CREATE TABLE T4 (C1 HEURE, C2 HEURE, C3 HEURE, C4 HEURE)`
Colonne de type horodatage	`CREATE TABLE T5 (C1 TIMESTAMP, C2 TIMESTAMP, C3 TIMESTAMP, C4 TIMESTAMP)`
colonne pour rubrique Conteneur	`CREATE TABLE T6 (C1 CONTENEUR, C2 CONTENEUR, C3 CONTENEUR, C4 CONTENEUR)`
colonne pour rubrique Conteneur de stockage externe	`CREATE TABLE T7 (C1 BLOB EXTERNAL 'Files/MyDatabase/' SECURE)` `CREATE TABLE T8 (C1 BLOB EXTERNAL 'Files/MyDatabase/' OPEN 'Objects')`

Instruction ALTER TABLE

L'instruction `ALTER TABLE` permet de modifier la structure d'une table existante dans un fichier de base de données. Chaque instruction ne permet de modifier qu'une seule colonne. Les formats de l'instruction `ALTER TABLE` sont les suivants :

```
ALTER TABLE nom_table ADD [COLUMN] définition_colonne

ALTER TABLE nom_table DROP [COLUMN] nom_colonne_non_qualifié

ALTER TABLE nom_table ALTER [COLUMN] définition_colonne SET DEFAULT expr

ALTER TABLE nom_table ALTER [COLUMN] définition_colonne DROP DEFAULT
```

Vous devez connaître la structure de la table et savoir comment vous souhaitez la modifier avant d'utiliser l'instruction `ALTER TABLE`.

Exemples

Pour	Exemple de code SQL
Ajouter des colonnes	`ALTER TABLE Vendeurs ADD (C1 VARCAR)`
Supprimer des colonnes	`ALTER TABLE Vendeurs DROP (C1)`
Définir la valeur par défaut d'une colonne	`ALTER TABLE Vendeurs ALTER Société SET DEFAULT 'FileMaker'`
Supprimer la valeur par défaut d'une colonne	`ALTER TABLE Vendeurs ALTER Société DROP DEFAULT`

Remarque `SET DEFAULT` et `DROP DEFAULT` n'affectent pas les rangées existantes de la table, mais modifient la valeur par défaut des rangées ajoutées par la suite à la table.

Instruction CREATE INDEX

L'instruction `CREATE INDEX` permet d'accélérer les recherches dans votre fichier de base de données. La syntaxe de cette instruction est la suivante :

```
CREATE INDEX ON nom_table.nom_col
CREATE INDEX ON nom_table (nom_col)
```

`CREATE INDEX` ne fonctionne qu'avec une colonne unique (les index multi-colonne ne sont pas pris en charge). Les index ne sont pas autorisés sur les colonnes correspondant à des rubriques de données de conteneur ou statistique, à des rubriques employant des options de stockage global ou à des rubriques de type calcul non stockées.

Quand vous créez un index pour une colonne de type texte, l'option de stockage **Minimal** est sélectionnée automatiquement dans **Indexation** pour la rubrique correspondante du fichier de base de données FileMaker. Quand vous créez un index pour une colonne qui n'est pas de type texte (ou pour une colonne de type texte au format japonais), l'option de stockage **Tout** est sélectionnée automatiquement dans **Indexation** pour la rubrique correspondante du fichier de base de données FileMaker.

Quand vous créez un index pour une colonne, l'option de stockage **Indexation automatique si nécessaire** est sélectionnée automatiquement dans **Indexation** pour la rubrique correspondante du fichier de base de données FileMaker.

FileMaker crée automatiquement les index selon les besoins. L'utilisation de `CREATE INDEX` entraîne la génération immédiate de l'index plutôt qu'à la demande.

Exemple

```
CREATE INDEX ON Vendeurs.ID_vendeur
```

Instruction DROP INDEX

L'instruction `DROP INDEX` permet de supprimer un index d'un fichier de base de données. Le format de l'instruction `DROP INDEX` est le suivant :

```
DROP INDEX ON nom_table.nom_col
DROP INDEX ON nom_table (nom_col)
```

Vous pouvez supprimer un index quand votre fichier de base de données est trop volumineux ou quand vous n'employez pas souvent une rubrique dans les requêtes.

Si les performances de vos requêtes ne sont pas satisfaisantes et que vous travaillez sur un fichier de base de données FileMaker extrêmement volumineux contenant un grand nombre de rubriques de type texte indexées, pensez à supprimer les index de certaines rubriques. Pensez également à supprimer les index des rubriques que vous utilisez rarement dans des instructions `SELECT`.

Quand vous supprimez un index pour une colonne, l'option de stockage **Aucun** est sélectionnée et l'option **Indexation automatique si nécessaire** est désélectionnée automatiquement dans **Indexation** pour la rubrique correspondante du fichier de base de données FileMaker.

L'attribut `PREVENT INDEX CREATION` n'est pas pris en charge.

Exemple

```
DROP INDEX ON Vendeurs.ID_Vendeur
```

Expressions SQL

Vous pouvez utiliser des expressions dans les clauses `WHERE`, `HAVING` et `ORDER BY` des instructions `SELECT` pour créer des requêtes de base de données complexes et détaillées. Les éléments d'expression valides sont les suivants :

- Noms de rubrique
- Constantes
- Notification en virgule flottante/scientifique
- Opérateurs numériques
- Opérateurs de caractères
- Opérateurs de dates
- Opérateurs relationnels
- Opérateurs logiques
- Fonctions

Noms de rubrique

L'expression la plus courante est un simple nom de rubrique, tel que `calc` ou `Informations_Ventes.ID_facture`.

Constantes

Les constantes sont des valeurs qui ne changent pas. Par exemple, dans l'expression `PRIX * 1.05`, la valeur 1,05 est une constante. Vous pouvez également affecter la valeur 30 à la constante `Nombre_de_jours_en_juin`.

Les valeurs de type constante doivent être placées entre guillemets simples ('). Pour inclure un guillemet simple (correspondant à une apostrophe) dans une constante de type chaîne de caractères placée entre guillemets simples, utilisez deux guillemets simples consécutifs (par exemple `'L''instance'`).

Dans les applications ODBC et JDBC, FileMaker accepte les constantes de date, d'heure et d'horodatage au format ODBC/JDBC entre accolades ({}), par exemple :

- `{D'05-06-2012'}`
- `{H'14:35:10'}`
- `{HD '05-06-2012 14:35:10'}`

FileMaker permet d'utiliser la version en majuscules ou en minuscules des spécificateurs de type (`D`, `H`, `HD`). Vous pouvez utiliser le nombre d'espaces souhaité après le spécificateur de type ou bien n'inclure aucun espace.

FileMaker accepte également les formats de date et d'heure ISO syntaxe SQL-92 sans accolades :

- `DATE 'JJ-MM-AAAA'`
- `HEURE 'HH:MM:SS'`
- `HORODATAGE 'JJ-MM-AAAA HH:MM:SS'`

La fonction EXECUTE SQL de FileMaker Pro n'accepte que les formats de date et d'heure ISO syntaxe SQL-92 sans accolades.

Constante	Syntaxe acceptable (exemples)
Texte	`'Paris'`
Nombre	`1,05`

Date	`DATE '05/06/2012'`
	`{ D '05-06-2012' }`
	`{05/06/2012}`
	`{05/06/12}`
	Remarque La syntaxe d'année à deux chiffres n'est pas prise en charge pour le format ODBC/JDBC ou le format SQL-92.

Heure	`HEURE '14:35:10'`
	`{ H '14:35:10' }`
	`{14:35:10}`

Horodatage	`HORODATAGE '05/06/2012 14:35:10'`
	`{ HD '05-06-2012 14:35:10'}`
	`{05/06/2012 14:35:10}`
	`{05/06/12 14:35:10}`
	Vérifiez que **type de données strictes : Année à 4 chiffres** n'est pas sélectionnée comme option de validation dans le fichier de base de données FileMaker pour une rubrique qui utilise cette syntaxe d'année à deux chiffres.
	Remarque La syntaxe d'année à deux chiffres n'est pas prise en charge pour le format ODBC/JDBC ou le format SQL-92.

Lors de la saisie de valeurs de date et d'heure, faites correspondre le format de la configuration locale du fichier de base de données. Par exemple, si la base de données a été créée sur un système italien, utilisez les formats italiens de date et d'heure.

Notification en virgule flottante/scientifique

Les nombres peuvent être exprimés à l'aide d'une notation scientifique.

Exemple

```
SELECT colonne1 / 3.4E+7 FROM table1 WHERE calc < 3.4E-6 * colonne2
```

Opérateurs numériques

Vous pouvez inclure les opérateurs suivants dans une expression numérique : +, -, *, /, et ^ ou ** (puissance).

Vous pouvez faire précéder des expressions numériques par un signe plus (+) ou moins (-) unaire.

Opérateurs de caractères

Vous pouvez concaténer des caractères.

Exemples

Dans les exemples suivants, `nom_famille` a pour valeur `'JONES '` et prénom a pour valeur `'ROBERT ':`

Opérateur :	Concaténation	Exemple	Résultat
+	Laisse les espaces finaux à leur place	prénom + nom_famille	'ROBERT MARTIN '
-	Déplace les espaces finaux à la fin	prénom - nom_famille	'ROBERTMARTIN '

Opérateurs de dates

Vous pouvez modifier des dates.

Exemples

Dans les exemples suivants, `date_embauche` a pour valeur `DATE '30-01-2013'.`

Opérateur :	Effet sur la date	Exemple	Résultat
+	Ajoute un nombre de jours à une date	`date_embauche` + 5	DATE '04-02-2013'
-	Recherche le nombre de jours entre deux dates	`date_embauche - DATE '01-01-2013'`	`29`
	Enlève un nombre de jours à une date	`date_embauche - 10`	DATE '20-1-2013'

Exemples supplémentaires :

```
SELECT Date_Vente, Date_Vente + 30 AS stat FROM Informations_Ventes
SELECT Date_Vente, Date_Vente - 30 AS stat FROM Informations_Ventes
```

Opérateurs relationnels

Opérateur	Signification
=	Egal
<>	N'est pas égal à
>	Supérieur à
>=	Supérieur ou égal à
<	Inférieur à
<=	Inférieur ou égal à
LIKE	Identique à une structure
NOT LIKE	Pas identique à une structure
IS NULL	A une valeur nulle
IS NOT NULL	N'a pas de valeur nulle
BETWEEN	Plage de valeurs entre une limite inférieure et une limite supérieure
IN	Membre d'un jeu de valeurs indiquées ou d'une sous-requête

Opérateur	Signification
NOT IN	Non membre d'un jeu de valeurs indiquées ou d'une sous-requête
EXISTS	Renvoie 'True' (Vrai) si une sous-requête a renvoyé au moins un enregistrement
ANY	Compare une valeur à chaque valeur renvoyée par une sous-requête (l'opérateur doit être précédé de =, <>, >, >=, <, ou <=) ; =Any est équivalent à In
ALL	Compare une valeur à chaque valeur renvoyée par une sous-requête (l'opérateur doit être précédé de =, <>, >, >=, <, ou <=)

Exemples

```
SELECT Informations_Ventes.ID_Facture FROM Informations_Ventes
   WHERE Informations_Ventes.ID_Vendeur = 'SP-1'

SELECT Informations_Ventes.Quantité FROM Informations_Ventes WHERE
Informations_Ventes.ID_Facture <> 125

SELECT Informations_Ventes.Quantité FROM Informations_Ventes WHERE
Informations_Ventes.Quantité > 3000

SELECT Informations_Ventes.Heure_Vente FROM Informations_Ventes
   WHERE Informations_Ventes.Heure_Vente < '12:00:00'

SELECT Informations_Ventes.Nom_Société FROM Informations_Ventes
   WHERE Informations_Ventes.Nom_Société LIKE '%University'

SELECT Informations_Ventes.Nom_Société FROM Informations_Ventes
   WHERE Informations_Ventes.Nom_Société NOT LIKE '%University'

SELECT Informations_Ventes.Montant FROM Informations_Ventes WHERE
Informations_Ventes.Montant IS NULL

SELECT Informations_Ventes.Montant FROM Informations_Ventes WHERE
Informations_Ventes.Montant IS NOT NULL

SELECT Informations_Ventes.ID_Facture FROM Informations_Ventes
   WHERE Informations_Ventes.ID_Facture BETWEEN 1 AND 10

SELECT COUNT(Informations_Ventes.ID_Facture) AS stat
   FROM Informations_Ventes WHERE Informations_Ventes.ID_FACTURE IN
(50,250,100)

SELECT COUNT(Informations_Ventes.ID_Facture) AS stat
   FROM Informations_Ventes WHERE Informations_Ventes.ID_FACTURE NOT IN
(50,250,100)
```

```
SELECT COUNT(Informations_Ventes.ID_Facture) AS stat FROM
Informations_Ventes
   WHERE Informations_Ventes.ID_FACTURE NOT IN (SELECT
Informations_Ventes.ID_Facture
   FROM Informations_Ventes WHERE Informations_Ventes.ID_Vendeur = 'SP-4')

SELECT *
   FROM Informations_Ventes WHERE EXISTS (SELECT
Informations_Ventes.Quantité
   FROM Informations_Ventes WHERE Informations_Ventes.ID_Vendeur IS NOT
NULL)

SELECT *
   FROM Informations_Ventes WHERE Informations_Ventes.Quantité = ANY
(SELECT Informations_Ventes.Quantité
   FROM Informations_Ventes WHERE Informations_Ventes.ID_Vendeur =
'SP-1')

SELECT *
   FROM Informations_Ventes WHERE Informations_Ventes.Quantité = ALL
(SELECT Informations_Ventes.Quantité
   FROM Informations_Ventes WHERE Informations_Ventes.ID_Vendeur
IS NULL)
```

Opérateurs logiques

Vous pouvez combiner deux conditions ou davantage. Ces conditions doivent être liées par `AND` ou `OR`, sous la forme :

```
salaire = 40000 AND exempt = 1
```

L'opérateur logique NOT sert à inverser la signification comme suit :

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

Exemples

```
SELECT * FROM Informations_Ventes WHERE Informations_Ventes.Nom_Société
   NOT LIKE '%University' AND Informations_Ventes.Montant > 3000

SELECT * FROM Informations_Ventes WHERE (Informations_Ventes.Nom_Société
   LIKE '%University' OR Informations_Ventes.Montant > 3000)
   AND Informations_Ventes.ID_Vendeur = 'SP-1'
```

Ordre de priorité des opérateurs

Au fur et à mesure que les expressions se compliquent, l'ordre dans lequel elles sont évaluées devient important. Le tableau suivant montre dans quel ordre les opérateurs sont évalués. Les opérateurs de la première ligne sont évalués en premier et ainsi de suite. Les opérateurs qui figurent sur une même ligne sont évalués de gauche à droite dans l'expression.

Ordre de priorité	Opérateur
1	Unaire `'-'`, Unaire `'+'`
2	^, **
3	*, /
4	+, -
5	=, <>, <, <=, >, >=, Like, Not Like, Is Null, Is Not Null, Between, In, Exists, Any, All
6	Not
7	ET
8	Indre

L'exemple suivant montre l'importance de l'ordre de priorité :

```
WHERE salaire > 4000 OR date_embauche > {30-01-2008} AND serv = 'D101'
```

Comme l'opérateur AND est évalué en premier, cette requête extrait les employés du service D101 embauchés après le 30.01.08, ainsi que les employés dont le salaire est supérieur à 40 000 euros quel que soit leur service ou leur date d'embauche.

Pour forcer la clause à être évaluée dans un ordre différent, utilisez des parenthèses pour spécifier les conditions à évaluer en premier. Par exemple :

```
WHERE (salaire > 40000 OR date_embauche > DATE '30-01-2008} AND
serv = 'D101'
```

extrait les employés du service D101 dont le salaire est supérieur à 40 000 euros ou qui ont été embauchés après le mercredi 30 janvier 2008

Fonctions SQL

Le langage SQL de FileMaker prend en charge un grand nombre de fonctions à utiliser dans des expressions. Certaines des fonctions renvoient des chaînes de caractères, des nombres, des dates de retour et d'autres renvoient des valeurs qui varient selon les conditions remplies par les arguments de fonctions.

Fonctions statistiques

Les fonctions statistiques renvoient une valeur unique à partir d'un jeu d'enregistrements. Vous pouvez employer une fonction statistique dans une instruction `SELECT` avec un nom de rubrique (par exemple, `AVG(SALAIRE)`) ou en combinaison avec une expression de colonne (par exemple, `AVG(SALAIRE * 1.07)`).

Vous pouvez faire précéder l'expression de colonne par l'opérateur DISTINCT pour éliminer les doublons. Par exemple :

```
COUNT (DISTINCT nom_famille)
```

Dans cet exemple, seules les valeurs de nom de famille uniques sont comptées.

Fonctions statistiques	Résultat
SUM	Total des valeurs d'une expression de rubrique de type numérique. Par exemple, SUM(SALAIRE) renvoie la somme de toutes les valeurs de la rubrique SALAIRE.
AVG	Moyenne des valeurs d'une expression de rubrique de type numérique. Par exemple, AVG(SALAIRE) renvoie la moyenne de toutes les valeurs de la rubrique SALAIRE.
COUNT	Nombre de valeurs d'une expression de rubrique. Par exemple, COUNT(NOM) renvoie le nombre de valeurs de la rubrique NOM. Utilisé avec un nom de rubrique, COUNT renvoie le nombre de valeurs non nulles de la rubrique. COUNT(*) est une forme spéciale qui renvoie le nombre d'enregistrements d'un jeu, en incluant ceux qui contiennent des valeurs nulles.
MAX	Valeur maximale d'une expression de rubrique. Par exemple, MAX(SALAIRE) renvoie la valeur maximale de la rubrique SALAIRE.
MIN	Valeur minimale d'une expression de rubrique. Par exemple, MIN(SALAIRE) renvoie la valeur minimale de la rubrique SALAIRE.

Exemples

```
SELECT SUM (Informations_Ventes.Quantité) AS stat FROM
Informations_Ventes

SELECT AVG (Informations_Ventes.Quantité) AS stat FROM
Informations_Ventes

SELECT COUNT (Informations_Ventes.Quantité) AS stat FROM
Informations_Ventes

SELECT MAX (Informations_Ventes.Quantité) AS stat FROM
Informations_Ventes
   WHERE Informations_Ventes.Quantité < 3000

SELECT MIN (Informations_Ventes.Quantité) AS stat FROM
Informations_Ventes
   WHERE Informations_Ventes.Quantité > 3000
```

Fonctions qui renvoient des chaînes de caractères

Fonctions qui renvoient des chaînes de caractères	Description	Exemple
CHR	Convertit un code ASCII en chaîne d'un caractère	CHR(67) renvoie **C**
CURRENT_USER	Renvoie l'ID de connexion indiqué pour établir la connexion	
DAYNAME	Renvoie le nom du jour correspondant à une date précise	
RTRIM	Supprime les espaces finaux d'une chaîne	RTRIM(' ABC ') renvoie ' ABC'
TRIM	Supprime les espaces de début et finaux d'une chaîne	TRIM(' ABC ') renvoie 'ABC'
LTRIM	Supprime les espaces de début d'une chaîne	LTRIM(' ABC ') renvoie 'ABC'
UPPER	Transforme chaque lettre d'une chaîne en majuscule	UPPER('Alain') renvoie 'ALAIN'
LOWER	Transforme chaque lettre d'une chaîne en minuscule	LOWER('Alain') renvoie 'alain'
LEFT	Renvoie les caractères les plus à gauche d'une chaîne	LEFT('Mathieu',3) renvoie 'Mat'
MONTHNAME	Renvoie le nom du mois calendaire	
RIGHT	Renvoie les caractères les plus à droites d'une chaîne	RIGHT('Mathieu',4) renvoie 'hieu'
SUBSTR SUBSTRING	Renvoie une sous-chaîne de chaîne avec les paramètres de la chaîne, le premier caractère à extraire et le nombre de caractères à extraire (facultatif)	SUBSTR('Conrad',2,3) renvoie 'onr' SUBSTR('Conrad',2) renvoie 'onrad'
SPACE	Génère une chaîne d'espaces	SPACE(5) renvoie ' '
STRVAL	Convertit une valeur de n'importe quel type en chaîne de caractères	STRVAL('Woltman') renvoie 'Woltman' STRVAL(5 * 3) renvoie '15' STRVAL(4 = 5) renvoie 'False' STRVAL(DATE '25-12-2008') renvoie '25-12-2008'
TIME TIMEVAL	Renvoie l'heure du jour sous forme de chaîne	A 21 h 49, TIME() renvoie **21:49:00**
USERNAME USER	Renvoie l'ID de connexion indiqué pour établir la connexion	

Remarque La fonction TIME() n'est plus utilisée. Utilisez le standard SQL CURRENT_TIME pour la remplacer.

Exemples

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

SELECT RTRIM(' ' + Vendeurs.ID_Vendeur) AS agg FROM Vendeurs

SELECT TRIM(SPACE(1) + Vendeurs.ID_Vendeur) AS stat FROM Vendeurs

SELECT LTRIM(' ' + Vendeurs.ID_Vendeur) AS agg FROM Vendeurs

SELECT UPPER(Vendeurs.Vendeur) AS stat FROM Vendeurs

SELECT LOWER(Vendeurs.Vendeur) AS stat FROM Vendeurs

SELECT LEFT(Vendeurs.Vendeur, 5) AS stat FROM Vendeurs

SELECT RIGHT(Vendeurs.Vendeur, 7) AS stat FROM Vendeurs

SELECT SUBSTR(Vendeurs.ID_Vendeur, 2, 2) + SUBSTR(Vendeurs.ID_Vendeur,
4, 2) AS stat FROM Vendeurs

SELECT SUBSTR(Vendeurs.ID_Vendeur, 2) + SUBSTR(Vendeurs.ID_Vendeur,
4) AS stat FROM Vendeurs

SELECT SPACE(2) + Vendeurs.ID_Vendeur AS ID_Vendeur FROM Vendeurs

SELECT STRVAL('60506') AS stat FROM Informations_Ventes WHERE
Informations_Ventes.Facture =1
```

Fonctions qui renvoient des nombres

Fonctions qui renvoient des nombres	Description	Exemple
ABS	Renvoie la valeur absolue de l'expression numérique	
ATAN	Renvoie la tangente inverse de l'argument sous la forme d'un angle exprimé en radians	
ATAN2	Renvoie la tangente inverse des coordonnées x et y sous la forme d'un angle exprimé en radians	
CEIL CEILING	Renvoie le plus petit entier supérieur ou égal à l'argument	
DEG DEGREES	Renvoie le nombre de degrés de l'argument, qui est un angle exprimé en radians	
DAY	Renvoie la partie d'une date correspondant au jour	`DAY(DATE '30-01-2012')` renvoie **30**
DAYOFWEEK	Renvoie le jour de la semaine (de 1 à 7) d'une expression de date	`DAYOFWEEK(DATE '01-05-2004')` renvoie **7**
MOD	Divise deux nombres et renvoie le reste de la division	`MOD(10,3)` renvoie **1**

Fonctions qui renvoient des nombres	Description	Exemple
EXP	Renvoie une valeur qui est la base du logarithme népérien (e) élevé à la puissance spécifiée par l'argument	
FLOOR	Renvoie le plus grand entier inférieur ou égal à l'argument	
HOUR	Renvoie la partie d'une valeur correspondant à l'heure	
INT	Renvoie la partie entière d'un nombre	`INT(6.4321)` renvoie **6**
LENGTH	Renvoie la longueur d'une chaîne	`LENGTH('ABC')` renvoie **3**
MONTH	Renvoie la partie d'une date correspondant au mois	`MONTH(DATE '30-01-2012')` renvoie **1**
LN	Renvoie le logarithme népérien de l'argument	
LOG	Renvoie le logarithme courant de l'argument	
MAX	Renvoie le plus grand de deux nombres	`MAX(66,89)` renvoie **89**
MIN	Renvoie le plus petit de deux nombres	`MIN(66,89)` renvoie **66**
MINUTE	Renvoie la partie d'une valeur correspondant aux minutes	
NUMVAL	Convertit une chaîne de caractères en nombre. La fonction échoue si la chaîne de caractère n'est pas un nombre valide.	`NUMVAL('123')` renvoie **123**
PI	Renvoie la valeur de constante de la constante mathématique pi	
RADIANS	Renvoie le nombre de radians pour un argument exprimé en degrés	
ROUND	Arrondit un nombre	`ROUND(123.456,0)` renvoie **123** `ROUND(123.456,2)` renvoie **123,46** `ROUND(123.456,- 2)` renvoie **100**
SECOND	Renvoie la partie d'une valeur correspondant aux secondes	
SIGN	Indicateur du signe de l'argument : - 1 pour négatif, 0 pour 0 et 1 pour positif	
SIN	Renvoie le sinus de l'argument	
SQRT	Renvoie la racine carrée de l'argument	
TAN	Renvoie la tangente de l'argument	
YEAR	Renvoie la partie d'une date correspondant à l'année	`YEAR(DATE '30-01-2013')` renvoie **2013**

Fonctions qui renvoient des dates

Fonctions qui renvoient des dates	Description	Exemple
`CURDATE` `CURRENT_DATE`	Renvoie la date du jour	
`CURTIME` `CURRENT_TIME`	Renvoie l'heure actuelle	
`CURTIMESTAMP` `CURRENT_TIMESTAMP`	Renvoie la valeur d'horodatage actuelle	
`TIMESTAMPVAL`	Convertit une chaîne de caractères en horodatage	`TIMESTAMPVAL('30-01-2013 14:00:00')` renvoie sa valeur d'horodatage.
`DATE` `TODAY`	Renvoie la date du jour	Si la date du jour est le 21/11/2013, `DATE()` renvoie **21-11-2013**
`DATEVAL`	Convertit une chaîne de caractères en date	`DATEVAL('30-01-2013')` renvoie **30-01-2013**

Remarque La fonction `DATE()` n'est plus utilisée. Utilisez le standard SQL CURRENT_DATE pour la remplacer.

Fonctions conditionnelles

Fonctions conditionnelles	Description	Exemple
CASE WHEN	**Format CASE simple** Compare la valeur de *exp_entrée* aux valeurs des arguments *exp_valeur* pour déterminer le résultat. CASE *exp_entrée* {WHEN *exp_valeur* THEN *résultat*...} [ELSE *résultat*] END	SELECT ID_Facture, CASE Nom_Société WHEN 'Exportations Royaume-Uni' THEN 'Exportations Royaume-Uni trouvées' WHEN 'Fournisseurs d'ameublement' THEN 'Fournisseurs d'ameublement trouvés' ELSE 'Aucune exportation Royaume-Uni ou fournisseur d'ameublement' END, ID_Vendeur FROM Informations_Ventes
	Format CASE recherché Renvoie un résultat selon que la condition spécifiée par une expression WHEN affiche la valeur True (Vrai). CASE {WHEN *exp_boléen* THEN *résultat*...} [ELSE *résultat*] END	SELECT ID_Facture, Montant, CASE WHEN Montant > 3000 THEN 'Au-dessus de 3000' WHEN Montant > 1000 THEN 'En-dessus de 3000' ELSE 'Entre 1000 et 3000' END, ID_Vendeur FROM Informations_Ventes
COALESCE	Renvoie la première valeur qui n'est pas nulle	SELECT ID_Vendeur, COALESCE(Directeur_Ventes, Vendeur) FROM Vendeurs
NULLIF	Compare deux valeurs et renvoie NULL si ces deux valeurs sont égales ; sinon, renvoie la première valeur.	SELECT ID_Facture, NULLIF(Montant, - 1), ID_Vendeur FROM Informations_Ventes

Mots-clés SQL réservés

Cette section répertorie les mots-clés réservés qui ne doivent pas être utilisés comme noms de colonnes, de tables, d'alias ou d'autres objets définis par l'utilisateur. Si vous obtenez des erreurs de syntaxe, ces erreurs peuvent être dues à l'utilisation de l'un de ces mots-clés réservés. Si vous souhaitez utiliser l'un d'entre eux, utilisez des guillemets pour éviter que le mot ne soit traité comme un mot-clé.

Par exemple, l'instruction `CREATE TABLE` suivante montre comment utiliser le mot-clé `DEC` comme nom d'élément de données.

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

ABSOLUTE	CHAR	CURTIME
ACTION	CHARACTER	CURTIMESTAMP
ADD	CHARACTER_LENGTH	DATE
ALL	CHAR_LENGTH	DATEVAL
ALLOCATE	CHECK	DAY
ALTER	CHR	DAYNAME
AND	CLOSE	DAYOFWEEK
ANY	COALESCE	DEALLOCATE
ARE	COLLATE	DEC
AS	COLLATION	DECIMAL
ASC	COLUMN	DECLARE
ASSERTION	COMMIT	DEFAULT
AT	CONNECT	DEFERRABLE
AUTHORIZATION	CONNECTION	DEFERRED
AVG	CONSTRAINT	DELETE
BEGIN	CONSTRAINTS	DESC
BETWEEN	CONTINUE	DESCRIBE
BINARY	CONVERT	DESCRIPTOR
BIT	CORRESPONDING	DIAGNOSTICS
BIT_LENGTH	COUNT	DISCONNECT
BLOB	CREATE	DISTINCT
BOOLEAN	CROSS	DOMAIN
BOTH	CURDATE	DOUBLE
BY	CURRENT	DROP
CASCADE	CURRENT_DATE	ELSE
CASCADED	CURRENT_TIME	END
CASE	CURRENT_TIMESTAMP	END_EXEC
CAST	CURRENT_USER	ESCAPE
CATALOG	CURSOR	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
NSERT
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
UNION
UNIQUE
UNKNOWN
UPDATE
UPPER
USAGE
USER
USERNAME
USING
VALUE
VALUES
VARBINARY
VARCHAR
VARYING
VIEW
WHEN
WHENEVER
WHERE
WITH
WORK
WRITE
YEAR
ZONE

Index

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W