

Les nouveautés de Microsoft Access 2010



par Christophe WARIN (<http://warin.developpez.com>)

Date de publication : 01 juillet 2009

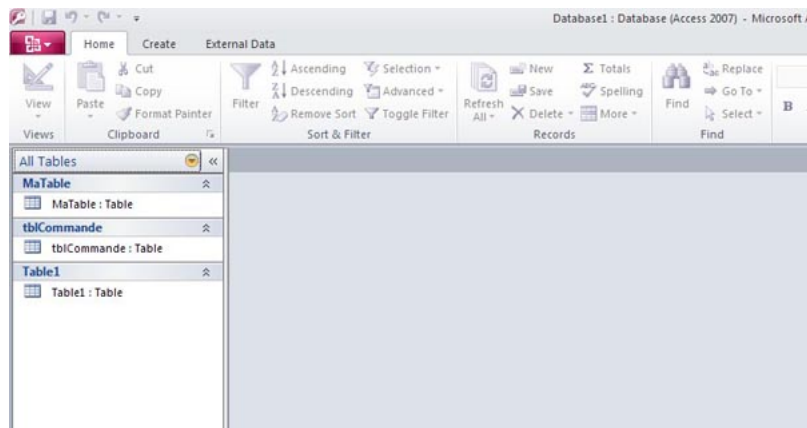
Dernière mise à jour :

Alors que la version Béta de Microsoft Office 2010 est sur le point de sortir nous vous offrons un petit aperçu de ce qui sera, demain, Microsoft Access 2010.

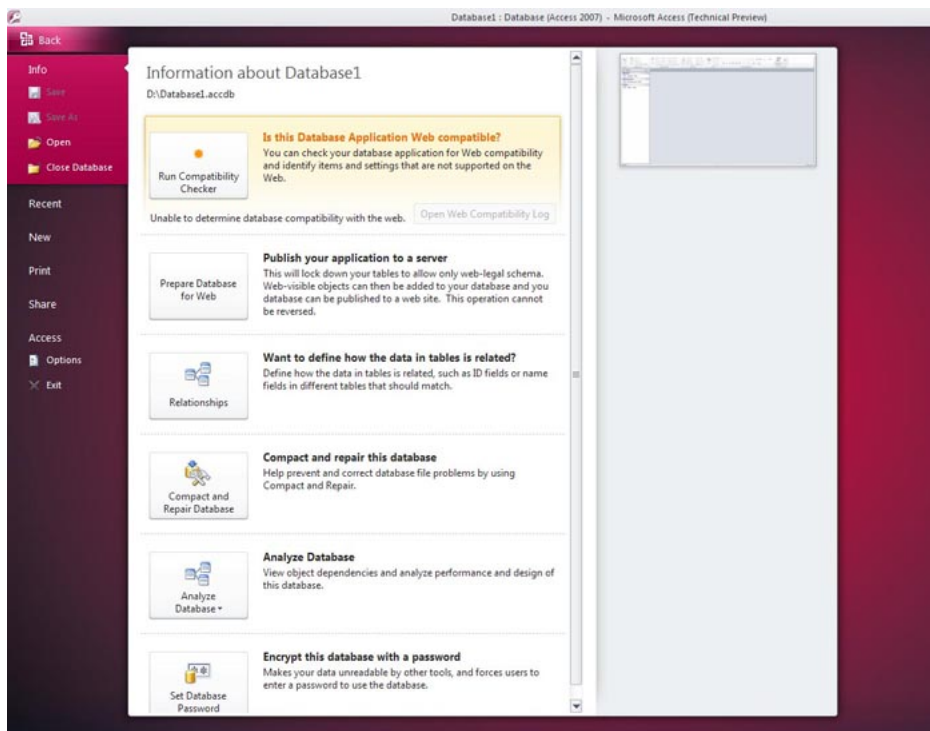
I - Le ruban.....	3
II - Les triggers.....	5
II-A - Les étapes à suivre et les blocs d'instructions.....	5
II-B - Création de la macro.....	6
II-C - Conclusion.....	9
III - La mise en forme conditionnelle.....	9
IV - Le générateur d'expressions.....	12
V - Les champs calculés.....	13
V-A - Introduction.....	13
V-B - Cas pratique.....	13
V-C - Banc d'essai.....	15
V-C-1 - Conclusion.....	16

I - Le ruban

A première vue, lorsqu'on lance Microsoft Access 2010, on a l'impression que le ruban n'a pas vraiment changé si ce n'est sa couleur. Il faut vraiment regarder en détails pour s'apercevoir des quelques changements qui, pour certains, vont vous changer la vie de développeur.



Le premier changement que l'on remarque se situe en haut à gauche où le logo aux couleurs d'Access vient remplacer le bouton commun à toutes les applications Office. En effet, désormais, chaque application dispose de son propre bouton "Fichier". C'est ainsi que Word s'offre un bouton bleu, Excel, un bouton vert, etc. Rien de bien extraordinaire, si on s'arrête à cela. Mais comme moi, je pense que lors du premier clic sur ce bouton vous serez surpris par la fenêtre qui va s'afficher sous vos yeux. Jugez plutôt :

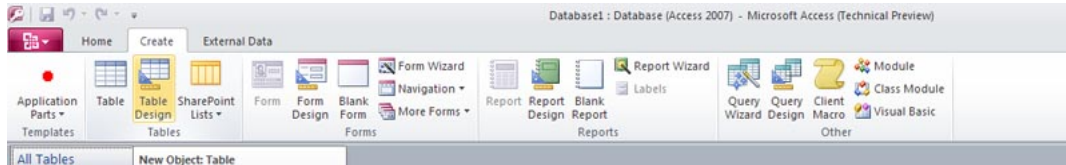


De toute évidence, et sans vouloir rentrer davantage dans les détails, on peut constater que toutes les opérations qui étaient hier présentes dans l'onglet **Outil de base de données** ont été relayées à ce nouveau menu. Ce choix paraît plutôt judicieux étant donné la faible fréquentation de ces outils. Le ruban est ainsi allégé d'un onglet, les débutants auront donc une zone de moins où se perdrent.

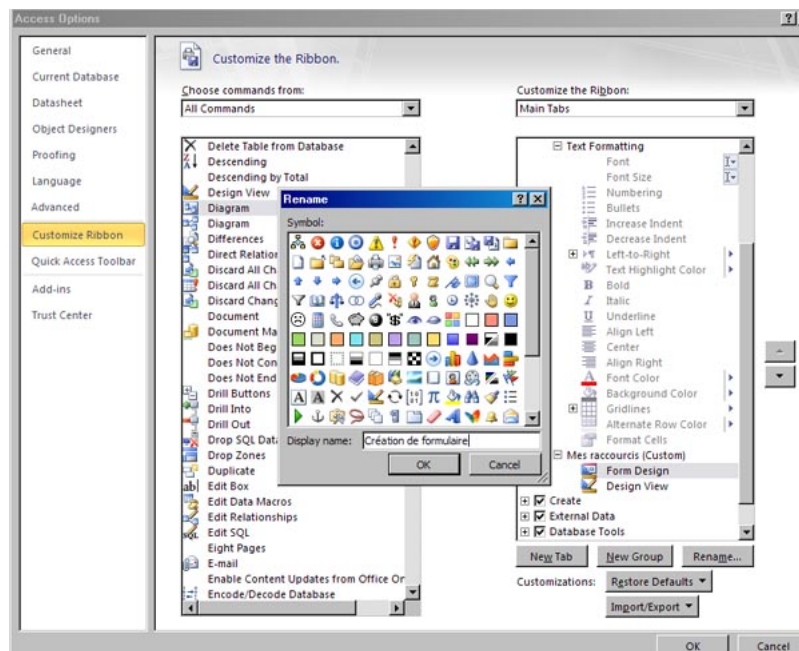
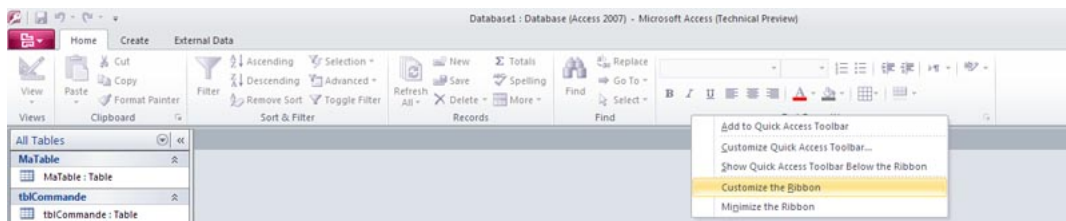
Autre petit plus du nouveau ruban :



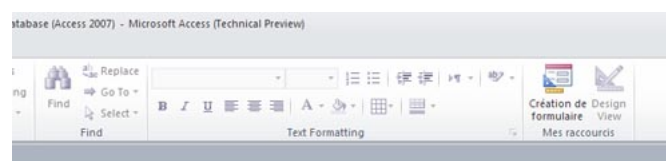
Une petite flèche située en haut à droite offre la possibilité de réduire le ruban. Je vous rassure, cette fonctionnalité est encore disponible dans le menu contextuel.



Mais la révolution, la vraie, celle que tout le monde attendait avec impatience, c'est bien la possibilité de modifier le ruban de l'interface de développement. En quelques clics vous allez pouvoir agrémenter votre ruban à votre façon, y placer les raccourcis dont vous avez besoin, créer vos onglets, regrouper des outils ...



Il est même possible de renommer et de changer l'icône des boutons.



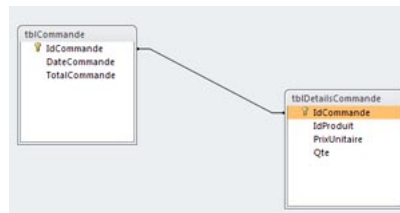
Attention, les modifications sont apportées à l'application Access et non à la base de données courante. Il ne s'agit pas de remplacer la programmation du ruban pour les utilisateurs mais de paramétrer votre environnement de développement afin d'accroître votre productivité. Avoir par exemple, comme moi, les boutons création de formulaire et création de table l'un à côté de l'autre, tout rassembler sur un seul onglet, etc. Le but étant de perdre le moins de temps possible à chercher les fonctionnalités. C'était la vocation du ruban d'Access 2007, Access 2010 conserve et améliore l'idée.

En conclusion, si vous avez aimé le ruban d'Access 2007, vous allez adorer celui d'Access 2010, et je vous invite dans quelques jours à découvrir une nouvelle fonctionnalité d'Access 2010 et plus particulièrement les Triggers, eux aussi tant attendus.

N'hésitez pas à donner votre avis sur les nouvelles fonctionnalités d'Access 2010 :

II - Les triggers

Pour aborder le sujet des Triggers (déclencheurs) dans Microsoft Access 2010, nous allons utiliser l'exemple suivant :



Un trigger est un ensemble d'actions exécutées par le moteur de base de données suite à un événement provenant d'une table. Il peut s'agir d'une insertion, d'une suppression ou d'une modification des données.

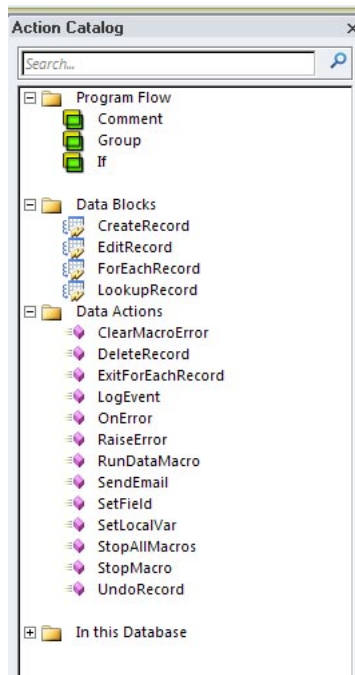
Dans cet exemple, nous allons montrer comment mettre à jour le champ **TotalCommande** de la table **tblCommande** à chaque fois qu'une nouvelle ligne est entrée dans la table **tblDetailsCommande**.

II-A - Les étapes à suivre et les blocs d'instructions

Tout d'abord, avant de se lancer à la création de la macro correspondante, il est nécessaire d'établir le chemin à emprunter. Il se compose de plusieurs étapes :

- 1 Mémoriser le numéro de la commande concernée
- 2 Parcourir toutes les lignes de **tblDetailsCommande** pour en mémoriser le prix
- 3 Faire le cumul de ces lignes
- 4 Rechercher l'enregistrement correspondant dans la table **tblCommande**
- 5 Mettre à jour le champ **TotalCommande**

Pour cela, Microsoft Access intègre un nouveau générateur de macro permettant de créer des blocs d'instructions régis par une condition ou bien encore une boucle. Ainsi le bloc **ForEachRecord** va permettre de parcourir les lignes d'une table, **LookupRecord** de rechercher un enregistrement, etc.

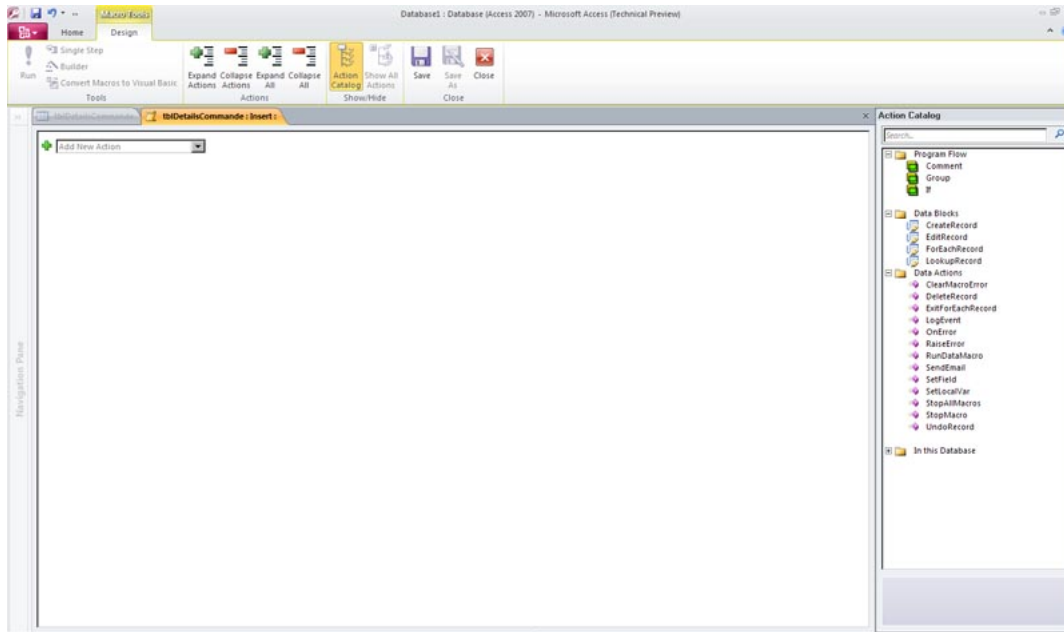


Il est important de garder en tête que lorsque vous placez une instruction dans un bloc vous ne pouvez atteindre que les données issues de ce bloc. Cette notion est peu commune mais un exemple saura vous éclairer : en début d'une macro attribuée par exemple à la table **tblDetailsCommande**, si vous faites appel au champ **[idCommande]** c'est celui de la table **tblDetailsCommande** qui sera concerné, en revanche si vous ouvrez par la suite un bloc **LookUpRecord** recherchant une information dans la table **tblCommande**, tout appel à **[idCommande]** dans ce bloc fera référence au champ **[idcommande]** de la table **tblCommande**. Etre vigilant à l'ordre des méthodes est donc primordial pour garantir que vous manipulez les bonnes données.

II-B - Création de la macro

Pour créer le déclencheur, rendez-vous sur la table **tblDetailsCommande** en mode *Création* et éditez sa macro nommée **After Insert** (après insertion).

L'éditeur suivant s'affiche :



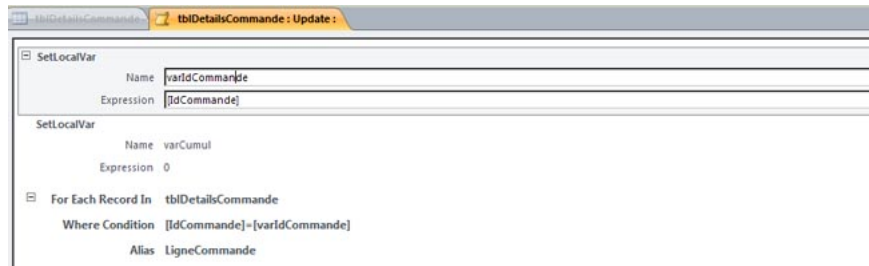
Comme nous l'avons indiqué dans la section précédente, la première étape consiste à mémoriser l'identifiant de la commande concernée. Pour cela, le générateur de macro intègre les **LocalVar** qui sont des variables permettant de stocker des données le temps de l'exécution de la macro. La méthode **SetLocalVar** permet d'en définir la valeur. Pour accéder à une variable mémorisée précédemment, il suffit de l'appeler par son nom, comme cela aurait été fait pour un champ. La variable recevant le numéro de commande sera nommée **varIdCommande**, elle sera donc accessible via la syntaxe : **[varIdCommande]**



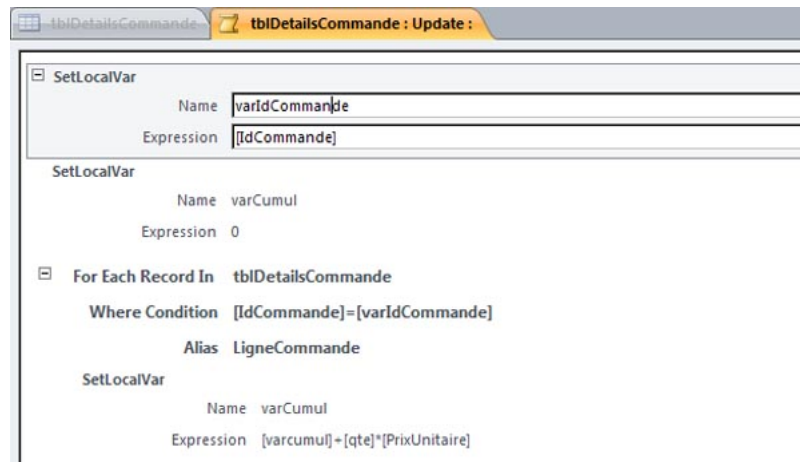
Comme vous pouvez le constater sur l'image ci-dessus, il n'y a pas de syntaxe spéciale pour définir que vous faites référence à l'enregistrement en cours. En fait, dès que vous appelez un champ de la table porteuse du trigger, la valeur retournée correspond à celle disponible dans l'enregistrement qui vient de lever l'évènement.

Dans le but de stocker le montant total de la commande au cours du traitement, il est nécessaire de créer une nouvelle variable, nous la nommerons **varCumul**. Sa valeur sera fixée à 0 au début de l'exécution, de nouveau via **SetLocalVar**.

Afin de remplir cette variable de cumul, il est nécessaire ensuite de parcourir tous les enregistrements qui ont le même numéro de commande que celui en cours. La méthode **ForEachRecord** permet cela. Son premier argument représente la table à parcourir, le deuxième le critère de recherche servant à filtrer les données et enfin l'argument **Alias** est le nom avec lequel il sera possible de faire référence à l'enregistrement. Alias est un paramètre que l'on retrouve dans plusieurs blocs de macros. Bien que facultatif, comme pour toute variable, une déclaration ayant du sens ne peut que vous servir.

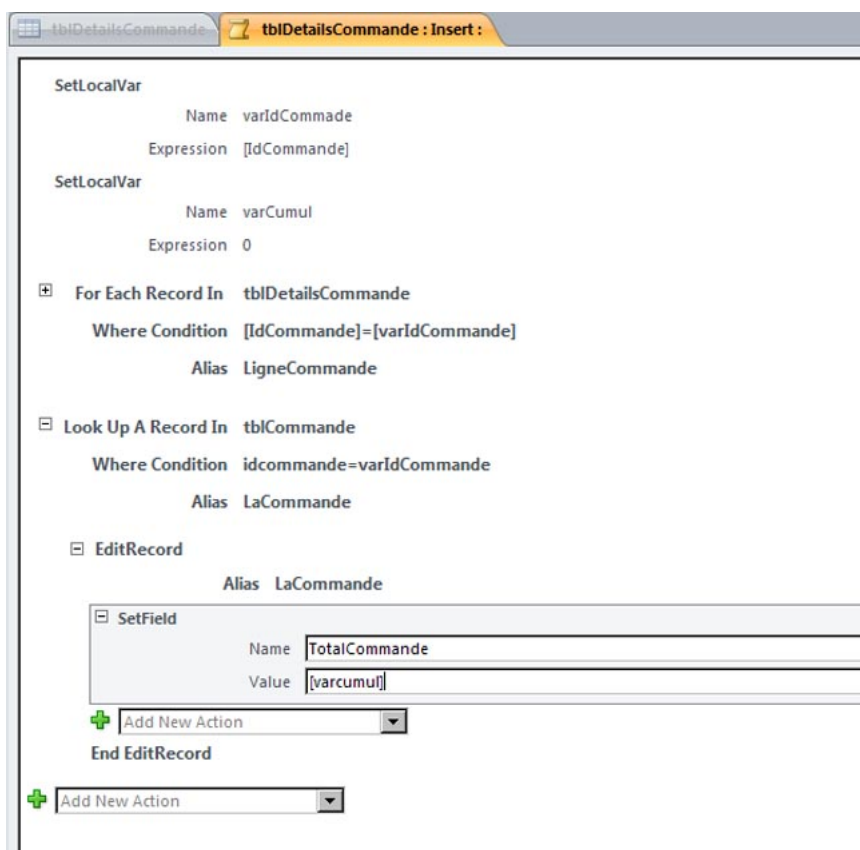


Pour chaque ligne trouvée par le bloc **ForEachRecord** nous allons mettre à jour la variable de cumul en faisant de nouveau appel à **SetLocalVar**. La nouvelle valeur sera : $[varcumul] + qte * PrixUnitaire$:



Etant donné qu'aucune action supplémentaire ne viendra s'inscrire dans le bloc **ForEachRecord**, il est conseillé de le replier en cliquant sur le sigle - à sa gauche afin de limiter les erreurs de saisie (il faut bien avouer, que pour l'instant, la saisie des macros est un peu décourageante)

La méthode **LookUpRecord** permet ensuite de rechercher la commande dans la table **tblCommande**. Le critère sera bien entendu basé sur la variable **varIdCommande**. Pour pouvoir éditer l'enregistrement retourné, il est nécessaire de faire appel au bloc **EditRecord**, un peu comme vous l'auriez fait avec la méthode **Edit** d'un recordset. Ici, l'alias est vraiment important faute de quoi le moteur ne saura pas quel jeu de données vous voulez éditer. Pour terminer, la méthode **SetField** dans le bloc **EditRecord** permet de mettre à jour la valeur du champ **TotalCommande** pour l'enregistrement trouvé. La gestion par bloc d'instructions est un régal lorsque l'on imbrique les méthodes comme ici. En effet, celles-ci s'exécutent à la condition que le bloc soit opérationnel. En d'autres termes, et contrairement à VBA, le développeur n'a pas à s'affranchir des tests d'exécution tels que *"l'enregistrement a été trouvé"*, *"l'enregistrement est éditible"* ...



Il ne reste plus qu'à enregistrer et à tester. Le résultat est bien celui attendu puisque toute insertion dans la table **tblDetailsCommande** met à jour la table **tblCommande**. Bien entendu, pour être cohérent, il est nécessaire d'appliquer le même calcul sur les événements **After Update** et **After Delete**.

II-C - Conclusion

Voilà une fonctionnalité qui a été longtemps dénoncée par son absence. Malheureusement, si la vue du bouton **Create Table Events** m'a vraiment réjoui, la découverte de la fonctionnalité me laisse un goût amer. Je ne pense pas que c'est à cela que pensaient ceux qui l'ont tant réclamée. Il faut se contenter de macro là où du code SQL aurait sûrement été plus pratique et plus performant. La volonté d'offrir les événements sur table aux novices a pris le dessus même s'il s'agit d'un non-sens selon moi. Quel débutant sous Access se sent réellement concerné par les problèmes de triggers, contraintes, intégrités, etc. ? Cela aura au moins eu le mérite de me faire écrire ma première macro, et ce ne sera certainement pas la dernière tant cette fonctionnalité de triggers est importante.

N'hésitez pas à donner votre avis sur les nouvelles fonctionnalités d'Access 2010 :

III - La mise en forme conditionnelle

Datant de Microsoft Access 2000, la mise en forme conditionnelle des données n'a pas subi la moindre évolution jusqu'à ce jour. Access 2010 va inverser la tendance en proposant un nouvel assistant de mise en forme mais surtout en augmentant considérablement la quantité de critères à apprécier.

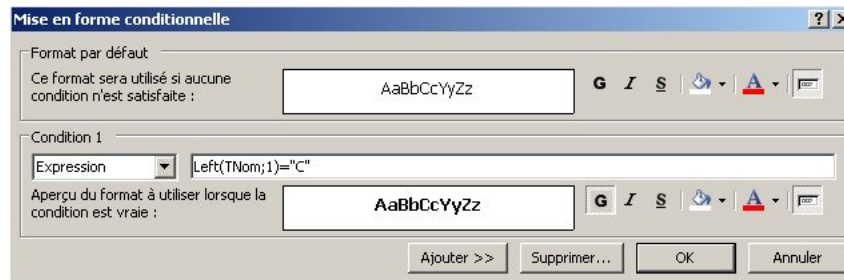
Si vous ne connaissez pas encore la mise en forme conditionnelle, vous pouvez consulter le document suivant :



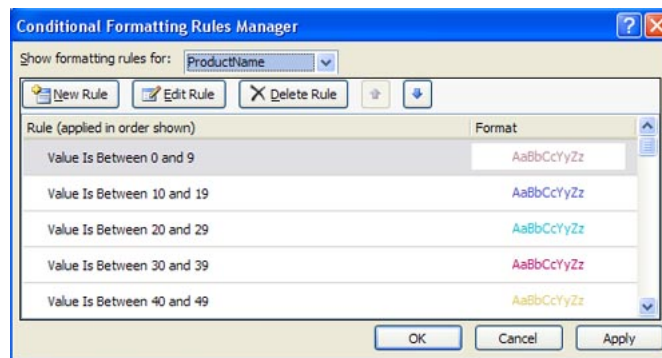
La mise en fore conditionnelle dans Access

Evidemment, l'ancienne interface limitée à trois expressions ne pouvait guère afficher plus d'éléments. L'interface a donc été repensée afin d'offrir la liste des mise en formes appliquées à un contrôle.

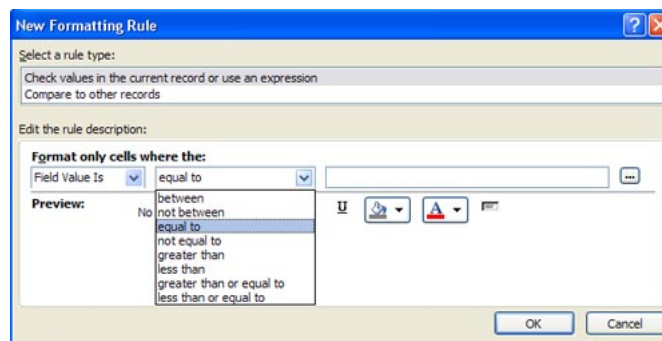
Access 2007 :



Access 2010 :



Bien que très simpliste, l'effet est optimal. La légèreté de l'interface comparée à la précédente permet d'être plus attentif aux différentes options, cela ne pourra que ravir les utilisateurs débutants sans compter le nombre de possibilités : 49 mises en forme sont paramétrables, autant dire que tous les cas seront représentés !



Passé outre ce changement d'interface, la mise en forme conditionnelle intègre une nouvelle fonctionnalité bien pratique permettant de représenter une barre de progression, un peu comme sous Excel.

Pour cela, il suffit de sélectionner le type de mise en forme conditionnelle nommé **Compare to other Record** (*Comparer aux autres enregistrements*) et de définir la borne minimale à partir de laquelle vous souhaitez que la barre démarre et la borne de fin :

Bien entendu, les objets VBA **FormatCondition** ainsi que leurs énumérations ont été complétés afin de prendre en compte cette nouvelle fonctionnalité. L'exemple suivant montre comment créer une nouvelle mise en forme conditionnelle en VBA :

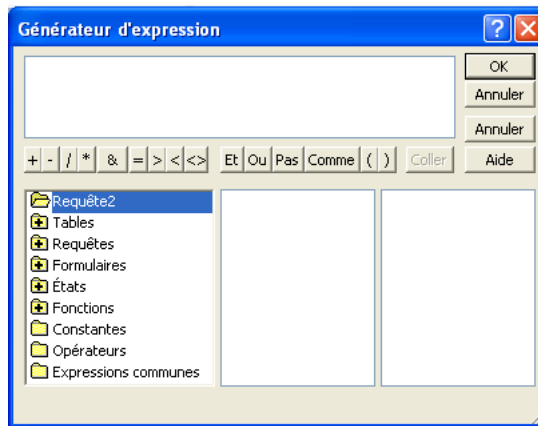
```
With TUnitPrice.FormatConditions
    .Add acDataBar, , 0
    With .Item(0)
        .LongestBarLimit = acPercent
        .LongestBarValue = 100
        .ShortestBarLimit = acPercent
        .ShortestBarValue = 0
        .BackColor = vbGreen
    End With
End With
```

Pour conclure, rien de bien révolutionnaire mais une chose est sûre, personne n'a compris pourquoi, jusqu'à cette version, il n'était pas possible de créer plus de trois mises en forme conditionnelle, à ce titre il faudra aussi nous expliquer pourquoi 49 dans Access 2010 et pourquoi pas 50 ou 255 comme la plupart des limites...

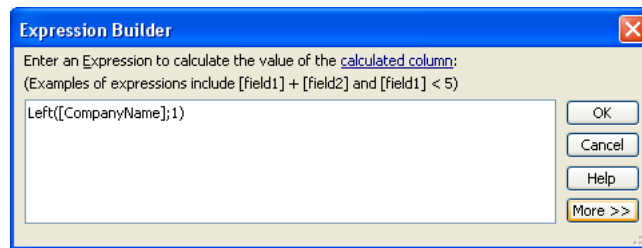
N'hésitez pas à donner votre avis sur les nouvelles fonctionnalités d'Access 2010 en rejoignant le forum

IV - Le générateur d'expressions

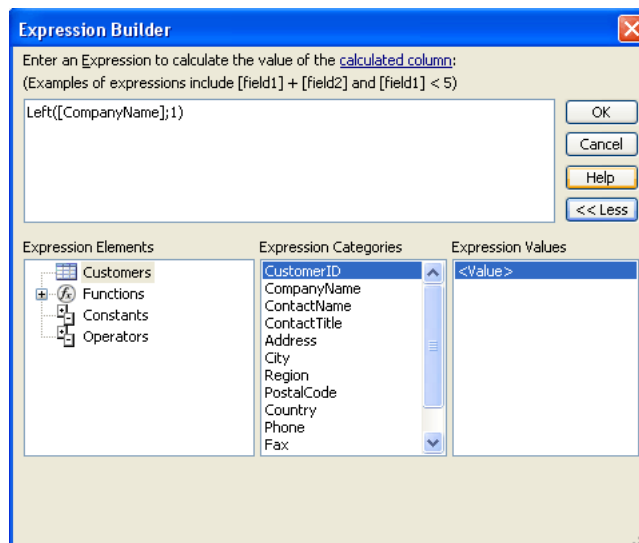
Parmi les nouveautés ou plutôt les mises à jour, il y en a qui ne réclament vraiment pas beaucoup de lignes pour les présenter. Le générateur d'expressions, c'est un peu le petit frère de la fenêtre de gestion des polices sous Windows : bien qu'on soit en 2009, le look est encore très proche de Windows 3.11. Je ne pense pas vraiment prendre de risque à affirmer que le générateur d'expressions n'a pas subi la moindre modification depuis Access 2.0 au regard de son design obsolète.



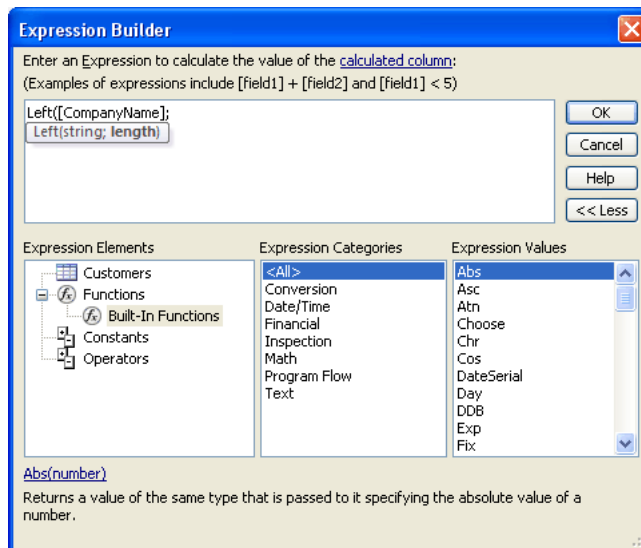
Avec Access 2010, l'heure du lifting est arrivée si bien que le générateur d'expressions en renaît :



Comme vous pouvez le voir sur l'image ci-dessus, la fenêtre est épurée et la lisibilité de la police semble améliorée. Un bouton latéral permet d'afficher ou masquer la liste des fonctions et des opérateurs disponibles dans l'unique but de ne pas alourdir l'interface inutilement.



Mais surtout, qui ne s'est jamais perdu dans la rédaction de ses expressions ? Confusion des séparateurs de paramètres, noms de paramètres laissés en lieu et place de leurs valeurs, les raisons étaient nombreuses et les ratés fréquents. Demain, il n'y aura plus aucun prétexte pour se justifier de ces erreurs : l'autocomplétion quitte enfin l'isolement de l'éditeur Visual Basic pour rejoindre le générateur d'expressions. Si on en juge par l'image ci-dessous : le résultat est assez convaincant.



Une fonctionnalité insignifiante pour certains, une révolution pour d'autres ? Personnellement, je dois avouer que j'adore la nouvelle interface et plus particulièrement mes yeux qui sont soulagés au point que j'en étais venu, parfois, à copier/coller mes expressions dans le bloc-notes.

V - Les champs calculés

V-A - Introduction

Dans un précédent message sur les forums de developpez.com, Maxence Hubiche annonçait l'arrivée des champs calculés au sein des tables de Microsoft Access 2010. Bien que la méthodologie Merise préconise d'exclure les données calculées des schémas, les SGBD évolués tels que SQL Server ont introduit des fonctionnalités permettant le stockage des calculs dans les tables afin de soulager les requêtes lorsque les opérations sont trop complexes ou fréquemment évaluées. A travers quelques tests nous vous proposons d'évaluer la pertinence de cette nouvelle fonctionnalité débarquant dans Microsoft Access.

V-B - Cas pratique

Pour illustrer nos propos, nous allons travailler sur l'exemple d'une table rassemblant les lignes des commandes d'une société de vente de produits ménager.

Voici ci-dessous la structure habituelle pour gérer ces données :

tblLigneCommande		
	Field Name	Data Type
?	IDCommande	Number
?	IDProduit	Number
	Qte	Number
	PrixHT	Currency

Le calcul du prix total d'une ligne de commande (Prix HT * Qte) est alors obtenu par la requête suivante :

```
SELECT
tblLigneCommande.IDCommande,
tblLigneCommande.IDProduit,
tblLigneCommande.Qte,
tblLigneCommande.PrixHT,
[Qte]*[PrixHT] AS TotalLigne
FROM tblLigneCommande;
```

IDCommande	IDProduit	Qte	PrixHT	TotalLigne
1	1	2	10,00 €	20,00 €
1	2	1	13,00 €	13,00 €
1	3	10	1,00 €	10,00 €
1	4	2	12,00 €	24,00 €

Sous Microsoft Access 2010, un nouveau type de champ fait son apparition, il s'agit du champ calculé destiné à stocker le calcul dans la table, voici la nouvelle implémentation :

Field Name	Data Type
IDCommande	Number
IDProduit	Number
Qte	Number
PrixHT	Currency
TotalLigne	Calculated

Text

Memo

Number

Date/Time

Currency

AutoNumber

Yes/No

OLE Object

Hyperlink

Attachment

Calculated

Lookup Wizard...

General	Lookup
Expression	[Qte]*[PrixHT]
Result Type	Currency
Format	
Decimal Places	Auto
Caption	
Smart Tags	
Text Align	General

La propriété **Expression** contient la règle de calcul à appliquer. Inutile de se soucier de **Result Type** (type du champ), Access adoptera le type adéquat en fonction des champs et des fonctions utilisés pour les calculs.

La requête d'affichage des lignes de commande se trouve donc allégée :

```
SELECT *
FROM tblLigneCommande;
```

IDCommanc	IDProduit	Qte	PrixHT	TotalLigne
1	1	2	10,00 €	20,00 €
1	2	1	13,00 €	13,00 €
1	3	10	1,00 €	10,00 €
1	4	2	12,00 €	24,00 €
*				

Si vous utilisez DAO pour créer vos champs, le nouveau type est défini via deux éléments :

- Le type passé à la méthode **CreateField** définit le type de retour attendu (texte, entier, etc.)
- La propriété **Expression** correspond à la règle de calcul à appliquer

Exemple :

```
Dim oFld As DAO.Field
Dim oTbl As DAO.TableDef

Set oFld = oTbl.CreateField("TotalLigne", dbCurrency)
oFld.Properties("Expression").Value = "[Qte]*[PrixHT]"
```

V-C - Banc d'essai

Bien entendu, un tel stockage n'est pas sans incidence et nous avons tenté d'en déterminer les répercussions sur l'ensemble de la base de données.

La première question posée consistait à savoir si la donnée était effectivement stockée ou bien seulement son opération mathématique. Un test avec 10000 enregistrements amène le constat suivant :

- Taille de la base de données avec la table **tblLigneCommande** vide : **412 Ko**
- Taille de la base de données avec la table **tblLigneCommande** remplie et possédant un champ **LigneCommande de type Currency** avec la valeur stockée : **912 Ko**
- Taille de la base de données avec la table **tblLigneCommande** remplie et possédant un champ **LigneCommande de type Calculated** : **1284 Ko**

Les tests ont été répétés des dizaines de fois avec compactage entre chaque étape. La différence de volume entre les deux solutions est difficilement explicable et dans le cas présent, le champ calculé occupe à lui seul près de 45% du volume total de la table **tblLigneCommande**. Ne pouvant réellement pas comprendre le mode de stockage de ce nouveau champ, nous nous sommes intéressés dans un second temps aux performances. L'insertion via recordset de 10000 enregistrements dans la table **tblLigneCommande** dépourvue de champ calculé est exécutée en 9.04 secondes (moyenne sur 10 essais) alors que les mêmes tests avec un champ calculé affiche une moyenne de 13.06 secondes (toujours en 10 essais).

D'autre part, sur ces 10000 enregistrements, il n'y a pas de différence significative entre les deux requêtes données plus haut correspondant toutes les deux à des structures différentes de la table **tblLigneCommande**. Sans champ calculé, la requête est exécutée 1000 fois en 7,94 secondes contre 7,92 dans le second cas.

Bien entendu, la règle de calcul utilisée ici est très simple, ce qui pourrait justifier des écarts aussi minimes et donc cacher l'avantage des champs calculés. Nous pourrions répondre par l'affirmative si toutefois les champs calculés n'avaient pas une limitation majeure : à l'heure actuelle, il est impossible de faire référence à d'autres enregistrements que celui pointé par le calcul. En d'autres termes, impossible de faire de moyenne, de sommes, de sous requêtes, ni même d'appel à des fonctions VBA personnalisées.

V-C-1 - Conclusion

Pour conclure, et en l'état de l'avancement du développement d'Office 2010, c'est-à-dire sur la *Technical Preview*, les champs calculés apparaissent plus comme un handicap qui va distraire le débutant en dé normalisant inutilement sa base de données rendant la maintenance et l'évolution vers d'autres plateformes encore plus difficile. Le développeur préférera sûrement cataloguer cette fonctionnalité avec les champs multi-valués d'Access 2007.