بسم الله الرحمن الرحيم

الحمد لله... اللهم صل على محمد وعلى آل محمد كما صليت على إبراهيم، وبارك على محمد وعلى آل محمد كما باركت على إبراهيم، في العالمين إنك حميد مجيد.

مقدمة إلى لغة SQL

هذه المادة هي أصلاً القسم الثاني من كتاب أصله هو الجزء الأول من سلسلة من الحلقات في منتديات الفريق العربي للبرمجة –قسم الأكسس- امتدت على مدى سبعة أشهر ونصف الشهر تقريباً. إذا عرفت هذا، فإنك تعرف سبب أي غرابة في ترتيب السرد أو أسلوب الكتابة أو صيغ الخطاب، على الرغم من محاولة التنقيح بحذف الردود، والردود على الردود.

رابط السلسلة في منتديات الفريق العربي للبرمجة:

http://www.arabteam2000-forum.com/index.php?showtopic=191995&st=0&start=0

أحمد مبارك الحيقي

جميع الحقوق محفوظة للكاتب: تستطيع بدون مقابل مادي أن تستخدم مادة هذا الكتاب لفائدتك الشخصية، وفي التدريس (ولو كان بمقابل مادي). كما تستطيع نشر هذا الكتاب مجاناً حيث شئت، شريطة عدم التغيير فيه. وفي حالة الاقتباس أو النسخ، عليك ذكر المصدر والعزو إلى الكتاب. لا يسمح باستخدام الكتاب تجارياً بأي شكل عدا التدريس إلا بعد مراجعة الكاتب على البريد التالي: ahmadalhaiqi@hotmail.com. هذا شرطي لاستخدام الكتاب، والمسلمون عند شروطهم.

1430 هـ - 2009 م

المحتويات

SQL: مقدمة..... 3
فكرة الوصول إلى البيانات في SQL..... 5
أقسام اللغة..... 8
تصفية السجلات..... 12
العوامل المنطقية..... 17
المزيد عن تقييد عدد السجلات..... 21
دوال التجميع..... 24
فكرة التجميع..... 26
المزيد عن التجميع..... 30
فكرة الاستعلام من أكثر من جدول..... 32
طرق الربط بين جدولين..... 34
أمثلة للاستعلام من أكثر من جدول..... 37
الاستعلامات الفرعية SUBQUERIES..... 40
دوال SQL..... 44
إضافة، تعديل، وحذف السجلات..... 50
خاتمة..... 52

القسم الثاني: SQL

SQL: مقدمة

لا بد مما ليس منه بد. والمقدمات لا بد منها. أنا لا أحب أن أقذفك مباشرة في الماء، ثم أراك تخبط بيديك، فإما أن تصل إلى الحافة، وإما....؟ وبالمناسبة، بما أنك ضيفي في هذه الدورة، فإنه يجدر بي أن أنبهك إلى بعض الأشياء التي أحبها، وربما تجد نفسك مضطراً إلى أن تتحملها مدة الدورة. أنا أحب المقدمات، وأحب الأمثلة، وأحب الأسئلة، وأحب الاستطراد أحياناً (الفلسفة بالعربي)، وأحب أن أخفت إضاءة وتباين الشاشة لأن عيني ليستا على ما يرام، وأحب الشاي بالحليب. والآن، دعني أسألك السؤال الأول...

هل من الضروري أن تتعلم SQL؟

نعم، هذا يجب أن يكون في أوائل أهدافك على طريق احتراف برمجة قواعد البيانات. باختصار، SQL هي اللغة الوحيدة التي تستطيع أن تستخدمها للوصول إلى قاعدة البيانات مباشرة. قد لا يكون هذا واضحاً بالنسبة للمبتدئ، وخاصة في الأكسس، لأنه لم ير حتى الآن إلا شاشات تصميم (جداول ، استعلامات، نماذج، تقارير). دعني في هذه النقطة أذكر فقط القليل من المهام التي ستحتاج فيها إلى SQL، وستتمنى عندها لو تعلمتها يوماً ما...

إنشاء كائنات قاعدة البيانات، وأهمها الجداول، بالكود.

إنشاء استعلامات معقدة، لا يعلم معالج الاستعلامات في أكسس حتى بوجودها (مثلاً، استعلامات التوحيد).

معالجة البيانات برمجياً، وصدقني، هذه الكلمة أكبر بكثير مما تتصور. لا تتوقع أن تكون برامجك كلها مجرد استعراض لبيانات الجداول، وإضافة أو حذف أو تعديل سجل هنا أو هناك، ستبدأ تطلق على نفسك اسم مبرمج قواعد بيانات عندما تقوم برامجك ببعض الاحتسابات التي لا يستطيع أن يقوم بها المستخدم عبر النماذج، مثلاً، احتساب الرواتب. إلى جانب لغة البرمجة التي تستخدمها، مثلاً VBA، ستكتب الكثر من عبارات SQL.

لا أود أن أطيل الوقوف هنا، لكن القصد أن تقتنع أنك تحتاج بالفعل إلى SQL، ومن الأفضل ألا تنام بينما تقرأ في هذه الدورة.

ولكن، ما هي SQL؟

ألم نقل ذلك بعد؟ غريب، إن SQL، ولا فخر، هي اللغة الرسمية لقواعد البيانات العلائقية (موضوع هذه السلسلة). بما أن هذه الدورة موجهة أساساً للمبتدئين، دعني أوضح بعض النقاط التي قد لا تأخذ حقها من الفهم. كما ذكرت من قبل، هدفي هنا هو الاهتمام بالمفاهيم الأولية، وما أكثر ما تفوت علينا المفاهيم الأولية!

في البداية، دعنا نتأمل قليلاً في معنى (لغة). من الجيد أن نتذكر بين حين وآخر أن الحاسوب الذي أصبحنا نعتمد عليه كثيراً هو في النهاية آلة إلكترونية أخرى، تعمل بالكهرباء. لكن الفرق الجوهري بينه وبين آلة الغسيل على سبيل المثال، هو أن تصميمه منذ البداية كان على أساس أنه جهاز متعدد المواهب. تعدد المواهب هذا يأتي من حقيقة أن المعالج (الذي هو جوهر الحاسوب) مصمم بطريقة خاصة، بحيث تختلف الوظيفة التي يؤديها حسب اختلاف مجموعة من الإشارات الكهربائية التي تستطيع دوائره الإلكترونية التعرف عليها. هذه الإشارات الإلكترونية تأتي في مستويين فقط (جهد عالي، وجهد منخفض)، اصطلحوا على تسميتهما (واحد وصفر). التشكيلات المختلفة التي يستطيع المعالج فهمها والاستجابة لها أسموها تعليمات. مجموع هذه التعليمات يسمى لغة، ولأن تعليمات هذه اللغة تتكون من إشارات من مستويين فقط، فإن لغة المعالج (تسمى أيضاً لغة الآلة) يقال لها: لغة ثنائية Binary Language. تستخدم تعليمات هذه اللغة في كتابة برامج، وهذا هو سر تعدد مواهب الحاسوب: أنه قابل للبرمجة. التفاصيل هنا كثيرة، لكن لكي أجعل القصة الطويلة أقصر مايمكن، أقول: إن الإنسان استمر في تسهيل الأمور على نفسه (ما انفك يبني طبقة فوق طبقة ليخفي التفاصيل الصعبة) حتى أصبح يبرمج الحاسوب ليس بلغته التي يفهمها مباشرة، لكن بلغات أسهل. هذه السهولة تكمن في أنه لم تعد هناك تشكيلات من (الصفر والواحد)، لكن كلمات بلغة يفهمها الإنسان (الإنجليزية)، يستطيع بواسطتها أن يكتب تعليمات من أجل مختلف المهام (مثل تعريف متغيرات، التحكم في سير البرنامج، نسخ قيم المتغيرات... إلخ). مجموع هذه الكلمات يسمى لغة برمجة. طبعاً، يحكم استخدام هذه الكلمات قواعد دقيقة، هي من ضمن ما يميز لغة برمجة عن أخرى. النقطة المهمة هنا، أن التشكيلات (التعليمات) المختلفة التي يستخدمها المبرمج الآن، ليست بينه وبين المعالج، لأن المعالج (بصفته جهازاً إليكترونياً في النهاية) لا يفهم هذه التشكيلات، وهذا يعني أن هناك (طبقة) ما، هي عبارة عن برنامج، هي التي تستقبل هذه التشكيلات من المبرمج، وتترجمها إلى تشكيلات يفهمها المعالج. هذا البرنامج يسمى المترجم أو المجمع (interpreter or compiler)، وكل مترجم أو مجمع مخصص للغة محددة. الخلاصة هنا (لمن بدأ يتثاءب هناك) أن لغة البرمجة المحددة هي مجموعة الكلمات التي يفهمها مترجم

ما. هذه الكلمات قد نسميها تعليمات، وهي على أنواع، مثل الكلمات في اللغة العربية (اسم وفعل وحرف، ما زلت تذكر ذلك، هه؟).

بالمثل، SQL هي لغة، لكنها ليست لغة برمجة تقليدية، ولا يترجمها مترجم أو مجمع مثل مجمع لغة الـC، أو Java . لغة SQL هي مجموعة من التعليمات بين المبرمج وبين برنامج خاص من نوع آخر، يسمى برنامج (مفاجأة) إدارة قواعد البيانات DBMS. لا بد أن المتابعين لهذه السلسلة قد بدأوا يكرهون هذا البرنامج من عدد المرات التي قرؤوا اسمه فيها. وهذه التعليمات تستخدم في مهام تمتد من إنشاء قاعدة البيانات مروراً بالوصول إلى بيانات قاعدة البيانات، ومعالجة هذه البيانات إلى التحكم في الوصول إلى قاعدة البيانات وبياناتها. هذه المهام أكثر بكثير مما يوحي به اسم اللغة: Structured Query Language أو لغة الاستعلامات البنيوية (أو الهيكلية أو سمها ما شئت). لكن يمكن من الاسم أن تدرك أن من أهم وأكثر استخدامات هذه اللغة هي الاستعلام عن (استعادة) البيانات من قاعدة البيانات. بالمناسبة، كما أن مترجم الفيجوال بيسك على سبيل المثال، لا يفهم إلا لغة البيسك، فإن الـDBMS (ونحن نتكلم هنا عن قواعد البيانات العلائقية Relational Databases) لا يفهم إلا لغة SQL. وعلى هذا فأنت بصدد التعرف على مجموعة من الكلمات (تشكل في مجموعها لغة SQL) تستخدم في الوصول ومعالجة بيانات قواعد البيانات (بالإضافة إلى المهام المذكورة أعلاه).

هذه اللغة صممت منذ البداية من أجل قواعد البيانات العلائقية، وعلى الرغم من أنها ليست الوحيدة، إلا أنها بدون منافس الأكثر استخداماً، وقد تبنتها مجموعة من المنظمات التي تصدر المعايير أو المقاييس standards، وهذا يعني أن لغة SQL الآن لغة standard، وهذا بدوره يعني أنها لغة متفق عليها ومدعومة من مصنعي برامج إدارة قواعد البيانات. بالمناسبة، هذا يعني أنك بتعلم هذه اللغة تستطيع التخاطب مع عدة أنظمة (غير الأكسس) مثل Oracle وSQL Server، وغيرها، على اختلافات طفيفة، على الأقل في أساسيات اللغة. من أجل ذلك قلنا في البداية إنها اللغة الرسمية لقواعد البيانات العلائقية.

أرجو أن تكون ماهية SQL الآن أوضح قليلاً مما كان قبل بضع دقائق...

فكرة الوصول إلى البيانات في SQL

والآن ندخل قليلاً في الجد...

إذا كنت قد قرأت الحلقة السابقة بتمعن، فربما وصلت إلى بعض الملحوظات:

SQL هي لغة، والـ(C) والـ (Java) هي أيضاً لغات. لكن الوسيط بينك وبين الحاسوب في حالة لغات (مثل VBA) على سبيل المثال، هو برنامج المترجم، أما الوسيط بينك وبين الحاسوب في حالة SQL هو برنامج DBMS الضخم (مثل الأكسس). كما أنه قد تمت الإشارة إلى أن SQL قد صممت منذ البداية من أجل قواعد البيانات العلائقية. هل هذا يعني أن هناك فرقاً ما؟

المتابع معنا من بداية السلسلة ربما يذكر شيئاً عن الصعوبة التي كان يواجهها المبرمج عندما يستخدم اللغات التقليدية قبل SQL، في برمجة تطبيقات قواعد بيانات. مصدر هذه الصعوبة هو أنه كان على المبرمج أن يستخدم ملفات نظام التشغيل مباشرة لحفظ واستعادة البيانات، وهذا يعني كابوساً حقيقياً، لأنه يعني أن على المبرمج التعامل مع التفاصيل الدقيقة (المتعبة) لتنسيق الملفات، وترتيب البيانات حسب هذا التنسيق، ومعرفة أين وكيف تم الحفظ بالضبط من أجل الاستعادة والمعالجة فيما بعد. هذا يعني كما ذكرنا في حلقات سابقة أنه كان على المبرمج التعامل مع الطبقة الفيزيائية لقاعدة البيانات، المتمثلة في الملفات على القرص الصلب. أضف إلى هذا كل أخطاء التكامل التي كان يعاني منها مثل هذا النظام (تحدثنا عنها سابقاً). أضف إلى هذا كل عمليات الإدارة المرهقة التي كانت تقع على عاتق المبرمج (مثل قضايا الأمان، والتزامن). بسبب هذا المستوى الدقيق من التحكم الذي كان مطلوباً من المبرمج، فإنه كان يستخدم لغات برمجة إجرائية (procedural)، وهذا يعني لغات برمجة تتيح للمبرمج تفصيل الكيفية الدقيقة للوصول إلى البيانات التي يحتاجها، ومن ثم معالجتها.

كان الحل لهذه الصعوبة مشابه لما ذكرناه في الحلقة السابقة فيما يتعلق بهروب المبرمج من لغة الآلة إلى لغات أسهل، مع إسناد مهمة الترجمة بين هذه اللغات الأسهل وبين لغة الآلة (التي لن يفهم المعالج سواها) إلى برنامج خاص، أسميناه المترجم أو المجمع. هذا البرنامج هو طبقة وسيطة بين المبرمج وبين التفاصيل الدقيقة للغة الحاسوب. بصورة مشابهة، تم بناء برنامج آخر ليتولى بدلاً من المبرمج التفاصيل الدقيقة لإدارة قاعدة البيانات، والتعامل مع مشاكل حفظ واستعادة البيانات فيزيائياً من ملفات نظام التشغيل، وليجعل حياة مبرمج قواعد البيانات (ومدير قاعدة البيانات) أقل كآبة. طبعاً، تمت تسمية هذا البرنامج نظام إدارة قواعد البيانات DBMS Database Management System. الآن، لم يعد المبرمج مضطراً إلى استخدام لغة إجرائية للوصول إلى البيانات ومعالجتها، لكنه مازال محتاجاً إلى لغة ما للتخاطب مع برنامج إدارة قواعد البيانات. بالنسبة لقواعد البيانات العلائقية، حدث أن أشهر لغة صممت لهذا الغرض هي SQL، وكما لك أن تتصور، فإن هذه اللغة لم تعد (إجرائية)، بل كانت (غير إجرائية non-procedural).

ولكن، ما الذي يعنيه هذا؟ هذا يعني أنك حين تستخدم لغة SQL، أنت لا تحدد التفاصيل الدقيقة لـ (كيف) تصل إلى البيانات، بل تحدد (ماذا) تريد من بيانات، وعلى برنامج إدارة قواعد البيانات أن يتصرف، وأن يحضر لك هذه البيانات بالكيفية التي تروق له. هذا بالمناسبة يعني أنك لا تملك التحكم الكامل بأداء برامجك، وبشكل أو بآخر تقع تحت رحمة برنامج إدارة قواعد بياناتك (ولهذا تتفاضل برامج إدارة قواعد البيانات في الميزات التي توفرها، وفي أدائها، وإذا اخترت واحدة منها، فليس أمامك الكثير لتضيفه).

لحسن الحظ، فإن SQL أصبحت اللغة القياسية التي اعتمدتها برامج إدارة قواعد البيانات، وأصبح لها مواصفات قياسية من منظمات مثل ANSI وISO، على كل منتج لبرنامج إدارة قواعد البيانات أن يوفرها على الأقل. هل يلتزم الجميع بهذه المواصفات؟ هذه نقطة ترجع إلى كل منتج، لكنه على الأقل سيخبرك بماذا التزم، وبماذا لم يلتزم. من أجل ذلك تجد بعض الفروق بين تفاصيل هذه اللغة بين منتج وآخر، لكن الأوامر الأساسية تجدها في كل مكان، وتبقى اللغة واحدة، وإن تعددت الميزات الإضافية، وبعض التفاصيل مثل أنواع البيانات والدوال مثلاً. نسخة أكسس من SQL مثلاً، تسمى Jet SQL. ليس هذا فقط، بل قام منتجو برامج إدارة قواعد البيانات بتزويد SQL بقابلية المعالجة الإجرائية التي تتوفر في اللغات الإجرائية، فأصبحت تجد عبارات التكرار Looping (مثل For, While)، والتفرع Branching (مثل IF). هذه النسخ الموسعة من SQL تختلف من منتج إلى آخر، ولذلك تختلف أسماؤها. مثلاً، في Oracle، تسمى PL/SQL، وفي MS SQL Server، تسمى T-SQL. في أكسس، يمكن استخدام VBA كلغة إجرائية تعوض نقص SQL في هذا الجانب.

(تمرين عارض اختياري، وليس إجبارياً، فقط للمهتم:

هذه العبارة منسوخة من ملفات مساعدة أكسس 2003 عندي:

Microsoft Jet database engine SQL is generally ANSI -89 Level 1 compliant. However, certain ANSI SQL features are not implemented in Microsoft® Jet SQL. With the release of

Microsoft Jet version 4.X, the Microsoft OLE DB Provider for Jet exposes more ANSI-92 SQL syntax. Conversely, Microsoft Jet SQL includes reserved words and features not supported in ANSI SQL.

ابحث في ملفات المساعدة عن:

Comparison of Microsoft Jet SQL and ANSI SQL

الآن دعني أوضح ما معنى أن SQL غير إجرائية، وأنها تتيح لك فقط تحديد (ماذا) تريد من بيانات. سأتبع هنا بإذن الله أسلوباً تصويرياً مبسطاً، وإن كان غير مألوف ربما، من أجل توصيل المفهوم.

نحن نعلم مما سبق من حلقات السلسلة أن الوحدة الأساسية لقواعد البيانات العلائقية هي الجدول (أو العلاقة، بالتعبير الرسمي لهذا الموديل). والجدول بدوره يتكون من مجموعة من الأعمدة التي تحدد بنية الجدول، ومن مجموعة من الصفوف التي تمثل أفراد الكائنات التي يمثلها هذا الجدول. الأعمدة لها أسماء، لأنها تمثل خصائص الكائنات التي يمثلها الجدول، وواحد أو أكثر من الأعمدة يستخدم كمفتاح للتمييز بين الصفوف المختلفة (المفتاح الأساسي Primary Key). المفتاح يعني حقلاً أو أكثر قيمه لا تتكرر في صفين أو أكثر. يمكن تمثيل هذا بالشكل التالي:

Column#1	Column#2	Column#3	Column#4	Column#5	Column#6	Column#7
1						
2						
3						
4						
5						
6						
7						

حيث يمثل العمود الأول ذو الخط، المفتاح الأساسي.

الفكرة هنا أن نفكر بأعمدة وصفوف فحسب عندما نريد الوصول إلى البيانات باستخدام SQL. فكر بالجدول على أنه مجموعة من الأعمدة، فإذا أردت كل تفاصيل الجدول، أي كل خصائص الكائنات التي يمثلها الجدول، فهذا يعني أنك تريد كل(أعمدة) الجدول، وإذا أردت بعض الخصائص فحسب، فأنت تريد بعض الأعمدة. كذلك إذا اخترت كل أو بعض الخصائص، فربما تريد كل أو بعض الكائنات في الجدول، وهذا يعني أنك تريد كل أو بعض (الصفوف) في الجدول. أكرر: وصولنا إلى البيانات هو عبر أعمدة وصفوف، وكل القيم التي سنحصل عليها هي ناتج تقاطع الأعمدة مع الصفوف.

(ماذا) أريد؟ أريد العمود الأول والثالث والسابع:

Column#1	Column#2	Column#3	Column#4	Column#5	Column#6	Column#7
1						
2						
3						
4						
5						
6						
7						

تريد كل الصفوف؟ لا، أريد الصفوف ذات المفاتيح 1 و2 و3 و7:

Column#1	Column#2	Column#3	Column#4	Column#5	Column#6	Column#7
1						
2						
3						
4						
5						
6						
7						

وبهذا، يكون (ما) أريده هو (اللون الرمادي الداكن جداً):

Column#1	Column#2	Column#3	Column#4	Column#5	Column#6	Column#7
1						
2						
3						
4						
5						
6						
7						

تتعامل SQL بهذا المبدأ. هي تتيح لك إمكانية اختيار الأعمدة، ثم تحديد بعض الصفوف، عبر أوامر وعبارات خاصة. ثم هي تتيح لك أيضاً ترتيب الصفوف التي اخترتها حسب عمود معين أو أكثر، واستبعاد المكرر من الصفوف إن وجد، وغيرها من العمليات. صفوف، وأعمدة. حتى في عمليات مثل ربط بين جدول وآخر، الفكرة هي لصق أعمدة جدول مع أعمدة جدول آخر. وعملية مثل التوحيد ما هي إلا لصق صفوف مع صفوف (نفس الأعمدة).

أقسام اللغة

يوحي اسم SQL (Structured Query Language لغة الاستعلامات البنيوية) بأن وظيفتها هي فقط استعادة البيانات، لكن الحق أن إمكانياتها أكبر من ذلك بكثير. دعني أوضح هذه النقطة قليلاً حتى نتجنب بعض اللبس، ثم نمضي بعد ذلك إلى المزيد من التفاصيل.

عرفنا أنك ستتعامل مع برنامج إدارة قواعد البيانات، وهو سيقوم عنك بكل التفاصيل، أعني تفاصيل المطلوب منه. كما عرفنا أنك ستحتاج إلى وسيلة ما لتخبره بالمطلوب، وهذه الوسيلة هي لغة SQL. الآن فكر معي، لو كنت ستصمم لغة للتخاطب بينك وبين برنامج إدارة قواعد البيانات، ما الإمكانيات التي يجب أن توفرها هذه اللغة؟

قاعدة البيانات مليئة بالبيانات التي نحتاج إلى وسيلة فعالة وسريعة لاستعادتها، واستخراج معلومات مفيدة منها. هذا يعني أننا بحاجة إلى أوامر في اللغة نستخدمها للاستعلام. ولكن، من أدخل هذه البيانات في قاعدة البيانات أصلاً؟ إذاً، نحن أيضاً بحاجة إلى أوامر لإدخال البيانات في المقام الأول. طبعاً، لو لم يكن في الحياة إلا عمليات إضافة واستعادة لكانت حياة جميلة حقاً، ولكنك بحاجة إلى تعديل مستمر للبيانات المدخلة، بل وحذف بعض ما سبق إدخاله، وهكذا. نحن أيضاً بحاجة إلى أوامر توفرها اللغة من أجل كل ذلك. هل هذا كل شيء؟ لا طبعاً. السؤال البديهي، أنت تدخل البيانات وتعدلها، أو تحذفها، من جداول في قاعدة البيانات. من صمم هذه الجداول؟ ليس برنامج إدارة قواعد البيانات بالتأكيد، وإلا لكنا الآن في الشارع نبيع حلويات رمضان، وليس في المنتدى نناقش تصميم قواعد البيانات. على الرغم من أن برنامج إدارة قواعد البيانات هو الذي (سينفذ) بناء الجداول، لكن تصميم الجداول ستخبره به أنت، وذلك عبر أوامر مناسبة بالطبع. ربما لا أحتاج إلى القول بأنك قد تحتاج إلى حذف بعض الجداول، لكن من المناسب أن أذكرك بأن هناك عمليات أخرى، ليست في بالك الآن، ستحتاج من برنامج إدارة قواعد البيانات أن يقوم بها من أجلك، تتعلق بإدارة المستخدمين مثلاً. أمور مثل إضافة وحذف مستخدم، ومنح صلاحية معينة، وما إلى ذلك. كل هذه المهام تحتاج إلى أوامر مناسبة توفرها لغة التخاطب الوحيدة مع برنامج إدارة قواعد البيانات.

ما الذي يعنيه كل هذا؟ يعني أن إمكانية إنشاء الجداول، ومعالجة البيانات، مثلها مثل إمكانية الاستعلام عن البيانات، كلها جزء من لغة SQL، وليست إضافة. ذكرنا من قبل أن الإضافات تتعلق بإمكانية المعالجة الإجرائية، التي لا توفرها SQL بصورتها الأصلية. مرة أخرى: المعالجة الإجرائية تعني أنك تحدد (كيفية) القيام بمهمة ما، وهذا يعني الخطوات التفصيلية للمهمة. في حالة إنشاء جدول مثلاً، أنت لا تملي لبرنامج إدارة قواعد البيانات (كيف) ينشئ الجدول، ولكنك تخبره فقط (ماذا) تريد أن يتكون منه الجدول. أشياء مثل أسماء الحقول (الأعمدة)، ونوع بياناتها. لكنك لن تخبره بالتفاصيل من قبيل (أين يقع الجدول على القرص الصلب، وما هي الهياكل المستخدمة لحفظ البيانات).

إذاً، أين هي الإضافات الإجرائية التي تتكلم عنها؟ في Oracle، تسمى PL/SQL (يمكنك أن تلاحظ الإضافة الإجرائية procedural في الجزء الأول من الاسم: Procedural Language PL). في SQL Server، تجد T-SQL (Transact SQL). في الأكسس، يمكنك التفكير باللغة VBA على أنه الجزء الإجرائي الذي يعوض نقص Jet SQL.

من جملة ما سبق، يمكن أن تلاحظ أن هناك عدة أنواع من أوامر SQL، حسب نوع العملية التي تؤديها. الواقع أنهم يقسمون اللغة عادة إلى قسمين أو ثلاثة. لاحظنا مثلاً أنه لا بد أن يكون هناك جزءاً من اللغة يوفر إمكانية إنشاء الجداول، وحذفها، وجزءاً آخر يتعامل أكثر مع البيانات من حيث إدخالها وتعديلها وحذفها وحتى استعادتها. على هذا، من الممكن أن نقسم لغة SQL إلى لغتين فرعيتين:

لغة تعريف البيانات (DDL) Data Definition Language
لغة معالجة البيانات (DML) Data Manipulation Language

أحياناً تجد نوعاً ثالثاً يسمى لغة التحكم بالبيانات (DCL) Data Control Language

القسم الأول يشمل الأوامر الخاصة بتعريف بنية الجداول، وحذفها، أو تعديلها، وإنشاء العلاقات، وربما تعريف المستخدمين وحذفهم والتحكم في صلاحياتهم، وكلمات مرورهم، وتغيير كلمة مرور قاعدة البيانات، وقليل من العمليات الأخرى. لاحظ أننا نتكلم هنا عن بنية أو هيكل الجداول (تعريف الحقول: عددها ونوعها)، ولا نتكلم عن البيانات التي سيحويها الجدول. الجزء الخاص بالتحكم بالمستخدمين قد يصنف أحياناً تحت قسم DCL.

القسم الثاني يشمل كل أوامر معالجة البيانات. هذه المعالجة تضم أربعة أنواع:

- الاستعلام عن البيانات
- إضافة البيانات
- حذف البيانات
- تعديل البيانات

لاحظ أن هذا القسم يشمل أكثر العمليات استخداماً، ولذلك سنتناوله هو إن شاء الله. لاحظ أيضاً أنه كما ذكرنا في الحلقة السابقة، يمكن الوصول إلى البيانات باعتبار الأعمدة والصفوف: في حالة الإضافة، يتم إضافة صفوف بكاملها إلى الجداول، وكذلك في حالة الحذف، تتم عمليات الحذف في صورة صفوف كاملة، أما في حالة التعديل أو الاستعلام، فيمكن الوصول إلى أعمدة محددة من جدول أو أكثر، ويمكن كذلك تحديد صفوف معينة، وبذلك يمكن الوصول حتى إلى خانة واحدة تمثل تقاطع عمود محدد مع صف محدد. ربما لاحظت كذلك أنه من جملة العمليات أعلاه، يمثل الاستعلام العملية الأكثر استخداماً، ليس فقط لأن الناس يدخلون البيانات مرة، ثم يستعيدونها بقية الوقت، ولكن عمليات مثل الحذف والتعديل (وأحياناً الإضافة) تحتاج إلى تحديد البيانات التي ستتم عليها العملية (مثلاً، تحديد الصف الذي سيحذف، أو العمود الذي سيعدل)، وآلية هذا التحديد هي نفسها آلية الاستعلام. من أجل هذا سيكون أول جزء نتناوله بإذن الله، هو الخاص بالاستعلام. في الحقيقة، إذا لم تخرج من هذه الدورة بأكثر من هذا الجزء، فقد خرجت غانماً بإذن الله.

هل أنت مستعد لأول أوامر لغة SQL؟ ولكن، ليس بعد!

على رسلك، حذرتك من أنني أحب المقدمات منذ البداية، لكنني أرجو أنك ستذكر هذه المقدمات يوماً ما، وتقول: لقد كان على حق! كما أنني أعمد إلى تكرار المعلومات من زوايا مختلفة، وفي سياقات مختلفة، لأن ذلك أدعى للفهم. على كل حال، ربما كنت مخطئاً في هذا الأسلوب، لكن حتى أكتشف ذلك، عليك أن تتحمل قليلاً، ربما خرجت من كل هذا الهراء بشيء في النهاية!

ما أود التقديم له هنا هو قليل من الاصطلاحات، حتى نضمن أن بعضنا يفهم بعضاً جيداً فيما بعد، مع التنبيه على أن ترجمة المصطلحات هي من اختياري.

نحن نستخدم لغة SQL في جمل أو عبارات، فنقول: عبارة SQL Statement.

أحياناً، تحوي العبارة جزءاً خاصاً له وظيفة ما، فتشير إلى هذا الجزء على أنه مقطع Clause.

العبارات تتكون من كلمات بالطبع، كلها تسمى كلمات محجوزة Reserved Words، بعضها أوامر Commands تشكل أساس العبارة، وهناك أجزاء مساعدة منها ما يسمى Predicate (لا ترجمة!)، ومنها ما يسمى عوامل Operators. يكفي أن تعلم ذلك الآن، حتى إذا جئنا على التفاصيل، ذهب كل شيء إلى مكانه المناسب.

الاستعلام عن البيانات

من أجل متابعة الدورة عملياً، عليك أن تنشئ جدولاً في أكسس حسب ما يلي، ثم تستخدم شاشة تصميم الاستعلامات (معاينة SQL). لا معاينات تصميم بعد اليوم ☺ إذا كنت لا تعلم ما هي معاينة SQL، فحاول البحث عنها لوحدك ☺

سنحتاج الآن إلى الجدول التالي (tblEmployee):

Field Name	Data Type	Description
EmpNO	Number	
EmpName	Text	
EmpManager	Number	
EmpHireDate	Date/Time	
EmpSal	Number	
EmpPhone	Text	
EmpDept	Number	

املأه بالبيانات التالية (حسناً، لأتجنب الدعوات الساخطة في رمضان، قمت بإرفاق الجدول في ملف ☺):

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
	٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
	٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
	٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
	٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
	٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
	٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
	٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
	٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
	٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
	٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
	٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
	٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
	٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
*							

عرفنا أن الوصول إلى بيانات جدول في SQL يتم باختيار حقل (عمود) أو أكثر. إذا اخترت حقلاً، فإن الذي سيعيده إليك أكسس كل قيم الحقل التي تمتد من السجل (الصف) الأول إلى السجل الأخير في الجدول. غالباً أنت لن تحتاج لكل سجلات جدول في استعلام ما، ولذلك بإمكانك أيضاً اختيار مجموعة محددة من السجلات فقط، عبر فلترة سجلات الجدول. بإمكانك بعدها ترتيب هذه السجلات المعادة تنازلياً أو تصاعدياً. هذه العمليات هي موضوع الحلقات التالية.

بما أننا نتحدث عن (الاختيار)، فإن اسماً مناسباً قد تم اختياره للأمر الذي سنستخدمه، وهو SELECT. هذا الأمر هو بطل لغة SQL بلا منازع، ويستخدم كالتالي:

SELECT … FROM …

في الفراغ الأول تضع أسماء الحقول التي تريد استعادتها (مفصولة بفاصلة بين كل اسم حقل وآخر)، وفي الفراغ الثاني تحدد الجدول أو الجداول التي تحوي هذه الحقول (أيضاً مفصولة بفاصلة، لكن هذا فيما بعد بإذن الله). مثلاً، نستطيع استعادة أرقام وأسماء ورواتب الموظفين في الجدول السابق باستخدام العبارة التالية:

```
SELECT EmpNo, EmpName, EmpSal FROM tblEmployee
```

إذا أردت أن تعطي الحقول أسماء أخرى في الاستعلام الناتج، وهو ما يسمى بالاسم المستعار Alias، فتستطيع استخدام الكلمة المحجوزة (AS) كالتالي:

```
SELECT EmpNo AS [رقم الموظف], EmpName AS [اسم الموظف], EmpSal AS [الراتب] FROM tblEmployee
```

الناتج هو:

	رقم الموظف	اسم الموظف	الراتب
▶	٧٣٦٩	SMITH	٨٠٠
	٧٤٩٩	ALLEN	١٦٠٠
	٧٥٢١	WARD	١٢٥٠
	٧٥٦٦	JONES	٢٩٧٥
	٧٦٥٤	MARTIN	١٢٥٠
	٧٦٩٨	BLAKE	٢٨٥٠
	٧٧٨٢	CLARK	٢٤٥٠
	٧٧٨٨	SCOTT	٣٠٠٠
	٧٨٣٩	KING	٥٠٠٠
	٧٨٤٤	TURNER	١٥٠٠
	٧٨٧٦	ADAMS	١١٠٠
	٧٩٠٠	JAMES	٩٥٠
	٧٩٠٢	FORD	٣٠٠٠
	٧٩٣٤	MILLER	١٣٠٠
*			

بإمكانك أيضاً استعادة كل حقول الجدول دفعة واحدة (بدلاً من كتابتها كلها واحداً واحداً) باستخدام النجمة مكان قائمة الحقول:

```
SELECT * FROM tbLEmployee
```

ما دمنا في قائمة الحقول، فمن المناسب التنبيه على أنه يمكنك تطبق العديد من العمليات على الحقول أثناء استعادتها. عندما نتعرض للدوال على اختلاف أنواعها (رياضية، نصية، تواريخ...) بإذن الله، فإننا سنرى هذه الإمكانية عن قرب، لكن كمثال سريع الآن، من الممكن أن نستخدم (+)، التي تلعب دور عامل لصق concatenation operator مع النصوص، ودور عامل الجمع مع الأرقام:

```
SELECT EmpNO AS [رقم الموظف], "Mr. " + EmpName AS [اسم الموظف], EmpSal + 1000 AS [ الراتب مع
1000 زيادة]
FROM tblEmployee
```

الناتج هو:

رقم الموظف	اسم الموظف	الراتب مع زيادة ١٠٠٠
٧٣٦٩	Mr. SMITH	١٨٠٠
٧٤٩٩	Mr. ALLEN	٢٦٠٠
٧٥٢١	Mr. WARD	٢٢٥٠
٧٥٦٦	Mr. JONES	٣٩٧٥
٧٦٥٤	Mr. MARTIN	٢٢٥٠
٧٦٩٨	Mr. BLAKE	٣٨٥٠
٧٧٨٢	Mr. CLARK	٣٤٥٠
٧٧٨٨	Mr. SCOTT	٤٠٠٠
٧٨٣٩	Mr. KING	٦٠٠٠
٧٨٤٤	Mr. TURNER	٢٥٠٠
٧٨٧٦	Mr. ADAMS	٢١٠٠
٧٩٠٠	Mr. JAMES	١٩٥٠
٧٩٠٢	Mr. FORD	٤٠٠٠
٧٩٣٤	Mr. MILLER	٢٣٠٠

تمرين:

نستطيع في الحقيقة تضمين حقول وهمية غير موجودة في الجدول، جرب التالي:

```
SELECT 1, * FROM tblEmployee
```

أو حتى:

```
SELECT "رمضان مبارك" FROM tblEmployee
```

لاحظ كيف كتبنا النص بين علامتي تنصيص ""، ولاحظ أيضاً من فضلك عدد الصفوف المعادة في الجملة الأخيرة. سنتطرق بإذن الله إلى الفائدة من مثل هذه الإمكانية فيما بعد. هل تستطيع التفكير في أي فائدة الآن؟

بعد اختيار الحقول المطلوبة، يبقى الاحتمال غير كبير في أنك ستحتاج إلى كل سجلات الجدول دفعة واحدة (تذكر أن الجداول التي ستتعامل معها في الحياة العملية ستحوي في الغالب أكثر بكثير من مجرد 14 سجلاً). الوسيلة من أجل فلترة سجلات الجدول هي استخدام مقطع إضافي في عبارة الأمر SELECT، وهذا المقطع يسمى WHERE Clause، ويبدأ (بالمصادفة البحتة) بالكلمة المحجوزة WHERE. هناك أيضاً وسيلة أخرى مفيدة للحد من عدد السجلات المستعادة، لكن كل هذا هو موضوع الحلقة القادمة بإذن الله...

تصفية السجلات

الأمر SELECT مجرداً، يعيد لك الحقول المختارة بكل سجلاتها (مثلاً، إذا حددت رقم واسم الموظف، فإن أرقام وأسماء كل الموظفين هي الناتج)، وهذا لن تحتاج إليه غالباً. سترغب عادة في جزء محدد من السجلات، يحقق شرطاً أو أكثر. مثلاً، أنت تريد استعادة أرقام وأسماء الموظفين الذين تم تعيينهم في السنة الماضية. لغة SQL تقدم بعض الطرق لتقييد عدد السجلات. لاحظ من فضلك أن (تقييد) عدد السجلات يمكن أن يكون بتصفية (فلترة) السجلات التي تحقق بعض الشروط كما تقدم، وقد تكون ببعض الطرق الأخرى، مثل استبعاد السجلات المكررة. من أجل تصفية السجلات، أنت تستخدم المقطع WHERE مع الأمر SELECT، وهو بإذن الله موضوع الأسطر التالية.

ما هي فكرة التصفية؟

التصفية هي استبعاد غير المرغوب. وهذا يعني انتقاء أو اختيار المرغوب. عندما تنتقي شخصاً ما لوظيفة تقدم إليها، فإن هناك على الأقل خاصية واحدة تختبرها لتنظر هل تحقق معياراً ما أم لا. مثلاً، أنت تنظر في خاصية آخر مؤهل، والشرط: هل هو جامعي أم لا؟ من الممكن أن يكون الشرط هو: هل هو فوق المستوى الثانوي؟ من الممكن أيضاً أن تنظر في خاصية أخرى وشرط آخر؛ مثلاً، الخبرة العملية: هل هي على الأقل سنتان؟ خاصية ثالثة؟ العمر؟ ممكن، أن يكون بين 25 و45 مثلاً.

الخصائص التي نملكها عند الحديث عن سجلات جدول هي الأعمدة، ولذلك، ستنظر عند تصفية السجلات في قيمة عمود أو أكثر، وتختبرها، هل تحقق شرطاً أو أكثر. الشروط في مجملها، كما هو الحال في المثال السابق، مقارنات من نوع أو آخر. قد تقارن قيمة حقل لتنظر هل (يساوي) قيمة محددة (في المثال السابق، هل المؤهل هو جامعي؟)، أم تقع ضمن مجال محدد من القيم (في المثال السابق، مجال العمر). المقارنات تمتد لتشمل أكثر من مجرد المقارنات الرياضية (=، >، <، =>، =<، <>)، إلى اختبار مطابقة قيمة ما لنمط معين pattern، بل حتى مقارنة قيمة حقل مع قائمة من القيم. لا تقلق، سنرى بإذن الله أمثلة بعد قليل.

عندما تصفي السجلات، فإنك أحياناً ترغب في استعادة سجل واحد محدد، وهذا يحدث غالباً عند عمليات البحث من أجل الاطلاع على بيانات موظف ما أو درجة طالب أو حالة مريض، أو التأكد من رصيد صنف ما على سبيل المثال، أو تعديلها أو حذفها. في هذه الحالة، ستستخدم حقل (أو حقول) المفتاح الأساسي من أجل تحديد هذا السجل بالذات، وتقارنه مع قيمة محددة. وفي الحقيقة، فإن هذه بالضبط هي فائدة المفتاح الأساسي: إمكانية التمييز بين الصفوف (السجلات) فيما بعد، والوصول إلى صف بعينه دون أدنى لبس.

في مقطع WHERE، أنت تستخدم كلمات خاصة محجوزة في اللغة لتجري هذه المقارنات مع عمود أو أكثر من الجدول. ولذلك، تسمى هذه الكلمات (عوامل مقارنة comparison operators). كما أشرت في الفقرة السابقة، عوامل المقارنة على أنواع. دعنا الآن نمر سريعاً على أهم هذه العوامل، لنرى كيف يمكن استخدامها في SQL.

المقارنات الرياضية

العوامل الرياضية مألوفة جداً للجميع، وإذا رجعنا إلى جدولنا المثال في الحلقة السابقة، فإن بإمكانك أن تطبق الأمثلة التالية التي توضح كيفية استخدام المقطع WHERE:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

```
SELECT * FROM tblEmployee
WHERE EmpNo = 7934
```

هذه العبارة تستعيد السجل الأخير فقط (بكل حقوله). لاحظ أننا استخدمنا المفتاح الأساسي، وهو رقم الموظف، لتحديد سجل واحد فقط. طبعاً، شرط المساواة قد يعيد أكثر من سجل، مثل الاستعلام التالي:

```
SELECT * FROM tblEmployee WHERE EmpDept = 30
```

ويعيد التالي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠

مثال آخر:

```
SELECT EmpNo, EmpName, EmpSal FROM tblEmployee
WHERE EmpSal <= 2000
```

الناتج من هذه العبارة هو:

EmpNo	EmpName	EmpSal
٧٣٦٩	SMITH	٨٠٠
٧٤٩٩	ALLEN	١٦٠٠
٧٥٢١	WARD	١٢٥٠
٧٦٥٤	MARTIN	١٢٥٠
٧٨٤٤	TURNER	١٥٠٠
٧٨٧٦	ADAMS	١١٠٠
٧٩٠٠	JAMES	٩٥٠
٧٩٣٤	MILLER	١٣٠٠

وهذا يستخلص السجلات التي تحقق الشرط التالي: الراتب أصغر أو يساوي 2000 (دولار؟ ريال؟ جنيه؟ بإمكانك أن تحلم!).

والآن، انظر كيف نكتب الشروط الخاصة بالنصوص والتواريخ في الأكسس:

```
SELECT * FROM tblEmployee WHERE empHireDate > #15/12/1981#
```

الناتج هو:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

لاحظ كيف يحاط التاريخ بين علامتي الهاش #. طبعاً الاستعلام أعلاه يعيد كل حقول السجلات التي تحقق الشرط: تاريخ التعيين أكبر من 1981/12/15 (لماذا هذا الاستعلام؟ لست أدري، ربما كانوا يريدون معرفة كل من تم توظيفه بعد سنة من افتتاح الشركة).

إذا أردت أن تعرف الرقم الوظيفي لمدير SCOTT، فيمكنك كتابة الاستعلام التالي:

```
SELECT EmpManager FROM tblEmployee WHERE EmpName = "SCOTT"
```

لاحظ من فضلك علامتي التنصيص حول القيمة النصية.

مقارنات المجال

ربما كانت القيم التي تريدها في حقل ما تقع في مجال محدد، مثلاً، لا تريد الموظفين الذين تم توظيفهم بعد تاريخ محدد، ولكن تريد الموظفين الذين تم توظيفهم في فترة محددة بين تاريخين (ربما فترة المدير السابق ☺). أو تريد استعادة الموظفين الذين تقع رواتبهم بين قيمتين. من أجل ذلك، تقدم SQL المعامل BETWEEN... AND...، وهو يستخدم كالتالي:

```
SELECT * FROM tblEmployee WHERE EmpSal BETWEEN 2000 AND 3000
```

الناتج هو:

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
	٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
	٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
	٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
	٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
*							

لاحظ أن القيمتين 2000 و3000 تدخلان في المجال المطلوب.

المقارنات مع قائمة من القيم

من الممكن أن تختبر القيم في حقل ما لتنظر هل هي من ضمن قائمة من القيم. هذه القائمة من الممكن أن تكتبها بنفسك في الاستعلام، لكن ستعلم فيما بعد أنك من الممكن أن تجلب قائمة من القيم عبر عبارة استعلام كاملة من جدول آخر. العامل المستخدم هنا هو IN، واسمه معبر جداً. في هذه المرحلة، دعنا نفترض أنك تريد بيانات ثلاثة من الموظفين: SMITH وJONES و MILLER:

```
SELECT * FROM tblEmployee WHERE EmpName IN ("SMITH", "JONES", "MILLER")
```

المقارنات مع الأنماط Pattern Matching

مررنا من قبل على استعلام يحوي مقارنة حقل بقيمة نصية محددة (اسم الموظف). المشكلة في الأسماء أنك قد لا تتذكرها كاملة، أو لا تعرف التهجئة الدقيقة للاسم. من الممكن أن تصف للأكسس بطريقة ما الجزء الذي تعرفه، والجزء الذي لا تعرفه، ويتولى هو الباقي. سيقوم بمطابقة قيم الحقل مع الوصف الذي أعطيته إياه، ويحاول استنتاج المطلوب. لكي تخبر الأكسس بهذا (الوصف) فإنك تستخدم العامل LIKE. هذا الوصف يسمى نمط (pattern)، ويحوي واحداً أو أكثر من رموز خاصة لها معاني محددة. هذه الرموز قد تكون صادفت مثلها عند البحث عن الملفات في نظام التشغيل، وتسمى wildcard characters.

ما هي الـ wildcard characters؟

هي رموز خاصة تقوم مقام حرف أو أكثر أو رقم، في الوصف (النمط) الذي تعطيه للأكسس كأساس للمقارنة مع قيم حقل. السبب أنك تحتاج إلى رموز خاصة لتقوم مقام الحروف هو أنك لا تعلم ما هي هذه الحروف التي يجب المقارنة معها بالضبط، فتكتب ما تعلمه، وتستخدم هذه الرموز لتقوم مقام ما لا تعلمه. السبب في أنها متنوعة هو

إعطاؤك إمكانية التحديد الدقيق لعدد ونوع الحروف التي نسيتها أو لا تعرفها. دعنا أولاً نلقي نظرة على هذه الرموز، ثم نوضح استخدامها ببعض الأمثلة.

الرمز	يقوم مقام...
؟	حرف واحد
*	لا حرف، حرف، أو أكثر
#	رقم واحد (من 0 – 9)
[a-z]	أي حرف واحد من المجال بين القوسين المربعين
[!a-z]	أي حرف واحد ليس من المجال بين القوسين المربعين

والآن، ما الذي يعنيه هذا؟

انظر إلى الكلمة التالية، ثم انظر إلى بعض الطرق التي يمكن أن تصفها بها باستخدام wildcard characters:

SMITH
?MITH
SM?TH
SMIT?
SMITH*
SMITH
*SMITH
SM*
S*
SMIT*
SM*TH
S*H
[S-Z]MITH
[SXZ]MITH
SMIT[!A-G]

والآن، دعنا نفترض بعض الحالات... مثلاً، أنت تريد بيانات الموظف (سكوت)، ولا تذكر رقمه، ولست متأكداً من أن اسمه ينتهي بحرف T واحد أو اثنين، لكنك تعلم أنه يبدأ بت(SCOT)، بدلاً من أن تنفذ الاستعلام مرتين (وتسمح للناس بأن يعيّروك بأنك لا تعرف الـ wildcard characters!)، من الممكن أن تكتب التالي:

```
SELECT * FROM tblEmployee WHERE EmpName LIKE "scot*"
```

لاحظ شيئين: الأول أنني تعمدت أن أكتب بحروف صغيرة حتى تعلم أنه لا فرق، والثاني أننا وضعنا النمط بين علامتي تنصيص لأن النمط يعامل كنص، حتى لو كنا نقارنه بأرقام كالتالي:

```
SELECT * FROM tblEmployee WHERE EmpNo LIKE "756#"
```

هنا، أنت تعلم أن رقم الموظف المطلوب يبدأ بـ(756)، لكنك نسيت الرقم الأخير. سيحضر الاستعلام بيانات الموظف JONES.

تمرين: جرب استخدام هذه الرموز في الاستعلام عن بيانات الموظف الذي تم تعيينه في سنة 1982.

اختبار القيم الخالية

من الأمور التي ينبغي أن تعيرها انتباهاً مفهوم القيمة الخالية. الكلمة المحجوزة NULL تعبر عن هذا المفهوم. هذه القيمة تستخدم في أي حقل من أي نوع لتدل على أن الحقل لم يستقبل أي إدخال بعد، أو أنه استقبل إدخالاً خاصاً بالقيمة الخالية NULL. لماذا تم ابتكار هذه القيمة؟ ألسنا نملك الصفر، والنص الفارغ ""؟ بلى، ولكن الصفر قيمة قد يكون لها دلالة معينة في الحقل، غير أن هذا الحقل فارغ. مثلاً، قيمة الخصم قد تكون صفراً بصورة متعمدة

للدلالة على أن الخصم (مقداره) صفر، وهذا يعني أن الحقل ليس بفارغ، ولكن قيمته صفر. أما القيمة الخالية فلها دلالات أخرى من أجلها تم ابتكار هذا المفهوم؛ هذه الدلالات هي:

- القيمة غير معروفة (مثلاً، لسنا نعلم هل للموظف رقم هاتف).
- القيمة مفقودة (مثلاً، للموظف هاتف، لكننا لا نعرف رقمه حالياً).
- القيمة غير قابلة للتطبيق (مثلاً، ليس للموظف هاتف أصلاً).

كل هذا التنظير القصد منه أن القيمة NULL هي قيمة مستقلة تختلف عن الصفر وعن السلسلة النصية الفارغة ("")، ولها دلالة خاصة. ما يهمنا هنا، أنه يحدث كثيراً أنك تحب أن تستبعد في استعلاماتك السجلات التي تحوي هذه القيمة الخالية في حقل ما، أو بالعكس، قد ترغب بشكل خاص باستعادة السجلات التي تحوي بعض حقولها هذه القيمة الخاصة (مثلاً، ما هي الأصناف التي ليس لها أرقام رف).

تقدم SQL العاملين IS NULL و IS NOT NULL من أجل هذه المهمة. وكمثال على استخدامهما، لنفرض أننا نود الاستعلام عن الموظفين الذين يملكون هواتف في مكاتبهم. العبارة التالية تقوم بالغرض:

```
SELECT * FROM tblEmployee WHERE EmpPhone IS NOT NULL
```

الناتج من هذا الاستعلام هو:

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
	٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
	٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
	٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
*							

لاحظ كيف أن الموظف TURNER قد ظهر في النتيجة بسبب أن حقل رقم هاتفه يحوي الحرف (0) بالخطأ. إذا أردت أن تعرف المدير من بين هؤلاء الموظفين، فابحث عن الموظف الذي لا مدير له! أي أن قيمة الحقل EmpManager هي القيمة الخالية:

```
SELECT * FROM tblEmployee WHERE EmpManager IS NULL
```

من الذي سيعيده هذا الاستعلام؟

سأتوقف اليوم هنا بإذن الله، لكن للحديث عن تصفية السجلات بقية إن شاء الله، فإلى ذلك الحين: هل من سؤال؟

العوامل المنطقية

ربما لاحظت أن أمثلة التصفية السابقة كانت تحوي مقارنة لحقل واحد مع قيمة واحدة، أو مع مجال من القيم، أو قائمة من القيم، أو مع نمط نصي باستخدام wildcard characters... لكن هناك دائماً شرطاً واحداً فقط. يحدث في كثير من الأحيان أن تحتاج إلى اختبار شرطين أو أكثر، ربما لنفس الحقل أو لأكثر من حقل، من أجل تصفية السجلات المعادة. مثال على ذلك، ربما كنت تريد معرفة الموظفين الذين تم توظيفهم قبل العام 1982، لكن راتبهم مازال تحت 1500. هذا الاختبار يحوي شرطين، وكل شرط يفحص حقلاً، واحد رقمي، والآخر حقل تاريخ. من أجل دمج أكثر من شرط في مقطع WHERE واحد، تقدم SQL العامل المنطقي AND، والعامل OR.

عوامل منطقية...! وهل بقية العوامل غير منطقية؟!

أخذت هذه العوامل نسبتها من علم المنطق، حين كان الناس بمزاج رائق لاختبار أمور مفروغ منها، على صعيد محاولة تأسيس طريقة رسمية للتفكير والاستنتاج. مثلاً، لنفرض أن كل عبارة لها قيمتان فقط: إما صحيحة وإما خاطئة. إذا كان لدينا عبارتين، فما قيمة العبارة الناتجة عنهما معاً؟ هذا يختلف حسب طريقة الجمع بينهما، وحسب قيمة العبارتين. على سبيل المثال، لنفترض أنك بدين، ولكي لا تنزعج، فلنفرض أنك وسيم أيضاً. عبارة مثل (أنت نحيف) هي خاطئة، لكن عبارة مثل (أنت نحيف أو وسيم) هي صحيحة، لأن الرابط (أو) يضمن أنك لن تخرج من المولد بلا حمص مادام هناك مولد أو حمص، أقصد أن الناتج هو صحيح مادام أحد الفرضين صحيح.. نحن نستفيد من هذا الهراء في الحاسوب في تطبيقات كثيرة، مثل التصميم المنطقي للدوائر، وهومبني على الجبر البولي Boolean Algebra، الذي يرتب الناتج من مثل هذه العمليات في جداول تسمى (جداول الحقيقة!) لكن من الأفضل تسميتها بجداول الصحة truth tables، حيث لكل عامل جدول يحدد الاحتمالات المختلفة لقيم الفرضيتين اللتين تدخلان في الاختبار، والناتج من كل احتمال. نستفيد من هذه الجداول في عبارات SQL، وفي العبارات الشرطية في لغات البرمجة أيضاً.

دعنا الآن من كل هذا (المنطق)، ولنر واحداً من هذه الجداول، لواحد من هذه العوامل، حتى نستطيع أن نمسك شيئاً في أيدينا. سأعبر عن هذا الجدول بكلمات تعبر عن موضوعنا. أفترض أن الشرط قيمته (صحيح) إذا تحقق، وقيمته (خطأ) إذا لم يتحقق. لننظر أولاً إلى العامل AND:

(الشرط الأول) AND (الشرط الثاني)	الشرط الثاني	الشرط الأول
خطأ	خطأ	خطأ
خطأ	خطأ	صحيح
خطأ	صحيح	خطأ
صحيح	صحيح	صحيح

العامل AND يختبر تحقق الشرطين معاً في نفس الوقت. الشرط الناتج لن يكون محققاً إلا إذا كان كلاً من الشرطين محققاً. مثلاً، هل أنت نحيف؟ لا. هل أنت وسيم؟ نعم. هل أنت نحيف (و) ووسيم؟ لا. فقط إن كنت نحيفاً ووسيماً في نفس الوقت، فإن الناتج هو نعم.

كيف نستفيد من هذا في تصفية السجلات؟ في المثال السابق، أنت تريد أن يتحقق شرطان معاً: الموظف تم توظيفه قبل عام 1982، (و) راتبه أقل من 1500. يمكن اعتبار الناتج من هذين الشرطين شرطاً كبيراً لن يتحقق إلا إذا تحقق كلا الشرطان. إذا كان الموظف لا يحقق واحداً من الشرطين، فأنت لا تريد استعادة سجله، ولذلك استخدمت العامل (و) AND.

ولكن، لنفرض أنك تود معرفة الموظفين المرشحين لنيل زيادة في الراتب. أنت تبحث الآن عن أي موظف تم توظيفه قبل عام 1982، (أو) راتبه أقل من 1500: تحقق واحد من الشرطين يكفي لاستعادة السجل. في هذه الحالة تستخدم العامل OR، الذي يبدو جدول صحته كالتالي:

(الشرط الأول) OR (الشرط الثاني)	الشرط الثاني	الشرط الأول
خطأ	خطأ	خطأ
صحيح	خطأ	صحيح
صحيح	صحيح	خطأ
صحيح	صحيح	صحيح

لاحظ أن هذا العامل مرن جداً: مادام واحد من الشرطين الداخلين في الاختبار صحيحاً (محققاً)، فإن الشرط الناتج هو محقق أيضاً. أنت تستطيع استخدام هذين العاملين لجمع عدد من الشروط في عبارة واحدة.

قبل الأمثلة، لا أريد أن أنسى أن هناك عاملاً منطقياً ثالثاً، هو عامل النفي: NOT. هذا العامل يستخدم للتأكد من أن شرطاً ما (ليس محققاً). أو إن شئت قل: يستخدم للتأكد من تحقق شرط بعدم صحة شرط ما (منطق، هه؟). مثلاً، أنت تريد أن تستعلم عن الموظفين الذين لا يبدأ اسمهم بالحرف S. لا أدري، ربما هو نوع من الحساسية من بعض الحروف. ربما كان أول مدرس رياضيات في حياتك يبدأ اسمه بهذا الحرف!

والآن، إلى الأمثلة (الجدول كاملاً مرة أخرى من أجل سهولة الرجوع إليه):

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
	٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
	٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
	٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
	٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٠٥٧	٣٠
	٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
	٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
	٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
	٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
	٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
	٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
	٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
	٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
	٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
*							

الموظفون الذين تم توظيفهم قبل عام 1982، وراتبهم أقل من 1500:

```
SELECT * FROM tblEmployee WHERE EmpHireDate< #1/1/1982# AND EmpSal < 1500
```

الناتج هو:

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
	٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
	٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٠٥٧	٣٠
	٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
*							

الموظفون الذين تم تعيينهم في عام 1981، ويعملون في القسم رقم 30، ولكن ليس مديرهم هو BLAKE ذي الرقم 7698:

```
SELECT * FROM tblEmployee WHERE (EmpHireDate Like '*1981') AND (EmpDept = 30) AND
(EmpManager <> 7698)
```

الناتج هو سجل BLAKE نفسه فقط. لاحظ استخدام الأقواس لتمييز كل شرط. ستعلم بعد قليل بإذن الله، أن الفائدة من هذا ليس مجرد جمال العبارة.

الموظفون الذين إما تم تعيينهم قبل العام 1982، وإما يقل راتبهم عن 1500:

```
SELECT * FROM tblEmployee WHERE (EmpHireDate < #1/1/1982#) OR (EmpSal < 1500)
```

قارن الناتج مع عبارة AND السابقة:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

عبارة OR أكثر مرونة، وقد أعادت كل الموظفين عدا SCOTT الذي تم توظيفه عام 1987 وراتبه 3000 (أرزاق!).

إذا كنت تكره الحرف S خصوصاً، ولا تريد أي موظف يحوي اسمه هذا الحرف، فتستطيع أن تشفي غليلك بهذا الاستعلام:

```
SELECT * FROM tblEmployee WHERE EmpName Not Like "*s*"
```

هذا يستبعد كلاًمن Smith، Jones، Scott، Adams، James.

بعض المحاذير

لنفترض أنك تريد المطلوب التالي: كل الموظفين الذين يحققون الشرطين التاليين:

1. تم توظيفهم قبل العام 1982
2. راتبهم أقل من 1000 أو أنهم من القسم رقم 20 (قسم ممتاز يستحق الترقية).

لديك مبرمج قرأ الجزء السابق من الحلقة، ولم يكمل حتى هذا الجزء، فقام بكتابة التالي:

```
SELECT * FROM tblEmployee WHERE EmpHireDate < #01/01/1982# AND EmpSal < 1000 OR
EmpDept =20
```

حاول أن تستنتج الناتج، يجب أن يكون تخمينك التالي (طبعاً، من الممكن أن تنفذ العبارة في الأكسس، لكن الأفضل أن تتدرب على آلية الاستعلام بنفسك):

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠

لاحظ أن الاستعلام قام باستعادة كل الموظفين الذين يقعون في القسم 20، حتى لو كان تاريخ تعيينهم في 1987، وهذا يتعارض مع الشرط المطلوب الأول.

الصحيح هو العبارة التالية:

```
SELECT * FROM tblEmployee WHERE EmpHireDate < #01/01/1982# AND (EmpSal < 1000 OR
EmpDept =20)
```

تعيد التالي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠

حاول أن تتأمل في الفرق من ناحية المعنى. الاختلاف في العبارة هو فقط في وضع الأقواس بعد AND، بحيث يتم أولاً اختبار الـOR، ثم يتم اختبار الناتج بـ AND مع الشرط الأول (شرط التاريخ). في العبارة الأولى كان يتم تجاهل شرط التاريخ لأن شرط القسم يتحقق، والعبارات كلها مرتبطة بـ OR. حاول أن تفك طلاسم هذا المنطق.

السبب في هذا اللبس كما رأينا هو ترتيب اختبار الشروط. بدون أقواس، يتم تنفيذ الاختبار بالترتيب من اليسار إلى اليمين، لكن باستخدام الأقواس، تستطيع إجبار الأكسس على اختبار شرط أولاً حتى لو لم يكن يقع في البداية (مثل اختبار الـ OR في المثال السابق بين شرط الراتب والقسم، قبل اختبار الـ AND بين شرط التاريخ وشرط الراتب). العبرة: انتبه لترتيب شروطك، واستخدم الأقواس بالشكل المناسب.

هل انتهينا من تصفية السجلات؟ ربما، لكننا لم ننته بعد من الإمكانيات الأخرى لتحديد وتقييد عدد السجلات المعادة... لكن هذا هو موضوع الحلقة القادمة بإذن الله... حتى ذلك الحين: هل من سؤال؟

المزيد عن تقييد عدد السجلات

رأينا كيف أنك تستخدم المقطع WHERE من أجل تقييد عدد السجلات المستعادة عبر تصفية السجلات. هذه التصفية تعني اختبار قيم حقل أو أكثر: هل تحقق شرطاً أو أكثر؟ هناك وسيلة أخرى لتقييد عدد السجلات المسترجعة، وذلك بمنع القيم المكررة، وباختيار عدد محدد من السجلات بغض النظر عن تحقق شرط ما غير أن تقع هذه السجلات في بداية أو نهاية مجموعة السجلات المعادة. نستخدم من أجل ذلك كلمات محجوزة تسمى Predicates، في مقطع SELECT قبل قائمة الحقول المختارة. فيما يلي توضيح لهذه الطريقة.

لا تذكرها مرتين (DISTINCT)

لفترض أن المدير الجديد يريد أن يأخذ فكرة عن أنواع الرواتب التي تصرف للموظفين، وطلب منك استعلاماً يعيد فقط أرقام الرواتب التي تصرف، وانسحبت أنت من عند المدير وأنت ترسم أمارات الجدية على وجهك، وفي داخلك ترتسم ابتسامة واسعة من الثقة؛ إنه استعلام تافه:

```
SELECT EmpSal FROM tblEmployee
```

أعاد الاستعلام التالي:

EmpSal
٨٠٠
١٦٠٠
١٢٥٠
٢٩٧٥
١٢٥٠
٢٨٥٠
٢٤٥٠
٣٠٠٠
٥٠٠٠
١٥٠٠
١١٠٠
٩٥٠
٣٠٠٠
١٣٠٠

وكعادتنا دائماً في البداية، لم تراجع نتيجة استعلامك، وسحبتها مباشرة إلى المدير، وعوضاً عن كلمات الثناء، أشار المدير بأصبعه إلى عدد من الأرقام، وهو يقول: ولكن، لماذا هذه مذكورة مرتين؟

كان المدير يشير إلى أرقام مثل 1250، و3000. ومن حسن حظك أن الموظفين هم فقط 14، فلو كان عدد الموظفين 100 مثلاً، فإن المدير سيطالع 100 رقم أغلبها مكرر، وربما لن يشير عندها إلى الورقة، بل سيشير إليك وهو يهز أصبعه متوعداً.

لو كنت قرأت هذا المثال قبل أن تستعجل بإخراج النتيجة للمدير، لاستخدمت الكلمة المحجوزة DISTINCT قبل اسم الحقل EmpSal من أجل استبعاد القيم المكررة في هذا الحقل:

```
SELECT DISTINCT EmpSal FROM tblEmployee
```

النتيجة هي 12 سجلاً بدلاً من الـ14 سجلاً السابقة (1250 و3000 موجودتان مرة واحدة فقط لكل منهما).

تمرين: حاول أن تغير الاستعلام السابق إلى التالي، وتنفذه:

```
SELECT DISTINCT EmpSal, EmpNO FROM tblEmployee
```

ماذا كانت النتيجة؟ ما الذي تستنتجه؟

الترتيب أولاً ORDER BY Clause

بعد يومين، نسي المدير الطيب حكاية الرواتب، وطلب منك أن تستخرج له آخر ثلاثة موظفين تعييناً. هذه المرة، تعمدت أن تطلع على المثال هنا قبل أن تنفذ الاستعلام، لكنك كنت تتساءل في نفسك باستمرار: الموضوع هنا ليس مسألة تقييد عدد سجلات... الموضوع فيه ترتيب! أنت على حق كالعادة، ولذلك دعنا نر أولاً كيف ترتب SQL مجموعة السجلات المعادة...

من أجل ترتيب مجموعة من السجلات، يجب أن تحدد الحقل (أو مجموعة الحقول) التي سيتم الترتيب على أساسها. تستطيع أن ترتب الموظفين مثلاً حسب حقل الراتب. في حالة تشابه الرواتب لبعض الموظفين، فيمكن بعدها ترتيب سجلات الموظفين النتشابهين في الراتب حسب الاسم. دعنا نر كيف يمكنك ترتيب الموظفين حسب تواريخ تعيينهم من أجل طلب المدير:

```
SELECT * FROM tblEmployee
ORDER BY EmpHireDate
```

فقط، بكل بساطة! أضف المقطع ORDER BY (رتّب حسب) إلى الأمر SELECT، وحدد الحقل أو الحقول التي تنوي أن ترتب مجموعة السجلات بحسبها، مع مراعاة ترتيب ذكر الحقول، لأن ترتيب السجلات سيتم بناءً على الحقل الأول، ثم حسب الحقل التالي في حالة تساوي قيمة الحقل الأول، وهكذا.

النتيجة من الاستعلام السابق هي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠

ربما لا تبدو هذه النتيجة مريحة بالنسبة لك، لأن التواريخ الأقدم مذكورة أولاً، وأنت تنبهني في كياسة إلى أن المدير يتوقع أن يجد أحدث التواريخ أولاً. لقد فكروا في هذا طبعاً عندما صمموا SQL، أليس كذلك؟ أنت تسأل. والحق أن نباهتك هذه الأيام مرتفعة بشكل ملحوظ (أهو نقص السكر في رمضان؟ ولكن، كيف؟)، والحق أن SQL تقدم الكلمتين المحجوزتين ASC وDESC اللتين تستخدمهما بعد كل حقل في المقطع ORDER BY لتحدد نوع الترتيب المطلوب: تصاعدي من الأدنى إلى الأعلى (ASC)، أو تنازلي من الأعلى إلى الأدنى (DESC). طبعاً، من الواضح أنك تحتاج إلى الطريقة الثانية تحديداً، وعلى هذا، فإن العبارة تصبح:

```
SELECT * FROM tblEmployee ORDER BY EmpHireDate DESC
```

والناتج هو:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠

والآن، لنعد إلى طلب المدير، فإنه لن ينتظر طويلاً...

أريد فقط الأول (TOP)

أنت الآن تفكر كيف يمكن استخلاص ثلاثة سجلات فقط من بداية النتيجة السابقة؟ من الممكن طبعاً أن تستخدم المقطع WHERE، وتحدد شرطاً على حقل التاريخ، ليكون أكبر أويساوي 1982/01/23. هذه الطريقة تفترض أنك تعلم قيمة أصغر ثلاثة تواريخ نعيين سلفاً، وهي جامدة بطريقة رهيبة، بحيث لو وقف المدير على رأسك الآن وفكر فجأة (بعد ما أعجب بكفاءتك) أن يستعلم عن أعلى ثلاثة رواتب، فإنك ستكون مضطراً لترتب أولاً الموظفين حسب رواتبهم تنازلياً، لتعرف أعلى ثلاثة رواتب، ثم تضيف مقطع WHERE لتكتب الشرط بناءً على أكبر ثالث راتب مثلاً. إذا كانت لدى المدير أدنى فكرة عن SQL (ربما مر على هذه الحلقة بينما كان يتصفح المنتدى بعد التراويح)، فإنه سيعيد النظر في مدى كفاءتك، ولربما نصحك بمطالعة هذه الدورة في تأنيب، وأنت طبعاً ستقول: ولكنه لم يكن قد وصل إلى تلك النقطة بعد، لقد خُدعت!

على كل حال، هكذا تستخدم الكلمة المحجوزة TOP أجل استعادة أول ثلاثة سجلات:

```
SELECT TOP 3 * FROM tblEmployee ORDER BY EmpHireDate DESC
```

فقط؟ نعم، فقط. أضف كلمة TOP بعد SELECT، ثم حدد عدد السجلات التي تريدها (أي عدد صحيح، في حالة كان العدد أكبر من عدد السجلات كلها، فإن كل السجلات يتم استرجاعها). نتيجة هذا الاستعلام هي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

لاحظ من فضلك أن نتيجة استخدام TOP تعتمد على ترتيب السجلات في المقام الأول، ثم على طريقة الترتيب. في حالة ترتيب الرواتب مثلاً تنازلياً، فإن TOP 5 تسترجع أعلى خمسة رواتب، أما في حالة ترتيب السجلات حسب الراتب تصاعدياً، فإن TOP 5 تسترجع أصغر خمسة رواتب. في حالة عدم استخدام المقطع ORDER BY، فإن TOP 4 على سبيل المثال ستعطيك أي أربعة سجلات عشوائياً حسب ملفات مساعدة أكسس، وإن كنت ستلاحظ عملياً أن الترتيب يتبع ترتيب إدخال السجلات (لا تعتمد على هذا).

هناك تنويع آخر للكلمة TOP، وهو أن تستخدمها مع الكلمة PERCENT (نسبة مئوية). وعلى هذا، فإن العدد المحدد بعد TOP يعني نسبة مئوية من العدد الكلي للسجلات. دعنا نر مثالاً على ذلك:

```
SELECT TOP 50 PERCENT * FROM tblEmployee ORDER BY EmpNo
```

كم عدد السجلات التي تتوقع أن يعيدها هذا الاستعلام؟ أنا شخصياً أتوقع أن تعيد التالي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠

الترتيب حسب رقم الموظف مفهوم، لكن لماذا كان عدد السجلات المسترجعة 7 بالتحديد؟ ذلك لأن 50 (كما في العبارة) تعني 50%، وذلك يعني النصف، ونصف 14 هو 7.

هذا كل شيء عن تحديد عدد السجلات. والآن: هل من سؤال؟

دوال التجميع

مراجعة سريعة

عرفنا كيف أن SQL هي لغة قواعد البيانات العلائقية، وقواعد البيانات العلائقية هي مجموعات من الجداول، والجدول هو مجموعة من الأعمدة ومجموعة من الصفوف. أشهر أمر في SQL، وهو SELECT يستخدم في اختيار مجموعة من (أو كل) أعمدة جدول. تستطيع تصفية صفوف هذه الأعمدة المستعادة باستخدام المقطع WHERE. فكرة التصفية هي تحديد شرط أو أكثر، حيث يقارن كل شرط قيم حقل بطريقة أو بأخرى مع قيم محددة. تختلف طريقة المقارنة حسب نوع بيانات الحقل، وحسب فكرة الشرط (هل تساوي قيمة حقل قيمة ما أو تقع بين قيمتين مثلاً). كما مررنا ببعض طرق تحديد عدد الصفوف المستعادة باستبعاد القيم المكررة في حقل، وبتحديد عدد مطلق (أو نسبة مئوية) للصفوف المستعادة، وفي أثناء ذلك، عرفنا كيف يمكن أيضاً ترتيب الصفوف المستعادة حسب قيم حقل أو أكثر. لاحظ من فضلك أن كل السابق كان في سياق الحديث عن أول نوع من عمليات معالجة البيانات، وهو الاستعلام (تبعاً للتصنيف الشائع يتقسيم لغة SQL إلى جزء خاص بمعالجة البيانات، يسمى لغة معالجة البيانات DML ، وجزء خاص بتعريف هياكل البيانات، يسمى DDL). إذا كنت تشعر أن بعض السطور السابقة غير واضح، فمن الأفضل مراجعة الحلقات القليلة السابقة، أو القراءة في أي مرجع للغة SQL (وما أكثرها).

في كثير من الأحيان، لا نريد سرد صفوف الجدول سرداً، حتى بعد تصفيتها، وترتيبها. في كثير من الأحيان، يكون ما نحتاج إليه هو خلاصات لهذه البيانات. مثلاً، أنت لا تريد أسماء موظفي كل قسم، ولكن تريد عددهم. ولا تحتاج إلى مطالعة تفاصيل كل فاتورة، ولكن تريد إجمالي كل فاتورة. نستخدم الدوال لاستخراج هذه الخلاصات، لكننا غالباً ما نلجأ إلى فرز وتجميع الصفوف في مجموعات، ولا نستخدم الدوال على كامل الجدول.

قبل عرض فكرة التجميع، ودوال التجميع، أرى أنه من المفيد هنا من أجل متابعة سليمة أن أتطرق إلى مفهوم الدوال بصورة عامة، ما دمنا نتحدث عن المبادئ، وما دامت الدورة موجهة أساساً للمبتدئين.

مفهوم الدالة

في الرياضيات، الدالة هي علاقة بين مجموعتين من القيم. أنت تعطي قيمة من المجموعة الأولى، والعلاقة تحدد قيمة من المجموعة الثانية، لذلك تسمى المجموعة الأولى منطلق الدالة، والثانية مستقرها. هذه العلاقة تتنوع تنوعاً هائلاً. قد تكون العلاقة هي التربيع، فنحصل على دالة التربيع: تعطي الدالة قيمة، فترد لك مربع القيمة. مثلاً، لنسم هذه الدالة sqr. هذا الاسم لا يكفي؛ يجب تحديد قيمة من منطلق الدالة من أجل استعادة قيمة من مستقرها. يتم ذلك اصطلاحاً بتحديد هذه القيمة بين قوسين بعد اسم الدالة، كالتالي sqr(3). نتيجة تطبيق هذه الدالة على هذا الرقم هي 9، فنكتب sqr(3) = 9.

في عالم البرمجة، مفهوم الدوال مشابه. العلاقة هنا هي برنامج كامل (لكنه صغير غالباً) يقوم بعملية الربط بين القيمة المعطاة، والقيمة المستعادة. مثال صغير ربما يجعل النقاش أكثر وضوحاً:

توفر لغة VBA دالة تسمى Len، وهي تأخذ قيمة نصية، وتعيد عدد حروف النص. تستخدم هذه الدالة كالتالي:

```
Dim lngLength As Long
lngLength = Len("SQL")
```

يحوي المتغير lngLength بعد تنفيذ السطر السابق القيمة 3، وهي عدد حروف النص "SQL". لاحظ هنا التالي:

1. الدالة لها اسم، وتأخذ قيمة بين قوسين.
2. الدالة تعيد قيمة، ولأننا مبرمجين نستخدم الحاسوب، ولسنا رياضيين نستخدم الأوراق، نحتاج إلى متغير لحفظ هذه القيمة المعادة، ولذلك عرفنا متغيراً اسمه lngLength لاستقبال نتيجة تطبيق الدالة Len على القيمة النصية "SQL".
3. العلاقة بين القيمة النصية والقيمة المعادة هي أن القيمة المعادة هي عدد حروف القيمة النصية، لكن من حدد هذه العلاقة؟ وكيف يتم تطبيقها؟ كما قلت قبل قليل، هذه العلاقة هي في الحقيقة برنامج صغير قد سبقت ترجمته، وهو محفوظ مع عدد كبير من البرامج (الدوال) في ملف واحد. هذا الملف يسمى مكتبة. مزيد من التفاصيل في هذا الصدد فيما بعد إن شاء الله، إن قدر لنا الحديث مبادئ البرمجة. في برامجك، من الممكن أن تكتب دوالك الخاصة من أجل استخدامها فيما بعد، ولكنك تستعين كثيراً بالدوال الجاهزة التي توفرها لغة البرمجة التي تستخدمها.

في لغات البرمجة، قد تستقبل الدالة أكثر من قيمة، لكنها تعيد دائماً قيمة واحدة. بالمناسبة، تسمى القيم التي تستقبلها الدالة معاملات (parameters).

لغة SQL أيضاً توفر لك العديد من الدوال، بعضها تستخدم في شروط المقطع WHERE من أجل المقارنة، وبعضها تستخدم في مقطع الأمر SELECT من أجل تطبيق عملية ما على الحقول المستعادة، وبعضها من أجل توفير تلخيص للبيانات. أشرنا سابقاً إلى أن التعامل في SQL يتم على أساس أعمدة (حقول) وصفوف (سجلات). ولهذا لك أن تتوقع أن الدوال في هذه اللغة تطبق على مستوى الأعمدة، وهذا يعني أن معاملها اسم عمود (مع معاملات أخرى ربما حسب نوع الدالة).

مثال:

الدالة ()count، تستقبل اسم حقل (أو أكثر)، وتعيد عدد الصفوف التي تحوي قيماً غير خالية في هذا الحقل. إذا أردت عدد الصفوف بشكل عام، حتى تلك التي تحوي قيماً خالية، استخدم النجمة مكان اسم الحقل. مثلاً، العبارة التالية:

```
SELECT COUNT(empPhone) FROM tblEmployee
```

يعيد حقلاً واحداً وصفاً واحداً يحوي القيمة 4 (لماذا؟)، في حين أن العبارة:

```
SELECT Count(empName) FROM tblEmployee
```

تعيد العدد 14، ومثلها العبارة

```
SELECT Count(*) FROM tblEmployee
```

هذه الدالة مثال على ما يسمى دوال تجميع SQL Aggregation Functions (ابحث عن هذا الاسم في ملفات مساعدة أكسس)، ومعها دوال مثل ()Avg، ()First، ()Last، ()Min، ()Max، وبالطبع الدالة الشهيرة ()Sum. هناك أنواع أخرى من الدوال تستخدم في معالجة النصوص والتواريخ على سبيل المثال. دعونا أولاً نركز على دوال التجميع من أجل تقديم مفهوم التجميع في الحلقة القادمة بإذن الله. الفكرة كلها في أنك كثيراً من الأحيان لن تحتاج إلى تطبيق دالة مثل مجموع قيم حقل (()sum) على جدول كامل بشكل عام. مثال على هذا، تستطيع الحصول على مجموع إجمالي كل الفواتير من جدول تفاصيل الفواتير باستخدام مثل هذه العبارة:

```
SELECT SUM(InvDetQty * InvDetPrice) FROM tblInvoiceDetails
```

لكن من الصعب أن يطلب أحد مثل هذا الاستعلام، لأنه يعيد قيمة واحدة هي مجموع إجماليات كل الفواتير في تاريخ المنشأة! حتى مع تقييد الاستعلام بتاريخين مثلاً، نحن نريد عادة مجموع إجمالي تفاصيل كل فاتورة على حدة، وليس مجموع إجماليات تفاصيل جميع الفواتير. من هنا جاء مفهوم التجميع، الذي أعرض له إن شاء الله في حلقة قادمة.

السؤال المعتاد: هل من سؤال؟
الجواب المعتاد:.....!

أمزح طبعاً!

فكرة التجميع

فكرة التجميع تتعلق بالصفوف: لا تطبق دوال التجميع على كافة الصفوف مرة واحدة، ولكن افرز الصفوف إلى مجموعات، ثم قم بتطبيق الدوال على كل مجموعة على حدة. كيف نفرز الصفوف إلى مجموعات؟ حسب حقل أو أكثر. مثلاً، بفرض أن لدينا جدول فيه بيانات عمال، مع حقل للبلد، في الشكل التالي كل بلد له لون:

الرقم	الاسم	تاريخ الميلاد	الوظيفة	البلد
				بلد1
				بلد2
				بلد1
				بلد3
				بلد2
				بلد2
				بلد2

بعد تجميع الصفوف حسب حقل البلد، يكون الناتج كالتالي:

الرقم	الاسم	تاريخ الميلاد	الوظيفة	البلد
				بلد1
				بلد1
				بلد2
				بلد2
				بلد2
				بلد2
				بلد3

لاحظ أن الصفوف المشتركة في حقل البلد تم تجميعها معاً، والترتيب حسب اسم البلد. الآن يمكن تطبيق دالة من دوال التجميع على هذه المجموعات. وعلى فرض أن الدالة كانت دالة العدد COUNT، فإن الناتج يكون كالتالي (صيغة العبارة في مثال لاحق إن شاء الله):

العدد	البلد
2	بلد1
4	بلد2
1	بلد3

بضع نقاط:

مصطلح التجميع يمكن تفسيره بـ(فرز إلى مجموعات).
لاحظ أننا نتحدث عن الصفوف بعد عملية التصفية، أي أننا نتحدث عن التجميع للصفوف التي نريدها بالفعل.
عملية الفرز تتم باستخدام مقطع خاص هو مقطع الكلمة المحجوزة GROUP BY، ويعني حرفياً (جمّع حسب...)، ومعيار الفرز هو حقل أو أكثر تتساوى في قيمته الصفوف المجمّعة معاً، كما في المثال التالي...

عوداً إلى جدول الموظفين، أعيد وضعه من أجل سهولة المرجع:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

ذكرنا في الحلقة السابقة أن التجميع له علاقة بالخلاصات، ومن أجل التمثيل، دعنا نفترض أن المطلوب يتعلق بعدد الموظفين. إذا كنت تريد الحصول على عدد الموظفين إجمالاً، فالعبارة التالية تفي بالغرض:

```
SELECT COUNT(EmpNo) FROM tblEmployee
```

لاحظ أنه كان من الممكن استخدام النجمة مكان اسم الحقل EmpNo (وهو أسرع في التنفيذ بالمناسبة)، لكنني اخترت اسم الحقل للتوضيح، ولأن حقل رقم الموظف مفتاح أساسي على كل حال، ونستطيع أن نطمئن إلى أنه يحوي قيمة دائماً (القيم الخالية لا تحسب في الدالة ()COUNT). في كثير من الأحيان، لا أحد يطلب خلاصة من هذا النوع، وإنما تكون هذه الخلاصة هي خلاصة الخلاصة. أعني أنه من المحتمل أكثر أن يكون المطلوب هو عدد الموظفين في كل قسم، ثم مجموع عدد الموظفين الكلي. في هذه الحالة، يتم تجميع بيانات الموظفين حسب حقل القسم، ثم يتم عد الموظفين في كل مجموعة (كل قسم هنا). طبعاً الذي يقوم بذلك هو الأكسس، لكن العبارة التي ستخبره بالمطلوب هي كالتالي:

```
SELECT COUNT(EmpNo) AS [Emp Count], EmpDept FROM tblEmployee
GROUP BY EmpDept
```

الناتج هو:

EmpDept	Emp Count
١٠	٣
٢٠	٥
٣٠	٦

لاحظ من فضلك التالي:

1. تمت تسمية العدد بـ(Emp Count). تستطيع تسميته بما تريد.
2. الجديد في العبارة السابقة هو المقطع GROUP BY، ويستخدم لطب تجميع الصفوف، ولتحديد الحقول التي سيتم حسبها التجميع.
3. أي حقل بجانب دالة التجميع، يجب ذكره في المقطع GROUP BY. في حالتنا هذه، الحقل EmpDept تم اختياره بجانب الدالة COUNT في مقطع الأمر SELECT، وتم ذكره في المقطع GROUP BY أيضاً.

النقطة الأخيرة تستحق المزيد من التفصيل. في المثال السابق، تم اختيار العدد واختيار حقل القسم، ولذلك كان لا بد من تكرار حقل القسم في المقطع GROUP BY. هذه هي القاعدة. في هذا المثال، تبدو القاعدة بديهية، وسخيفة، لأنه بالطبع يجب ذكر حقل القسم في المقطع GROUP BY حتى يعرف الأكسس على أي أساس سيتم تجميع الصفوف، لكن في حالات أخرى لا يكون المر واضحاً بهذا الشكل، ولذلك كان كسر هذه القاعدة من أكثر أخطاء المبتدئين شيوعاً في استخدام التجميع. أحياناً، يكون بجانب دالة (أو دوال) التجميع في مقطع

SELECT أكثر من حقل، وأنت تريد التجميع حسب حقل واحد منها مثلاً، لكن يلزمك ذكر جميع الحقول في مقطع GROUP BY حتى ولو لم يكن القصد التجميع حسبها.

بالمناسبة، يتم اختيار حقول إلى جانب دوال التجميع (في هذه الحالة حقل القسم) لأنه بدون أي حقول أخرى سيبدو الاستعلام بدون معنى؛ مجرد أعداد لا تدري أيها عدد القسم 10، وأيها تبع للقسم 20...

مثال آخر، رواتب الموظفين مجمعة حسب الأقسام:

```
SELECT EmpDept, Sum(EmpSal) AS SalaryByDept FROM tblEmployee GROUP BY EmpDept
```

EmpDept	SalaryByDept
١٠	٨٧٥٠
٢٠	١٠٨٧٥
٣٠	٩٤٠٠

مثال آخر، عدد الموظفين حسب سنة التعيين:

```
SELECT Year(EmpHireDate) AS HireYear, Count(tblEmployee.EmpNO) AS CountByYear
FROM tblEmployee
GROUP BY Year(EmpHireDate)
```

HireYear	CountByYear
١٩٨٠	١
١٩٨١	١٠
١٩٨٢	١
١٩٨٧	٢

تم استخدام دالة من دوال VBA، وهي الدالة Year، وتستقبل تاريخاً، ثم تعيد قسم السنة من هذا التاريخ. استخدمنا قسم السنة من تاريخ التعيين لتجميع الموظفين. لاحظ أننا طبقنا الدالة على التاريخ في المقطعين SELECT و GROUP BY.

نستطيع إضافة مستوى تجميع آخر إلى المثال السابق، فنجمع الموظفين حسب سنة التعيين (ثم) حسب القسم، كالتالي:

```
SELECT Year(EmpHireDate) AS HireYear, EmpDept, Count(tblEmployee.EmpNO) AS CountByYear
FROM tblEmployee
GROUP BY Year(EmpHireDate), EmpDept
```

نتيجة هذا الاستعلام هي:

HireYear	EmpDept	CountByYear
١٩٨٠	٢٠	١
١٩٨١	١٠	٢
١٩٨١	٢٠	٢
١٩٨١	٣٠	٦
١٩٨٢	١٠	١
١٩٨٧	٢٠	٢

لاحظ أن عدد صفوف الخلاصة قد زاد، لأننا فصلنا موظفي كل سنة حسب أقسامهم، ومن النتيجة أعلاه نجد أن سنة 1981 قد شملت تعيين موظفين في كل الأقسام، ونعلم حصة كل قسم من التعيين كل سنة.

تمرين: نفذ الاستعلام التالي

```
SELECT EmpDept, EmpName, Sum(EmpSal)
FROM tblEmployee
GROUP BY EmpDept, EmpName
```

لاحظ أن الاستعلام سليم تماماً من ناحية القواعد. ما نتيجة الاستعلام؟ هل توقعت ذلك؟ ما هو السبب؟

تمرين آخر: استخرج أدنى مرتب في كل قسم باستخدام التجميع والدالة ()MIN.

أرجو أن يكون مفهوم التجميع واضحاً، لأنه من الطرق المهمة في استخراج التقارير والاستعلامات، وفي حالة أي إشكال، فمن فضلك حاول أن تعيد قراءة الحلقة، أو أن تبحث عن مصدر أكثر وضوحاً، لكن لا تبق مجرداً من هذه الطريقة في تلخيص البيانات.

المزيد عن التجميع

مرة أخرى، ملخص سريع لعملية الاستعلام من جدول باستخدام SQL، حسب ما تم عرضه إلى الآن:

- اختيار حقل أو أكثر من الجدول في الأمر SELECT. تستطيع الاستعلام عن كل الحقول بكتابة النجمة فقط.
- إذا لم تكن ترغب في استحضار كل صفوف هذه الحقول، استخدم المقطع WHERE لتحديد بعض الشروط.
- هذه الشروط هي بشكل أساسي مقارنة لقيم الحقول (مستعادة أو غيرها) مع قيم تحددها أنت في الشرط.
- للتفاصيل عن أنواع هذه المقارنات، راجع من فضلك الحلقات السابقة.
- تستطيع أيضاً تقييد عدد الصفوف المستعادة باختيار عدد محدد من الأعلى باستخدام الكلمة TOP.
- تستطيع أيضاً التخلص من الصفوف المتطابقة باستخدام الكلمة DISTINCT.
- يمكنك ترتيب الصفوف المستعادة حسب أي حقل (أو حسب أكثر من حقل، واحداً بعد الآخر) باستخدام المقطع ORDER BY.
- من الطبيعي أنك ستحتاج إلى طريقة أخرى لاستخراج البيانات في شكل خلاصات وليس مجرد سرد كامل أو جزئي لصفوف الجدول. في هذه الحالة، أنت تبحث عن حسابات تستطيع تطبيقها على البيانات داخل الجدول. تتيح لك SQL القيام بهذه الحسابات عبر دوال تجميعية تستطيع إعادة مجموع أو عدد أو القيمة الصغرى أو الكبرى أو المتوسط الحسابي لقيم حقل (لاحظ مرة أخرى أننا نتعامل مع حقول). طبعاً بإمكانك تطبيق هذه الدوال على أكثر من حقل، لكن ليست هذه هي المشكلة. المشكلة أنك لا تريد تطبيق الحسابات على كل صفوف الجدول دفعة واحدة، وإنما على مجموعات داخل الجدول يجمعها رابط مشترك. مثلاً، تريد مجموع رواتب الموظفين بعد فرزهم في مجموعات، كل مجموعة تمثل قسماً في الشركة. الرابط المشترك هنا هو قيمة حقل القسم، لأن كلهم لهم نفس القيمة في هذا الحقل. نقول هنا إن (التجميع) تم حسب حقل القسم. أيضاً، نريد عدد الزبائن، لكن بعد تجميعهم حسب حقل المدينة، وهكذا. عملية التجميع ممكنة عبر المقطع GROUP BY.
- هناك بعض القيود في استخدام هذا المقطع؛ في قائمة الأمر SELECT، كل الحقول التي لم تدخل في حسابات تجميعية (لم نطبق عليها دوال تجميعية)، يجب ذكرها في قائمة المقطع GROUP BY.

أرجو أن تتأكد من استيعابك لكل الخطوات السابقة إجمالاً من ناحية المفهوم والوظيفة، وليس من ناحية طريقة كتابة الأمر، لأن هذا سيأتي بإذن الله مع الممارسة رغماً عنك. لكن إذا لم تفهم جيداً دور كل جزء بالضبط، فقد تجد نفسك عاجزاً عن حل مشكلة في الاستعلامات حتى مع استخدامك اليومي لأمر الاستعلام.

في هذه الحلقة، أعرض بإذن الله لمقطع يتعلق بالتجميع لا بد من المرور عليه، ويستخدم لتصفية السجلات أيضاً، ولكن بعد التجميع...

HAVING

لنفترض أنك تريد عدد الموظفين في كل قسم، لكن تود استبعاد الأقسام التي يقل عدد موظفيها عن خمسة موظفين. هل تستطيع استخدام WHERE لتنفيذ المطلوب؟

من أجل الإجابة عن هذا السؤال، نرجع إلى فكرة استخدام WHERE. هذا المقطع يستخدم لمقارنة قيم الحقول مع قيمة مطلقة أو مجموعة من القيم. هل هناك حقل في الجدول يحوي عدد الموظفين؟ إذاً، كيف ستتم المقارنة؟ ولهذا لا يمكن استخدام هذه الطريقة من أجل هذا المطلوب. أنت تقول: ولكن هذا الحقل سيكون موجوداً بعد تنفيذ الاستعلام، لأن الدالة COUNT ستعيد عدد الموظفين، ويمكن تجميعهم حسب القسم. كيف نستطيع الاستفادة من هذه الحقيقة في تصفية الخلاصات المستعادة؟ هنا يأتي دور المقطع HAVING. هذا المقطع يطبق شرطاً أو أكثر بطريقة مشابهة لما يفعله المقطع WHERE، ولكن على نتيجة دوال التجميع وليس على الحقول قبل التجميع.

نستخدم هذا المقطع لتنفيذ المثال السابق كالتالي:

```
SELECT Count(EmpNO) AS CountOfEmpNO, EmpDept
FROM tblEmployee
GROUP BY EmpDept
HAVING Count(EmpNO) >= 5
```

حاول استخدام العبارة التالية، وانتبه لرسالة الخطأ:

```
SELECT Count(EmpNO) AS CountOfEmpNO, EmpDept
FROM tblEmployee
GROUP BY EmpDept
HAVING Year(EmpHireDate) = 1981
```

ماذا تقول الرسالة؟ جرب استخدام هذاه العبارة:

```
SELECT Count(EmpNO) AS CountOfEmpNO, EmpDept FROM tblEmployee GROUP BY EmpDept
HAVING EmpDept =30
```

وماذا عن هذه العبارة؟

```
SELECT Count(EmpNO) AS CountOfEmpNO, EmpDept FROM tblEmployee GROUP BY EmpDept
HAVING EmpNo =7844
```

ما الذي تستنتجه؟

وهكذا لا تستطيع استخدام هذا المقطع إلا على الحقول المشاركة في التجميع أو نتيجته. الأمر مختلف كما علمنا سابقاً مع WHERE (كيف؟)

مثال

لنفترض أن المطلوب هو القسم الذي يحوي أكبر عدد من الموظفين مع عدد الموظفين فيه. لا تستطيع استخدام الدالة MAX مباشرة هنا في حدود ما تعلمناه، لأنه ليس هناك حقل يحوي عدد الموظفين في الجدول. لكن دعنا نرى ما يمكن عمله:

العبارة الأولى أعلاه تعيد عدد الموظفين في كل قسم. الصفوف مرتبة حسب حقل التجميع وهو القسم. هذه هي نتيجة العبارة الأولى:

CountOfEmpNO	EmpDept
٣	١٠
٥	٢٠
٦	٣٠

إذا فكرنا باستخدام TOP من أجل استعادة أول صف، فإنه يجب أن يكون الصف الأول صاحب أكبرعدد للموظفين، وهذا يعني أن نرتب أولاً الصفوف حسب عدد الموظفين، تنازلياً، هكذا:

```
SELECT Count(EmpNO) AS CountOfEmpNO, EmpDept
FROM tblEmployee
GROUP BY EmpDept
ORDER BY Count(EmpNO) DESC
```

النتيجة هي:

CountOfEmpNO	EmpDept
٦	٣٠
٥	٢٠
٣	١٠

لم يبق إلا إضافة TOP 1 لإتمام المطلوب:

```
SELECT  TOP  1Count(EmpNO) AS CountOfEmpNO, EmpDept
FROM tblEmployee
GROUP BY EmpDept
ORDER BY Count(EmpNO) DESC
```

الناتج هو سطر وحيد يمثل القسم 30 وعدد موظفيه 6.

في الحلقة القادمة بإذن الله سنحاول أن نتجاوز حدود الجدول الواحد، ونرى كيف نستخدم SQL استخداماً عملياً حقيقياً، وهو ما يستدعي دوماً الربط بين أكثر جدولين أو أكثر...

فكرة الاستعلام من أكثر من جدول

مقدمة

كان قد تقرر معنا أن قاعدة البيانات تحوي كل بيانات النظام، على الأرجح في عدد من الجداول، وليس في جدول واحد فقط. السبب في ذلك أن هنالك في الغالب أكثر من كائن واحد في النظام، كما أن قواعد التسوية تقودك إلى تقسيم بعض الجداول كما مر بنا في السابق. ثم إنه من الطبيعي أن تحتاج إلى البيانات المتفرقة في هذه الجداول من أجل استخراج معلومات مفيدة من قاعدة البيانات. كل ما سبق يقود إلى ضرورة توفر وسيلة للاستعلام من أكثر من جدول. حتى الآن، رأينا كيف نستعلم من جدول واحد، وربما تصورنا أنه من الممكن جداً أن نسحب نفس الطريقة على الاستعلام من جدولين فأكثر. مثلاً، في القسم الخاص بأسماء الحقول من مقطع الأمر SELECT يمكن أم نكتب أسماء حقول الجدولين، ثم نكتب اسمي الجدولين بعد الكلمة FROM. هذا من حيث الآلية الأساسية صحيح، لكن ثمة الكثير من التعقيدات التي تتعلق بكيفية اكتشاف البيانات المرتبطة بعضها مع بعض في الجدولين حتى يتسنى جمعها وعرضها معاً. لا يمكن أن تكتب أسماء حقول جدول الخصومات لموظف، وتكتب أيضاً أسماء حقول البيانات الشخصية للموظف في عبارة SELECT، ثم تطلب من الأكسس أن يحضر لك هذه البيانات من جدولين مختلفين. لا بد من توفر وسيلة يستطيع الأكسس بناء عليها أن يربط بين الموظف وخصوماته. هذه الوسيلة، كما لا بد أنك تحدث نفسك الآن، هي العلاقات.

العلاقات

تكلمنا سابقاً عن مفهوم العلاقات، وأنواعها. وأجد أنه من أجل الفهم السليم لهذه الحلقة، نحتاج إلى استذكار فكرة أو اثنتين حول العلاقات.

الأولى هي قضية منشأ العلاقات بين الجداول. من الجيد أن تتذكر أنك لا تخترع العلاقات اختراعاً، ولكنها موجودة في الأصل بين الوحدات المختلفة التي تم تمثيلها في قاعدة البيانات، لأنها تنتمي إلى نفس النظام. في الحقيقة، إذا لم تكن هناك علاقات طبيعية بين الوحدات، فإن وجودها في قاعدة بيانات واحدة هو محل نظر من الأساس. على سبيل المثال، في نظام للمبيعات، هناك علاقة طبيعية بين الفواتير والعملاء، لأن كل فاتورة محررة لعميل. هذا معناه أنك تكتشف العلاقات ولا تبتكرها.

الثانية، كيف يتم تمثيل هذه العلاقات في قواعد البيانات العلائقية بين مجموعة من الجداول؟ كل ما لدينا في قواعد البيانات العلائقية من مخازن للبيانات هو الجداول. والجداول هي مجموعة من الأعمدة (الحقول) والصفوف (السجلات). في الأنواع الأخرى من قواعد البيانات، كانوا يحتاجون إلى وسائل خارجية لتمثيل العلاقات بين الوحدات المختلفة، مثلاً المؤشرات، لكن قواعد البيانات العلائقية تمتاز بأن هذه العلاقات يتم تمثيلها ضمن البيانات نفسها في الجداول. كيف ذلك؟ العلاقة بين جدولين أو أكثر يتم تمثيلها باستخدام حقل (عمود) مشترك بين الجداول. لاحظ أن هذا يعني تكراراً لقيم هذا الحقل المشترك في أكثر من مكان، لكن هذا ثمن لا بد من دفعه من أجل الاحتفاظ بالعلاقات، ومن ثم تكامل قاعدة البيانات (أي نوع من التكامل؟).

بالرجوع إلى المثال السابق، من أجل تمثيل العلاقة بين جدول الفواتير وجدول العملاء، نحتاج إلى حقل مشترك بين الجدولين. هل نختار حقلاً من جدول الفواتير، ونكرره في جدول العملاء، أم نختار حقلاً من جدول العملاء، ونكرره في جدول الفواتير؟ دعنا نجيب عن هذا السؤال على مرحلتين. أولاً، على فرض اختيار حقل من جدول الفواتير، ووضعه في جدول العملاء، أو العكس، أي حقل يجب أن يكون؟ أي حقل سنختار من الجدول الأول لتمثيل العلاقة مع الجدول الثاني؟ لا بد أن نتفق على أن هذا الحقل يجب أن تكون قيمه في الجدول الأول فريدة لا تتكرر. لماذا؟ حتى يمكن أن نعلم تحديداً أي صف في الجدول الأول مشارك في العلاقة. هب أننا وضعنا حقل اسم العميل في جدول الفواتير من تمثيل العلاقة. اسم العميل قد يتكرر في جدول العملاء، ولهذا لن يستطيع الأكسس معرفة أي عميل بالضبط هو المقصود في فاتورة معينة حتى يحضر بقية بيانات العميل. أيضاً، إن وضعنا حقلاً مثل تاريخ الفاتورة في جدول العملاء من أجل تأسيس العلاقة بين العملاء والفواتير، لن نعرف أي فاتورة بالضبط مقصودة في صف من صفوف جدول العملاء، لأن هناك العديد من الفواتير في نفس التاريخ.

النقاش السابق يوضح أن الحقل المختار من جدول لتأسيس العلاقة مع جدول آخر ينبغي أن تكون قيمه فريدة لا تتكرر في الجدول الأول حتى نستطيع الرجوع إلى صف محدد من الجدول الأول عندما نقرأ قيمة هذا الحقل الموجودة في الجدول الثاني. اقرأ من فضلك العبارة السابقة ثانية، حتى تتأكد من فهمها، لأنني أنا نفسي أجد صعوبة في تتبع كلمات الجدول الكثيرة. الآن، هناك دائماً حقل تتوفر فيه صفة عدم التكرار بالإضافة إلى صفة أخرى مطلوبة هنا أيضاً، وهي التأكد من وجود قيمة دوماً، حتى نتأكد من وجود العلاقة دوماً. هذا الحقل يسمى المفتاح الأساسي كما تعلم. وعلى هذا نتفق على أننا إما سنضع حقل رقم العميل في جدول الفواتير، أو حقل رقم الفاتورة في جدول العملاء، لنستطيع فيما بعد ربط العملاء بفواتيرهم.

نضع أي حقل في أي جدول؟ إذا وضعنا حقل رقم الفاتورة في جدول العملاء، فإنه مع كل فاتورة تحرر لعميل يجب إضافة صف في جدول العملاء فيه بيانات هذا العميل بالضبط ما عدا رقم الفاتورة الجديدة سيختلف. من الواضح أن

هذا خيار غير ممكن، لأن رقم العميل سيتكرر وهذا غير ممكن (رقم العميل مفتاح أساسي). إذاً، فالحل هو وضع حقل رقم العميل في جدول الفواتير، وهذا لن يضير جدول الفواتير لأن لكل فاتورة عميل واحد أصلاً، فلن نضطر لتكرار رقم الفاتورة مع أي عميل آخر، ولا مع نفس العميل (في المرة القادمة سنحرر فاتورة جديدة برقم جديد لهذا العميل). وللذين ما يزالون يذكرون أنواع العلاقات من الحلقات الماضية، فإن هذه علاقة من نوع واحد إلى متعدد، وكقاعدة، ضع دوماً الحقل المشترك بين الجدولين في جهة المتعدد (جهة جدول الفواتير هنا، لأن لكل فاتورة عميل واحد فقط، لكن لكل عميل عدد من الفواتير).

لاحظ من فضلك أن هذا الحقل المشترك الذي افقنا على أنه مفتاح أساسي في أحد الجدولين، يسمى المفتاح الأجنبي في الجدول الآخر، ويمكن أن تتكرر قيمه هنا، إذ فقد بغربته ميزاته السابقة. فيمكن أن تجد رقم العميل مع أكثر من فاتورة في أكثر من صف من جدول الفواتير. كما أنه لا يتوجب علي التنبيه مرة أخرى على أن المفتاح الأساسي قد يتكون من أكثر من حقل، وعلى هذا قد نجد أكثر من حقل مشترك بين الجدولين المرتبطين بعلاقة.

الآن نحن نملك العلاقة، وهي المطلب الأساسي من أجل إمكانية الاستعلام من أكثر من جدول أساساً. فهل هذا كل شيء؟ للأسف، كالعادة، الحياة أكثر تعقيداً من ذلك. هنالك ما ينبغي أن تعلمه أولاً عن إشكاليات قد تواجهها عند الربط بين الجدولين.

طرق الربط بين جدولين

عدم وجود علاقة بين جدولين لا يسمح بالربط السليم بينهما. هذه العبارة تتضمن أن هنالك ربطاً غير سليم. ومن أجل استيعاب مفهوم الربط بين جدولين، وأهمية وجود علاقة بينهما، بل ضرورة الإعلان عن هذه العلاقة واستخدامها بالشكل المطلوب في عبارات SQL، دعنا نفترض أن لدينا جدولين من حقل واحد فقط لكل منهما، كما يلي:

t1
0
2
3
6

t2
1
2
3
4
5

ثم دعنا نفترض أننا نريد أن نعلم عن الأرقام في الجدولين، مما يضعنا أمام عدة خيارات منطقية (هناك خيارات غير منطقية، مثل استعادة أرقام عشوائية من الجدولين كيفما اتفق):

أحد هذه الخيارات، أن نفكر في استعراض الأرقام بشكل كامل، فنجمع بين أرقام الجدولين بطريقة منظمة، بحيث نغطي كل احتمالات ربط رقم من الجدول الأول برقم من الجدول الثاني، هكذا:

1	0
2	0
3	0
4	0
5	0
1	2
2	2
3	2
4	2
5	2
1	3
2	3
3	3
4	3
5	3
1	6
2	6
3	6
4	6
5	6

شكل الجدول الناتج لا يبدو مشجعاً، ويحوي الكثير من التكرار الذي لا يقدم أي معلومات مفيدة. في الحقيقة، ما حصلنا عليه للتو يسمى في الجبر العلائقي (حاصل الضرب الديكارتي لعلاقتين). تذكر من فضلك أنه في النموذج الرياضي الذي بنى عليه Codd نموذج قواعد البيانات العلائقية، يسمى الجدول (علاقة). هذا الضرب يجمع بين كل زوج في العلاقتين في عدد من الاحتمالات يساوي حاصل ضرب عدد عناصر العلاقة الأولى في عدد عناصر العلاقة الثانية (4 * 5 = 20 في حالتنا هذه). كل قيمة في الحقل الأول ارتبطت بكل القيم من الحقل الثاني. هذا النوع من الربط غير ذي جدوى في قواعد البيانات، وينتج عند عدم وجود (أو عدم استخدام) العلاقة بين جدولين. يمكنك الحصول على هذه النتيجة بواسطة SQL بأن تذكر حقول واسمي الجدولين فحسب في عبارة SELECT، كما يلي (هذه العبارة لا يقبلها الأكسس لسبب أذكره حالاً بإذن الله):

```
SELECT n, n FROM t1, t2
```

الفرض أن t1 هو اسم الجدول الأول، وt2 هو اسم الجدول الثاني. كما أن n هو اسم الحقل الرقمي في الجدولين. لكن المشكلة في هذه العبارة أن اسمي الحقلين متطابقان، وهذا يضع الأكسس في حيرة: أي الحقلين هو المقصود بـ n الأولى، وأيهما هو المقصود بـ n الثانية؟ لا تقل لي أن الأمر أوضح من أن يحتمل الحيرة، وأنه يمكن للأكسس أن يتصرف، ويفترض أن كل حقل من أحد الجدولين. هذا بالنسبة لي ولك، وللنملة التي تمشي الآن في أحد الأركان (ربما) يكون صحيحاً. لكن بالنسبة لجهاز أفضل تعريف له هو أنه أسرع غبي في العالم، فإن الحل الأنجع هو أن يقذف في وجهك رسالة تخبرك بكل أدب أن (اسم الحقل يحتمل أن يشير إلى أكثر من جدول)!

الحل لهذا الإشكال أن تميز كل حقل باسم الجدول الذي ينتمي إليه، وذلك بكتابة اسم الجدول بعده نقطة ثم اسم الحقل. اسم الجدول هنا يسمى qualifier، وتعني شيئاً معرفاً. تصبح العبارة كالتالي:

```
SELECT t1.n, t2.n FROM t1, t2
```

وهكذا تحصل على حاصل الضرب الديكارتي بسلام. لكن هذا سيكون آخر ما تريده في الحياة العملية، ولذلك قد تفكر جدياً في استغلال أي علاقة موجودة بين الجدولين، من أجل الربط بينهما بصورة أكثر فائدة. عادة، في الحياة العملية، سترغب في استعادة بيانات من جدولين تربط صفوفهما قيم مشتركة (راجع مفهوم العلاقة في الحلقة السابقة). هذا يقودك إلى الخيار الثاني في الربط بين الجدولين، وهو باشتراط تساوي الأرقام في الجدولين. تستطيع تحديد هذا الشرط للأكسس بكتابة عبارة SQL التالية:

```
SELECT t1.n, t2.n FROM t1, t2 WHERE t1.n = t2.n
```

ناتج هذه العبارة هو التالي:

2	2
3	3

هل أعجبك هذا الجواب؟ بغض النظر عن ذلك، ما قمت به هو عملية ربط JOIN بين الجدولين بطريقة خاصة، يسميها الأكسس عملية الربط الداخلي INNER JOIN Operation، ويخصص لها صيغة محددة في تحديد العلاقة، فتصبح العبارة السابقة كالتالي:

```
SELECT t1.n, t2.n FROM t1 INNER JOIN t2 ON t1.n = t2.n
```

هذه هي الصيغة الرسمية، ومن الجيد أن تتذكرها لأنها تشبه صيغ الطرق الباقية للربط. الناتج هو نفس ناتج WHERE السابقة، ويمكن أن نفكر به كالتالي: استعرض الحقلين الرقميين، ولكن فقط القيم التي لها ما يطابقها في الجدول الآخر.

إذا كانت قيم أحد الجداول لها مكانة خاصة في نفسك، وتريد استعراضها على كل حال، بغض النظر عن وجود ما يطابقها في الجدول الثاني، فبإمكانك ان تعمد إلى الخيار الثالث، وهو ما يسمى بالربط الخارجي OUTER JOIN. الربط الخارجي هو طريقة خاصة في الربط بين جدولين، بحيث يتم استعادة كل صفوف جدول، ثم البحث عن صفوف الجدول الثاني التي تتطابق فيها قيم الحقل المشترك مع الجدول الأول. قد يحدث أن تبقى بعض صفوف الجدول الأول بدون ما يقابلها من الجدول الثاني، وفي هذه الحالة تبقى الحقول الممثلة للجدول الثاني خالية. في حالة لم يكن هذا الكلام مفهوماً، سيصبح كذلك بإذن الله بعد أن نرى تطبيقه على جدولينا الصغيرين، لكن قبل ذلك، ينبغي أن أذكر أن الربط الخارجي نوعان، لأننا ذكرنا أنه تتم استعادة البيانات من جدول واحد بشكل أساسي، ثم ما يقابله من الجدول الثاني، وعلى هذا نتوقع أن تختلف النتيجة باختلاف الجدول الذي نختاره كأساس في الربط. لأننا نفترض الربط بين جدولين، فإنه قد تم اختيار موقع الجدولين للتمييز بينهما. إذا تم اختيار الجدول الأيمن، فإن الربط الأيمن RIGHT JOIN هو الذي يستخدم، وإلا فإن الربط الأيسر LEFT JOIN هو الذي يستخدم. لاحظ أن كلمة الخارجي قد تم حذفها اختصاراً فقط.

كتطبيق للربط الخارجي بالنسبة لجدولي الأرقام، قارن بين العبارتين التاليتين:

```
SELECT t1.n, t2.n FROM t1 RIGHT JOIN t2 ON t1.n = t2.n
```

ناتجها هو:

	1
2	2
3	3
	4
	5

```
SELECT t1.n, t2.n FROM t1 LEFT JOIN t2 ON t1.n = t2.n
```

والناتج هو:

0	
2	2
3	3
6	

لاحظ من فضلك هذه النقاط المهمة:

- إذا تم ربط جدولين بمفتاح أساسي في أحدهما، فينبغي لهذا المفتاح (الأجنبي في الجدول الآخر) ألا يحتوي قيماً لا توجد في المفتاح الأساسي، على خلاف المثال المركب هاهنا.
- العبارتان التاليتان متكافئتان:

```
SELECT t1.n, t2.n FROM t1 RIGHT JOIN t2 ON t1.n = t2.n
SELECT t1.n, t2.n FROM t2 LEFT JOIN t1 ON t1.n = t2.n
```

- لو تأملنا الجداول الناتجة في حالة الربط الخارجي، لوجدنا أن بنيتها تمهد لنا الحصول على القيم الموجودة في أحد الجدولين، ولا توجد في الآخر، وهو الفرق بين الجدولين (مطلوب شائع في تطبيقات قواعد البيانات). مثلاً، في المثال الأول، نستطيع معرفة الفرق بين t2 وt1، أي القيم الموجودة في الحقل المشترك في t2 ولا توجد في t1 باشتراط أن تساوي حقول t1 القيمة الخالية في العبارة المستخدمة بالمثال، كالتالي:

```
SELECT t2.n FROM t1 RIGHT JOIN t2 ON t1.n = t2.n WHERE t1.n IS NULL
```

تمرين: استخرج القيمتين 0 و6 الموجودتين في حقل t1 وليستا في حقل t2، باستخدام استعلام مناسب.

في الحلقة القادمة بإذن الله، نرى أمثلة أكثر عملية للاستعلام من أكثر من جدول...

أمثلة للاستعلام من أكثر من جدول

أبدأ هذه الحلقة بالاقتباس من الحلقة السابعة عشر، فقرة تلمس نقطة أحب التأكيد عليها ثانية هاهنا:

> اسمحوا لي قبل أن أشرع في موضوع هذه الحلقة أن أضيف بعض الملحوظات على موضوع العلاقات قبل أن أنسى، لأن هناك ما قد يثير حيرة المبتدئ عند الحديث عن العلاقات في سياقات مختلفة. سأوجز هذا في كلمات قليلة حتى لا يتشعب بنا الأمر ونستطيع العودة إلى الموضوع الأصلي... أنت تنشئ العلاقات بين الجداول باستخدام المفاتيح الأجنبية بمعنى أنك تتيح إمكانية الربط فيما بعد بين هذه الجداول من أجل استخراج المعلومات. إلى الآن لا يعلم برنامج إدارة قواعد البيانات DBMS (مثل الأكسس) شيئاً عن تلك العلاقات. فيما بعد، أنت تخبر DBMS بهذه العلاقات بطريقتين. إما عبر حفظ هذه العلاقات في قاعدة البيانات نفسها، وفي الأكسس مثلاً تقوم بذلك من خلال شاشة محرر العلاقات (أيقونة علاقات في شريط الأدوات عندما تكون في تبويب الجداول)؛ الهدف الرئيس من حفظ العلاقات بهذا الشكل هو تمكين DBMS من حفظ التكامل المرجعي بين الجداول، الذي سنتكلم عنه بإذن الله فيما بعد. الطريقة الثانية التي تستخدم فيها العلاقات هي للربط بين الجداول في عبارات SQL عند الاستعلام أو معالجة البيانات. برنامجDBMS يستخدم ما تخبره به من أجل الربط بين الجداول وإحضار أو تنفيذ المطلوب. المزيد عن ذلك بإذن الله عند الحديث عن أنواع الربط وعن لغة SQL. بالمناسبة، إذا استخدمت معالج الاستعلامات في الأكسس، فإنه يستفيد من العلاقات المحفوظة في الربط بين الجداول عند إنشاء الاستعلامات. هذا يكفي الآن، لنعد إلى موضوعنا...

بحمد الله قد أتينا فيما سبق على ما وعدت به بشأن الحديث عن التكامل المرجعي، وعن طرق الربط، لكن ما أود التنبيه عليه هنا هو ضرورة التمييز بين العلاقة واستخدامها. العلاقة تنشأ باستخدام الحقول المشتركة: بشكل عام المفتاح الأساسي في جدول وصورته (المفتاح الأجنبي) في الجدول الآخر يعرفان العلاقة بين الجدولين. عند استعادة بيانات فعلياً من الجدولين، لا بد لك من استخدام هذه العلاقة في عبارة SQL في واحدة من طرق الربط التي تحدثنا عنها للتو في الحلقة السابقة: JOIN {INNER | LEFT | RIGHT}. الرمز (|) يعني (أو)، حيث أنك تستخدم واحدة فقط من هذه العمليات في المرة الواحدة. كما أشرت في الفقرة أعلاه، تعريف العلاقة يسمح للأكسس بحفظ التكامل المرجعي، فلا يسمح بوجود قيم في المفتاح الأجنبي لم توجد بعد في المفتاح الأساسي مثلاً، وكذلك يسمح للأكسس باقتراح عمليات ربط مناسبة عند استخدام مصمم الاستعلامات. أما وجود العلاقة من الأساس فهو الذي يتيح الربط بين الجدولين من أجل استعادة البيانات منهما. من المفروض أن العلاقة موجودة طبيعياً، ولكن يتم تمثيلها بالمفاتيح كما سبق.

الاستعلام من أكثر من جدول: أمثلة

بالرجوع إلى مثالنا العتيد لجدول الموظفين، أعيد عرضه هاهنا لسهولة المتابعة:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠

الملاحظ في الحقل الأخير (EmpDept)، أن قسم الموظف قد تم التعبير عنه باستخدام رقم. هذا غير واقعي طبعاً، ولا بد أنك لاحظت مبكراً أن جدول الأقسام قد تم إغفال ذكره للتركيز على تعلم الاستعلام من جدول واحد أولاً. إذا كنت تتساءل عن سبب وجود جدول للأقسام فأرجو أن تراجع الدروس السابقة التي تتعلق بالتصميم والتسوية. على كل حال، الأقسام تشكل وحدات قائمة بذاتها، ولها جدول خاص مكون من حقلين كحد أدنى، نفرضه كالتالي:

dptNo	dptName
١٠	Accounting
٢٠	Research
٣٠	Sales
٤٠	Operations
٥٠	IT

العلاقة بين الجدولين تتمثل في حقل dptNo في جدول الأقسام، وهو المفتاح الأساسي في هذا الجدول، وصورته (EmpDept) في جدول الموظفين، وهو مفتاح أجنبي هنا. هذا المثال يوضح أيضاً نقطة قد يغفل عنها البعض: ليس شرطاً أن تتطابق أسماء المفتاحين في الجدولين، لكن المهم هو أن تكون قيمهما من نفس المجال (راجع الحلقتين 15 و22). نوع العلاقة هنا واحد إلى متعدد من جدول الأقسام إلى جدول الموظفين، وهذا يعني أن كل قسم قد يضم أكثر من موظف، لكن الموظف الواحد لا يعمل في أكثر من قسم.

أكثر مثال تحتاج فيه إلى الربط بين الجدولين هو ربما عند استعادة بيانات الموظفين، ومن ضمنها أقسامهم. المطلوب دوماً هو عرض اسم القسم وليس رقمه، وهذا يحتاج إلى الربط بين الجدولين من أجل استعادة اسم القسم من جدول الأقسام. يعرف الأكسس اسم قسم كل موظف لأن رقم القسم في سجل الموظف هو نفسه رقمه في جدول الأقسام. من المفيد أن نلاحظ هنا أنه لا بد لكل موظف من قسم، وهذا يعني أن حقل رقم القسم في جدول الموظفين لا بد أن يحوي قيمة، ولا يسمح له بالبقاء خالياً. هذا يعني أنه بإمكاننا الاعتماد على عملية الربط INNER JOIN بين الجدولين، لأن هناك دائماً قيمتين متطابقتين في حقلي رقم القسم في الجدولين. العبارة التالية تقوم بالمطلوب:

```
SELECT tblEmployee.EmpNo, tblEmployee.EmpName, tblEmployee.EmpHireDate, tblEmployee.EmpSal,
tblDepartment.dptName
FROM tblDepartment INNER JOIN tblEmployee ON tblDepartment.dptNo = tblEmployee.EmpDept
```

استخدمنا هنا أسماء الجداول كمعرفات قبل أسماء الحقول، وإن كنا غير ملزمين بذلك في هذه الحالة بسبب اختلاف أسماء الحقول في الجدولين، وبسبب أن اسم كل حقل معبر بالفعل عن الجدول الذي ينتمي إليه. الناتج من هذه العبارة هو:

EmpNO	EmpName	EmpHireDate	EmpSal	dptName
٧٧٨٢	CLARK	٠٩/٠٦/١٩٨١	٢٤٥٠	Accounting
٧٨٣٩	KING	١٧/١١/١٩٨١	٥٠٠٠	Accounting
٧٩٣٤	MILLER	٢٣/٠١/١٩٨٢	١٣٠٠	Accounting
٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	٨٠٠	Research
٧٥٦٦	JONES	٠٢/٠٤/١٩٨١	٢٩٧٥	Research
٧٧٨٨	SCOTT	١٩/٠٤/١٩٨٧	٣٠٠٠	Research
٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	١١٠٠	Research
٧٩٠٢	FORD	٠٣/١٢/١٩٨١	٣٠٠٠	Research
٧٤٩٩	ALLEN	٢٠/٠٢/١٩٨١	١٦٠٠	Sales
٧٥٢١	WARD	٢٢/٠٢/١٩٨١	١٢٥٠	Sales
٧٦٥٤	MARTIN	٢٨/٠٩/١٩٨١	١٢٥٠	Sales
٧٦٩٨	BLAKE	٠١/٠٥/١٩٨١	٢٨٥٠	Sales
٧٨٤٤	TURNER	٠٨/٠٩/١٩٨١	١٥٠٠	Sales
٧٩٠٠	JAMES	٠٣/١٢/١٩٨١	٩٥٠	Sales

مثال آخر أقل وضوحاً هو التالي: المطلوب استعراض بيانات كل الأقسام مع أعداد الموظفين في كل قسم. تحليل المطلوب يقودنا إلى النقاط التالية:

- بيانات الأقسام المطلوبة، تشمل الرقم والاسم، لا بد من استخراجها من جدول الأقسام.
- أعداد الموظفين لا نستطيع استخراجها إلا من جدول الموظفين.
- المطلوب هو عدد الموظفين في كل قسم، وهذا يعني تجميع الموظفين حسب الأقسام. نعم، نحتاج هنا إلى GROUP BY.

قد تبدأ بعبارة كالتالي:

```
SELECT tblDepartment.dptNo, tblDepartment.dptName, Count (tblEmployee.EmpNO) AS
CountOfEmpNO
FROM tblDepartment INNER JOIN tblEmployee ON tblDepartment.dptNo = tblEmployee.EmpDept
GROUP BY tblDepartment.dptNo, tblDepartment.dptName
```

وناتجها هو:

dptNo	dptName	CountOfEmpNO
١٠	Accounting	٣
٢٠	Research	٥
٣٠	Sales	٦

تبدو هذه النتيجة جيدة ومضبوطة. للأسف، قد أغفلت هذه العبارة الأقسام التي لا تحوي موظفين بعد (ربما هي قيد التأسيس). المشكلة هي، كما توقعت أنت تماماً، هي في طريقة الربط. الربط الداخلي لا ينفع هنا، لأن هناك قيماً في حقل رقم القسم في جدول الأقسام لا تقابلها قيم في حقل رقم القسم في جدول الموظفين. الذي نحتاجه هو نوع من الربط الخارجي، بحيث نحصل على كل صفوف جدول الأقسام، وما يقابلها من جدول الموظفين. تبعاً لترتيب ذكر الجدولين، نستخدم ربطاً خارجياً أيسر أو أيمن كالتالي:

```
SELECT tblDepartment.dptNo, tblDepartment.dptName, Count (tblEmployee.EmpNO) AS
CountOfEmpNO
FROM tblDepartment LEFT JOIN tblEmployee ON tblDepartment.dptNo = tblEmployee.EmpDept
GROUP BY tblDepartment.dptNo, tblDepartment.dptName
```

```
SELECT tblDepartment.dptNo, tblDepartment.dptName, Count (tblEmployee.EmpNO) AS
CountOfEmpNO
FROM tblEmployee RIGHT JOIN tblDepartment ON tblDepartment.dptNo = tblEmployee.EmpDept
GROUP BY tblDepartment.dptNo, tblDepartment.dptName
```

العبارتان أعلاه متكافئتان. لاحظ من فضلك أن استخدام العبارة الأولى مثلاً، لكن بالربط الأيمن (كل الصفوف من جدول الموظفين، مع ما يقابلها من جدول الأقسام) يكافئ استخدام الربط الداخلي السابق بيانه. الناتج من العبارتين السابقتين هو:

dptNo	dptName	CountOfEmpNO
١٠	Accounting	٣
٢٠	Research	٥
٣٠	Sales	٦
٤٠	Operations	٠
٥٠	IT	٠

مثال أخير. كما أشرنا في الحلقة السابقة، فإنه من الممكن استخراج الفرق بين جدول الأقسام وجدول الموظفين (الأقسام الموجودة في جدول القسام بدون موظفين في جدول الموظفين) باستخدام العبارة التالية:

```
SELECT tblDepartment.*
FROM tblDepartment LEFT JOIN tblEmployee ON tblDepartment.dptNo = tblEmployee.EmpDept
WHERE tblEmployee.EmpNO Is Null
```

الناتج هو طبعاً:

dptNo	dptName
٤٠	Operations
٥٠	IT

في الحلقات القادمة بإذن الله نستعرض الاستعلامات الفرعية، ودوال SQL، ثم يمكن أن نمر باختصار على عبارات معالجة البيانات من إضافة وحذف وتعديل إن يسر الله... لعلك لاحظت أنه ليس هناك تمرين في هذه الحلقة، فيمكنك أن تتمرن بمراجعة الحلقات السابقة، أو التحضير للحلقات القادمة، المهم أن تقرأ، تفهم، تجرب، تفهم أكثر أو تصحح الفهم.

الاستعلامات الفرعية SUBQUERIES

الربط بين جدولين ليس هو الطريقة الوحيدة لإشراك أكثر من جدول في استعلام. توفر SQL ميزة مرنة للغاية للاستفادة من البيانات في أكثر من جدول لتنفيذ استعلام. هذه الميزة تتمثل في إمكانية تنفيذ استعلامات فرعية إلى جانب الاستعلامات الأصلية، والاستفادة من مردودات هذه الاستعلامات الفرعية في عمل الاستعلام الأصل.

فكرة الاستعلام الفرعي

الاستعلام الفرعي هو عبارة SELECT اعتيادية، لكنك تستطيع استخدامها داخل عبارة SQL أخرى، إما مكان حقل من الحقول في عبارة SELECT، وهنا نتوقع أن تعيد العبارة الفرعية قيمة لها علاقة مع بقية الحقول في العبارة الأصلية، وإما أن تستخدم ناتج العبارة الفرعية في مقارنة مع حقول العبارة الأصلية في مقطع WHERE أو HAVING. بل إنك تستطيع استخدام استعلام فرعي مكان جدول في جزء FROM من عبارة SELECT الأصلية.

من أبسط الأمثلة لتوضيح هذه الفكرة، مثال استخراج الموظف صاحب أكبر راتب في الشركة. نفذنا هذا المطلوب من قبل أو نحوه باستخدام TOP مع الترتيب التنازلي. بالإمكان المقارنة المباشرة مع أكبر راتب في مقطع WHERE إذا كنا نعلم هذا الراتب. يمكن معرفة الراتب باستعلام يستخدم الدالة MAX. لدينا هاهنا استعلامان، استعلام يستخرج الراتب الكبر، وآخر يستخرج الموظف صاحب هذا الراتب الكبر بمقارنة راتبه مع هذا الراتب. المطلوب هو تنفيذ الاستعلامين في عبارة واحدة، والاستفادة من نتيجة أحدهما في تنفيذ الآخر. في مثل هذه الحالات، تأتي الاستعلامات الفرعية لتقول بكل ثقة: لو سمحتم، أفسحوا الطريق!

استخدام الاستعلامات الفرعية ضمن قائمة حقول SELECT

أحياناً، تكون إمكانية استخدام استعلام كامل ضمن الحقول المستعادة مفيدة جداً. في هذا الاستعلام تستطيع جلب بيانات من جدول آخر. في بعض الحالات يكون أداء نفس المهمة بواسطة الربط التقليدي بين الجداول (JOIN) واضحاً، لكن في حالات أخرى يكون صعباً تصور القيام بنفس المهمة التي يمكن أن يؤديها الاستعلام الفرعي.

كمثال على الحالة الأولى، وحتى ترى أمامك عبارة SQL حية، فلنفرض أنك تريد الاستعلام عن بيانات الموظفين، ومع كل موظف تاريخ تعيين مديره. في جدول الموظفين الذي تعاملنا معه حتى الان، هناك رقم مدير الموظف ضمن بيانات كل موظف. ما نريده هو أن نستعيد مع كل حقول كل صف من صفوف الموظفين حقلاً إضافياً هو تاريخ تعيين مدير هذا الموظف. نستطيع معرفة المدير بالطبع من رقمه الذي هو أحد حقول صف الموظف. لكن من أجل الخطوة الإضافية في معرفة تاريخ تعيين هذا المدير نستطيع استخدام استعلام كامل يجلب التاريخ بناء على رقم المدير (وهو رقم موظف في النهاية) من جدول الموظفين نفسه. هذا الاستعلام هو استعلام فرعي سنضعه ضمن قائمة الحقول المستعادة كالتالي:

```
SELECT e1.EmpNO, e1.EmpName, e1.EmpHireDate, e1.EmpManager, (SELECT e2.EmpHireDate FROM
tblEmployee AS e2 WHERE e2.EmpNo = e1.EmpManager) AS [ManagerHiredate]
FROM tblEmployee AS e1
```

السر في العبارة السابقة هو التمييز بين جدول الموظفين داخل الاستعلام الفرعي وفي الاستعلام الأساسي باستخدام أسماء مستعارة (e1 للاستعلام الأساسي وe2 للاستعلام الفرعي)، بحيث تم ربط جدول الموظفين في الاستعلام الفرعي بجدول الموظفين في الاستعلام الأساسي بمساواة رقم الموظف في الاستعلام الفرعي برقم مدير الموظف في الاستعلام الأساسي. لاحظ هنا أن الاستعلام الفرعي يتم تنفيذه لكل صف. قمت باستجلاب أرقام مديري الموظفين وتواريخ تعيين الموظفين من أجل سهولة التأكد من نتيجة الاستعلام (قارن تاريخ تعيين مدير بتاريخ تعييم موظف يحمل نفس رقم المدير). ناتج هذه العبارة هو التالي:

	EmpNO	EmpName	EmpHireDate	EmpManager	ManagerHiredat
▶	٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	٧٩٠٢	٠٣/١٢/١٩٨١
	٧٤٩٩	ALLEN	٢٠/٠٢/١٩٨١	٧٦٩٨	٠١/٠٥/١٩٨١
	٧٥٢١	WARD	٢٢/٠٢/١٩٨١	٧٦٩٨	٠١/٠٥/١٩٨١
	٧٥٦٦	JONES	٠٢/٠٤/١٩٨١	٧٨٣٩	١٧/١١/١٩٨١
	٧٦٥٤	MARTIN	٢٨/٠٩/١٩٨١	٧٦٩٨	٠١/٠٥/١٩٨١
	٧٦٩٨	BLAKE	٠١/٠٥/١٩٨١	٧٨٣٩	١٧/١١/١٩٨١
	٧٧٨٢	CLARK	٠٩/٠٦/١٩٨١	٧٨٣٩	١٧/١١/١٩٨١
	٧٧٨٨	SCOTT	١٩/٠٤/١٩٨٧	٧٥٦٦	٠٢/٠٤/١٩٨١
	٧٨٣٩	KING	١٧/١١/١٩٨١		
	٧٨٤٤	TURNER	٠٨/٠٩/١٩٨١	٧٦٩٨	٠١/٠٥/١٩٨١
	٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	٧٧٨٨	١٩/٠٤/١٩٨٧
	٧٩٠٠	JAMES	٠٣/١٢/١٩٨١	٧٦٩٨	٠١/٠٥/١٩٨١
	٧٩٠٢	FORD	٠٣/١٢/١٩٨١	٧٥٦٦	٠٢/٠٤/١٩٨١
	٧٩٣٤	MILLER	٢٣/٠١/١٩٨٢	٧٧٨٢	٠٩/٠٦/١٩٨١

لنتناول مثالاً آخر حيث تكون فائدة الاستعلام الفرعي أكثر وضوحاً. المطلوب هو استعادة بيانات كل الموظفين، لكن مع ترقيم الصفوف المستعادة تسلسلياً، رغم عدم وجود حقل يرقم الموظفين تسلسلياً حسب رقم الموظف مثلاً في الجدول. الفكرة هنا في الاستعلام الفرعي هي عد صفوف الموظفين التي تصغر أرقامهم أو تساوي رقم الموظف في الصف الحالي. وهكذا حين يتم تنفيذ هذا الاستعلام مع كل صف نحصل على ترتيب الموظف في كل صف. الموظف الأول عدد الموظفين الذين تصغر أرقامهم أو تساوي رقمه هو فقط واحد، وهو نفسه رقمه هو. الموظف الثاني عدد الموظفين الذين تصغر أرقامهم أو تساوي رقمه هو اثنان، رقمه هو ورقم الموظف الأول، وهكذا. العبارة هي:

```
SELECT (SELECT COUNT (*) FROM tblEmployee AS e2 WHERE e2.EmpNo <= e1.EmpNo) AS Serial,
EmpNo, EmpName, EmpHireDate, EmpSal
FROM tblEmployee AS e1
```

وناتج العبارة هو:

Serial	EmpNO	EmpName	EmpHireDate	EmpSal
١	٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	٨٠٠
٢	٧٤٩٩	ALLEN	٢٠/٠٢/١٩٨١	١٦٠٠
٣	٧٥٢١	WARD	٢٢/٠٢/١٩٨١	١٢٥٠
٤	٧٥٦٦	JONES	٠٢/٠٤/١٩٨١	٢٩٧٥
٥	٧٦٥٤	MARTIN	٢٨/٠٩/١٩٨١	١٢٥٠
٦	٧٦٩٨	BLAKE	٠١/٠٥/١٩٨١	٢٨٥٠
٧	٧٧٨٢	CLARK	٠٩/٠٦/١٩٨١	٢٤٥٠
٨	٧٧٨٨	SCOTT	١٩/٠٤/١٩٨٧	٣٠٠٠
٩	٧٨٣٩	KING	١٧/١١/١٩٨١	٥٠٠٠
١٠	٧٨٤٤	TURNER	٠٨/٠٩/١٩٨١	١٥٠٠
١١	٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	١١٠٠
١٢	٧٩٠٠	JAMES	٠٣/١٢/١٩٨١	٩٥٠
١٣	٧٩٠٢	FORD	٠٣/١٢/١٩٨١	٣٠٠٠
١٤	٧٩٣٤	MILLER	٢٣/٠١/١٩٨٢	١٣٠٠

لاحظ أنك لا تستطيع استعادة أكثر من حقل واحد فقط في الاستعلام الفرعي هنا. كما أن أي استعلام فرعي يجب أن يكون محصوراً بين قوسين.

استخدام الاستعلامات الفرعية في مقطع WHERE (أو HAVING)

استخدام الاستعلامات الفرعية في شروط المقارنة متنوع، وتوجد العديد من الكلمات المحجوزة التي تسمى predicates، وتحدد طريقة معينة لمقارنة الناتج من الاستعلام الفرعي مع قيم حقول الاستعلام الأساسي. في البدء، دعنا نر كيف يمكن استخدام عوامل المقارنة الرياضية المألوفة مع الاستعلامات الفرعية. إذا عدنا إلى المثال الأول في هذه الحلقة، والذي يطلب استعادة الموظف صاحب أكبر راتب، فإن الاستعلام الفرعي الذي نحتاجه يعيد أكبر راتب، وتتم مقارنة هذا الراتب مع راتب كل موظف بواسطة عامل المقارنة (=) من أجل استخراج الموظف صاحب هذا الراتب. طبعاً إذا كان هناك أكثر من موظف بنفس الراتب فإن كلاً منهم سيتم استعراضه، لكن الاستعلام الفرعي يجي ألا يعيد أكثر من قيمة واحدة فقط. العبارة المطلوبة هي:

```
SELECT EmpNo, EmpName, EmpHireDate, EmpSal
FROM tblEmployee
WHERE EmpSal = (SELECT MAX (EmpSal) FROM tblEmployee)
```

لاحظ أن الاستعلام الفرعي في هذه الحالة لا يرتبط بالاستعلام الأساسي. إنه يستعيد قيمة مستقلة عن الاستعلام الأساسي، لكننا نستخدم هذه القيمة من أجل تصفية صفوف الاستعلام الأساسي. الناتج هو صف وحيد كما هو متوقع:

EmpNO	EmpName	EmpHireDate	EmpSal
٧٨٣٩	KING	١٧/١١/١٩٨١	٥٠٠٠

طريقة أخرى هي مع الكلمة IN. وقد تناولنا فيما سبق أن هذه الكلمة تستخدم في المقارنات مع قائمة من القيم، وكنا نكتب هذه القائمة يدوياً، لكن الحقيقة أننا نستطيع استعادة قائمة من القيم من قاعدة البيانات عبر استعلام. المثال على ذلك باستخدام الجدولين الصغيرين الذين نتعلم معهما قد لا يكون معبراً، إذ أن الفائدة من الاستعلام الفرعي هي في استحضار قائمة من القيم من جدول آخر يصعب ربطه مباشرة مع الجدول مصدر الاستعلام الأساسي. لكن من باب توضيح الطريقة، لنفرض أن المطلوب هو استعادة الموظفين من قسمي المبيعات والبحوث، وأننا لا نعلم رقمي هذين القسمين. من الممكن كتابة العبارة التالية:

```
SELECT EmpNo, EmpName, EmpHireDate, EmpSal
FROM tblEmployee
WHERE EmpDept IN (SELECT dptNo from tblDepartment WHERE dptName IN ("Sales", "Research"))
```

تم هنا استخدام IN بالطريقتين: استعلام فرعي، وقيم يدوية. لاحظ أن الاستعلام الفرعي لا بد أن يعيد حقلاً مفرداً لأن المقارنة تتم مع حقل مفرد. ناتج العبارة هو:

EmpNO	EmpName	EmpHireDate	EmpSal
٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	٨٠٠
٧٤٩٩	ALLEN	٢٠/٠٢/١٩٨١	١٦٠٠
٧٥٢١	WARD	٢٢/٠٢/١٩٨١	١٢٥٠
٧٥٦٦	JONES	٠٢/٠٤/١٩٨١	٢٩٧٥
٧٦٥٤	MARTIN	٢٨/٠٩/١٩٨١	١٢٥٠
٧٦٩٨	BLAKE	٠١/٠٥/١٩٨١	٢٨٥٠
٧٧٨٨	SCOTT	١٩/٠٤/١٩٨٧	٣٠٠٠
٧٨٤٤	TURNER	٠٨/٠٩/١٩٨١	١٥٠٠
٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	١١٠٠
٧٩٠٠	JAMES	٠٣/١٢/١٩٨١	٩٥٠
٧٩٠٢	FORD	٠٣/١٢/١٩٨١	٣٠٠٠

تجدر أيضاً الإشارة إلى أنه يمكن استخدام NOT IN لاستثناء القيم التي يعيدها الاستعلام الفرعي من الاستعلام الأساسي.

هنالك أيضاً كما أشرت بضع كلمات محجوزة تستخدم مع الاستعلامات الفرعية خاصة. هذه الكلمات هي ALL, ANY, SOME, EXISTS (NOT EXISTS). تستطيع استخدام EXISTS لاستعادة صف في الاستعلام الأساسي في حالة كان الاستعلام الفرعي يعيد صفاً أو أكثر، فهي تستخدم للتأكد من (وجود) قيم في الاستعلام الفرعي. مثال على ذلك الاستعلام عن الأقسام بشرط وجود موظفين يزيد راتبهم عن 3000 في القسم. العبارة المستخدمة قد تشبه التالي:

```
SELECT tblDepartment.* FROM tblDepartment
WHERE EXISTS (SELECT tblEmployee.* FROM tblEmployee WHERE EmpSal >= 3000 AND
tblEmployee.EmpDept = tblDepartment.dptNo)
```

لاحظ كيف ربطنا في الاستعلام الفرعي بيم رقم القسم في جدول الأقسام (الاستعلام الأساسي)، ورقم القسم في جدول الموظفين (الاستعلام الفرعي). في حالةوجود أي موظف بالشرط المذكور في الاستعلام الفرعي، فإنه تتم استعادة القسم في الاستعلام الأساسي. الناتج من العبارة السابقة هو:

dptNo	dptName
١٠	Accounting
٢٠	Research

مثال آخر، الاستعلام عن العملاء في حالة وجود مشتريات لهم في الشهر الأخير. الاستعلام الأساسي يستحضر بيانات العميل، والاستعلام الفرعي يربط كل صف من جدول العملاء بجدول الفواتير، ويتأكد من وجود أي صفوف مستعادة من هذا الجدول لكل عميل في الشهر الأخير؛ فهو عبارة عن استعلام عن فواتير العميل الحالي في الاستعلام الأساسي بالنسبة للشهر الأخير. يمكن التأكد من عدم وجود صفوف مستعادة من الاستعلام الفرعي لاستعادة الصف في الاستعلام الأساسي باستخدام NOT، كما أرجو أن تلاحظ انه بإمكاننا باستخدام EXISTS فقط استعادة أكثر من حقل في الاستعلام الفرعي.

تستخدم ALL لمقارنة قيم حقل في الاستعلام الأساسي بكل القيم المستعادة من استعلام فرعي.المقارنة قد تتم باستخدام عوامل المقارنة الرياضية. مثلاً، تريد استعادة الموظف من قسم البحوث الذي يقل راتبه عن رواتب كل الموظفين في قسم المحاسبة. تستخدم ALL مع الاستعلام الفرعي كالتالي:

```
SELECT EmpNo, EmpName, EmpHireDate, EmpSal
FROM tblDepartment INNER JOIN tblEmployee ON tblDepartment.dptNo=tblEmployee.EmpDept
WHERE dptName="Research"
AND EmpSal < ALL (SELECT EmpSal FROM tblEmployee WHERE EmpDept IN (SELECT dptNo FROM
tblDepartment WHERE dptName = "Accounting"))
```

الاستعلام الفرعي يعيد رواتب الموظفين في قسم المحاسبة. الاستعلام الأساسي يعيد بيانات الموظفين من قسم البحوث، لكن بشرط. تتم مقارنة حقل الراتب مع كل القيم المعادة من الاستعلام الفرعي، فإذا كان شرط أن الراتب أصغر من (كل) الرواتب في الاستعلام الفرعي فإن الموظف تتم استعادة بياناته في الاستعلام الأساسي. لاحظ أيضاً أننا استخدمنا هنا IN داخل الاستعلام الفرعي من أجل الوصول إلى اسم القسم في جدول الأقسام، في حين استخدمنا الربط بين الجدولين من أجل الوصول إلى اسم القسم في الاستعلام الأساسي. جعلت الأمر هكذا من أجل أن تدرك تنوع الطرق الممكنة. الناتج من العبارة السابقة، وهو موظفي البحوث الذين تقل رواتبهم عن رواتب كل موظفي المحاسبة هو:

	EmpNO	EmpName	EmpHireDate	EmpSal
▶	٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	٨٠٠
	٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	١١٠٠
*				

الكلمتان SOME وANY مترادفتان في المعنى. وتقومان بمقارنة قيم حقل في الاستعلام الأساسي، بقيم حقل في الاستعلام الفرعي، فإذا كانت المقارنة ناجحة ولو بقيمة واحدة من الاستعلام الفرعي، تمت استعادة الصف في الاستعلام الأساسي. هذا يعني أنه باستخدامهما نستطيع استعادة الصفوف من الاستعلام الأساسي التي تحقق شرط المقارنة مع (أي) أو (بعض) القيم المعادة في الاستعلام الفرعي. هذا على خلاف ALL، التي تشترط تحقق شرط المقارنة مع (كل) القيم المعادة في الاستعلام الفرعي. المثال على ذلك هو استعادة بيانات الموظفين الذين يقل راتبهم عن راتب أي موظف في قسم آخر. الصيغة تشبه تماماً صيغة ALL، لذا أترك تنفيذ المثال كتمرين.

هذه هي الطرق الأساسية لاستخدام الاستعلامات الفرعية. ربما كان يجدر بي التنويه أيضاً إلى أنك قد تستخدم استعلاماً مؤقتاً كمصدر للاستعلام بعد الكلمة FROM كما في العبارة التالية:

```
SELECT EmpNo, EmpName
FROM (SELECT EmpNo, EmpName FROM tblEmployee WHERE EmpDept = 10)
```

في هذه العبارة، مصدر الاستعلام هو استعلام آخر مؤقت يعيد الموظفين في القسم رقم 10. تستطيع الاستغناء عن هذا النوع بإنشاء استعلامات (رسمية)، ثم استخدامها كمصدر مع بقية الجداول.

قبل أن أنهي نقاشنا عن الاستعلامات الفرعية، من المهم أن تدرك هنا أن استخدام هذه الميزة قد يكون مفيداً جداً في أحوال خاصة. ربما لاحظت قلة استخدام هذه الاستعلامات الفرعية، وهذا يعود لأسباب. بعضها يعود ببساطة إلى غفلة بعض المبتدئين عنها، ربما لأنهم لا يصلون عادة في القراءة إلى ما بعد الأساسيات الأولية، أو يصلون ولا يبذلون الجهد الكافي لاستيعابها. لكن هناك بعض الأسباب الوجيهة التي تتعلق بالأداء. قد يكون أداء الاستعلامات التي تحوي استعلامات فرعية أقل (من حيث السرعة) من أداء استعلامات لا تعتمد عليها. كمثال على ذلك، عرفنا في الحلقة السابقة أن الربط الخارجي قد يستخدم لاستخراج الفرق في حقل بين جدولين، مثل الأرقام التي توجد في حقل بالجدول الأول ولا توجد في حقل مشابه بالجدول الثاني. من الواضح أن هذا الفرق يمكن تحديده بسهولة باستخدام NOT IN، إذ يمكن كتابة استعلام أساسي يستعيد قيم حقل من جدول بشرط عدم وجودها في قائمة القيم المستعادة بالاستعلام الفرعي. لكن بشكل عام، أداء الربط الخارجي أفضل من أداء الاستعلام الفرعي.

هذا كل شيء في هذه الحلقة، فإن لم تكن قد استوعبت بعض النقاط، فهذا طبيعي، عد بعد فترة، واقرأ مرة أخرى، ثم حاول أن تدرس الأمثلة، فإن لم ينفع ذلك، حاول أن تقرأ من مصدر آخر أوضح في الشرح. المهم، حاول أن تدفع نفسك خطوة إلى الأمام في الطريق إلى إجادة الحديث بلغة قواعد البيانات SQL.

دوال SQL

مقدمة

عندما نحفظ البيانات في الجداول، فإن ذلك بغرض الحفظ من حيث هو، شيء مثل التوثيق، ولكنه لا يعني بالضرورة أننا سنستخدم هذه البيانات كما هي فيما بعد. إن ذلك يشبه أن تضع الطعام في الثلاجة من أجل استخدامه لاحقاً. قليل من أنواع الطعام ستتناولها كما هي من الثلاجة، بعضها قد ترتبه في الصحن من أجل سهولة التناول أو حتى من قبيل التزيين وفتح الشهية، والكثير على الأرجح سيخضع لعمليات طويلة من تقطيع وطبخ وإعداد (أرجو ألا تتحمس الآن وتقوم لتتفقد الثلاجة). في عالم الحاسوب، نسمي هذه العمليات (معالجة)، ونحن نعالج البيانات من أجل تحويلها إلى صورة قابلة للأكل، أقصد للإفادة، ونسمي الصورة الناتجة (معلومات). رأينا سابقاً بعض صور المعالجة من تجميع وترتيب وربط للبيانات المتعلق بعضها ببعض من عدة جداول. في أثناء ذلك، مررنا على مفهوم مهم للغاية، يعد واحداً من أهم وسائل المعالجة للبيانات: استخدام الدوال في تطبيق عمليات معينة على البيانات. من الضروري هنا أن نعيد التنبيه على أن المعالجة بشكل عام، في سياق SQL، تتم على حقول (أعمدة)، وليس على وحدات أخرى مثل المتغيرات أو الملفات التي تجدها في سياقات أخرى. كما قد يجدر بي التذكير بمفهوم الدالة من أجل أن نطمئن إلى أننا نسير في نفس الخط.

تذكير سريع بمفهوم الدالة

الدالة هي في النهاية برنامج مستقل تمت كتابته من أجل هدف محدد. هذا الهدف هو في الغالب تنفيذ عملية ما على بعض القيم، واحتساب النتيجة. القيم التي تتم عليها المعالجة تعطى للدالة من قبل المبرمج (مستخدم الدالة) وتسمى معاملات parameters الدالة. القيمة الناتجة هي التي تهم المبرمج. لاحظ أن بعض الدوال لا يستقبل أي قيم، ولكنه قد يعيد قيمة. مثلاً، هناك دالة تستدعيها من أجل معرفة تاريخ اليوم؛ لا تستقبل هذه الدالة أي معاملات، لكنها تعيد قيمة هي تاريخ اليوم. مرة أخرى، هذه الدالة هي برنامج سبقت كتابته وترجمته، وهو جاهز في صيغته التنفيذية في مكتبة ما (ملف ما من نوع خاص)، واستدعاء الدالة التي تعيد قيمة لا يعدو استخدامها (كتابة اسمها مع معاملاتها بين قوسين) في تعبير ما expression مثل عملية حسابية أو منطقية، ولأن الدالة تعيد قيمة، فإنك تستخدمها مثل ما تستخدم أي متغير. يجب الانتباه هنا إلى نوع القيمة التي تعيدها الدالة من أجل الاستخدام الصحيح للدالة. كثيراً ما تستقبل القيم التي تعيدها الدوال في متغيرات، ولكل متغير نوع بيانات، لذا ينبغي أن يتوافق نوع المتغير مع نوع القيمة المتوقعة من الدالة. سنحن لا نتحدث هنا عن البرامج التي لا تعيد قيمة، وتسمى في بعض اللغات بشكل مختلف عن الدوال؛ في VBA مثلاً، تسمى هذه البرامج Sub عوضاً عن Function.

لماذا الدوال؟

هناك الكثير من العمليات التي نحتاج إليها، ويحتاج إليها غيرنا، مراراً وتكراراً. هنا، يكون من الحكمة كتابة العملية مرة واحدة بشكل جيد، والتأكد من فعاليتها، ثم استخدامها فيما بعد (من على الرف). بدلاً من أن يكتب كل واحد دالته بنفسه، من حسن الحظ أن منتجي مترجمات لغات البرمجة، وفي حالتنا، منتجي برامج إدارة قواعد البيانات، من ضمن توفيرهم لمتطلبات لغة قواعد البيانات في حدود المعايير الموضوعة من قبل منظمات متخصصة، يقومون بتوفير عدد من الدوال الجاهزة التي يغلب استخدامها في تطبيقات قواعد البيانات، وفي استخدامات إدارة قواعد البيانات. بل إن كل منتج قد يتعدى الحد الأدنى المطلوب، ويمتاز بتوفير دوال غير متوفرة في المنتجات الأخرى. تخيل لو كان مطلوباً منك أن تكتب برامجاً لكل عمليات المعالجة التي تحتاجها على البيانات، صغيرة أو كبيرة، كيف سيكون الوضع. أكثرنا بكل بساطة لا يملك القدرة أو الوقت أو كليهما، ولهذا لن تكون تطبيقاتنا كما هي عليه الآن، هذا إن وجدت.

أنواع الدوال

قبل أن أتحدث عن الدوال من جهة اختلاف أنواع البيانات، أرى من المهم التنبيه على أنني لا تحدث هنا الدوال التجميعية، التي تشكل نوعاً قائماً بذاته. تكلمنا عن هذه الدوال فيما سبق، ورأينا استخدامها في سياق التجميع مع المقطع GROUP BY. الفرق بين هذه الدوال وبقية الدوال، أنها عند تطبيقها على حقل، تدخل كل قيم الحقل في الاحتساب، وهذا يعني أن كل صفوف الجدول، أو كل صفوف المجموعة الواحدة في حالة وجود تجميع (الحالة الأغلب) تدخل في الاحتساب، وتنتج قيمة واحدة فقط عن كامل الجدول أو عن كل مجموعة في التجميع. مثلاً عند وجود ثلاثة أقسام في جدول من مئة صف مثلاً، وتم التجميع حسب القسم، فإن دالة مثل COUNT على حقل رقم الموظف تعيد فقط ثلاثة قيم (في ثلاثة صفوف)، هي عدد الموظفين في كل قسم من الأقسام الثلاثة. أما الدوال غير التجميعية، فإنها تطبق على كل قيمة في الحقل، في كل صف، بدون أي تجميع. بمعنى أنها تؤثر على قيم الحقل المفردة في كل صف، ولا تختصر عدد الصفوف. ولذلك، فإنها تستدعى (يتم تنفيذ البرنامج الذي تمثله الدالة) مع كل صف يتم استعادته من الجدول أو الجداول موضوع الاستعلام. يمكنك استخدام الدوال غير التجميعية في قائمة حقول الأمر SELECT وفي غيرها من المقاطع، مثل المقطع WHERE كما سنرى لاحقاً بإذن الله.

تختلف البيانات التي تحفظ في قاعدة البيانات، ولهذا تختلف الدوال التي تعالج هذه البيانات. بشكل عام، هناك دوال تعالج البيانات النصية، ودوال تعالج البيانات العددية (على تنوع الأعداد)، ودوال لمعالجة التواريخ والأوقات، لكن هناك أيضاً دوالّ لا يمكن بشكل دقيق تصنيفها تحت واحد من هذه البنود، مثلاً دوال تتعامل مع القيم الخالية. هناك الكثير من هذه الدوال المشتركة في لغة SQL بين كل منتجي برامج إدارة قواعد البيانات، على الرغم من وجود الكثير من التميز أيضاً. في Jet SQL (تطبيق مايكروسوفت للغة SQL في أكسس)، يتم استخدام دوال VBA (يتم استدعاء الدوال في مكتبات لغة VBA). آخر ما أقصده هنا هو توفير مرجع لهذه الدوال، وإنما القصدهو مجرد تقديم الفكرة والتنويه على وجود هذه الدوال، والحث على البحث عنها في مظانها، واستخدامها. يمكن البحث عن هذه الدوال من أجل مراجعة تفاصيلها من حيث الوظيفة، والمعاملات المطلوبة، والنتيجة المعادة، وطريقة الاستخدام في ملفات المساعدة أو في عدد ضخم من الكتب المتوفرة، أو في عالم النت الرحيب (الرحيب لدرجة الضياع).
فيما يلي من سطور بإذن الله، أعرض أمثلة لبعض الدوال المتوفرة في الأكسس (في VBA بشكل دقيق)، مرتبة حسب أنواع الدوال.

الدوال النصية

عند التعامل مع النصوص (سلاسل الحروف، مثل الأسماء)، فإن ما يهمك هو عمليات من قبيل عدد حروف نص أو البحث عن حرف معين أو نص فرعي معين في نص آخر، أو لصق نص مع آخر، أو استقطاع جزء من نص، أو التخلص من الفراغات أو استبدال حرف بآخر أو تغيير حالة حرف إنجليزي مثلاً، وغير ذلك. هناك دالة معدة لكل عملية من هذه العمليات (وأكثر). أنت ترى أن المبرمجين أيضاً يتمتعون بالرفاهية في هذه الحياة. والآن، دعنا نلمس هذه الرفاهية لمسة خفيفة...

هذا هو جدولنا الذي أبى أن يفارقنا في هذه الحلقات، أكرره هنا حتى لا تضطر إلى الرجوع صفحات للخلف من أجل معاينة نتيجة عبارات SQL التي نستخدمها:

	EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
▶	٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
	٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
	٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
	٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
	٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
	٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠		٣٠
	٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
	٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
	٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
	٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
	٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
	٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
	٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
	٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
*							

Len(), Left(), UCase(), LCase() وقصص أخرى

من أجل مثال يجمع عدداً من دوال معالجة النصوص، دعنا نفترض أننا نريد الاستعلام عن أسماء وأرقام هواتف الموظفين بشروط خاصة: الحرف الأول من الاسم يجب أن يكون حرفاً كبيراً capital أو upper case، وبقية الأحرف صغيرةlower case ، كما أن أرقام الهاتف ينبغي أن تنسق حسب التالي: 0123034880 يصبح 012-303-4880.
انظر إلى العبارة التالية:

```
SELECT UCase(Left(EmpName,1)) & LCase(Right(EmpName, Len(EmpName)-1)) AS Name,
Left(EmpPhone,3) & "-" & Mid(EmpPhone,4,3) & "-" & Mid(EmpPhone,7) AS Phone1,
Format(EmpPhone,"0##-###-####") AS Phone2
FROM tblEmployee
```

استخدمنا من أجل تنسيق رقم الهاتف طريقتين، الطريقة الطويلة باستخدام دالتين من أجل توضيح استخدامهما، والطريقة الأقصر باستخدام الدالة Format. اتضح أن الدالة الأخيرة أفضل من ناحية التعامل مع القيم الخالية، ولذلك كان الخرج أفضل، كما هو واضح في ناتج العبارة التالي:

	Name	Phone١	Phone٢
▶	Smith	--	
	Allen	٠١٢-٣٠٣-٤٨٨٠	٠١٢-٣٠٣-٤٨٨٠
	Ward	٠١٧-٨٨٩-٦٢٠١	٠١٧-٨٨٩-٦٢٠١
	Jones	--	
	Martin	٠١٢-٢٥٥-٠٠٥٧	٠١٢-٢٥٥-٠٠٥٧
	Blake	--	
	Clark	--	
	Scott	--	
	King	--	
	Turner	٠--	٠٠٠-٠٠٠-٠٠٠٠
	Adams	--	
	James	--	
	Ford	--	
	Miller	--	
*			

هاهنا العديد من الدوال؛ ولأنني كما أشرت قبل قليل لا أنوي بأي حال من الأحوال أن ألعب دور المرجع لصيغ الدوال وتفاصيلها في هذه الحلقات، فإنني أمر سريعاً على الدوال المستخدمة لبيان سبب ونتيجة استخدامها، ومن المفروض أن الشرح هنا يكفي ليعطيك فكرة عن معنى كل هذا الكلام عن الدوال.

الدالة UCase() تستخدم لتحويل حالة أحرف معاملها النصي (الإنجليزي) إلى أحرف كبيرة إن كانت صغيرة من قبل. ومثلها الدالة LCase، ولكن التحويل يتم هاهنا من الأحرف الكبيرة إلى الصغيرة. لو استخدمنا هذه الدالة مباشرة على حقل اسم الموظف، فإن الناتج في الصف الأول، وهو LCase(SMITH) يصبح smith. لكن المطلوب هو أن يكون الحرف الأول كبيراً، لذلك قمنا بتقطيع كل اسم إلى جزءين: جزء هو عبارة عن حرف واحد من اليسار، والآخر عبارة عن بقية الأحرف من اليمين. من أجل استقطاع حرف من اليسار نستطيع استخدام أكثر من دالة، منها الدالة الواضحة في هذا السياق، Left(). هذه الدالة تأخذ معاملين: نصاً، وعدد الحروف المطلوبة على يسار هذا النص. من أجل استقطاع الحرف في أقصى اليسار، وتحويله إلى حرف كبير، نستخدم دالة UCase()، على نتيجة الدالة Left()، كالتالي:

```
UCase(Left('SMITH', 1) = UCase('S') = 'S'
```

بالمثل، نستطيع استخدام الدالة Right() من أجل استقطاع بقية الأحرف من اليمين. لكننا هنا لا نريد فقط حرفاً واحداً، ولا اثنين، ولكن عدداً متغيراً من الأحرف لا نعلم مسبقاً مقداره بالضبط. الحل هو أن نعمد إلى استخدام معادلة عامة تنفع مع كل النصوص. هذه المعادلة بسيطة جداً، لأن بقية عدد الحروف المطلوبة هو عدد حروف النص كله ماعدا واحداً. هذا يعني أننا نحتاج إلى معرفة العدد الكلي لحروف النص، ثم نطرح واحداً من النتيجة. لدينا دالة تعيد عدد حروف معاملها النصي، ولدينا معامل الطرح، لذلك فإن عدد الحروف عدا الحرف الأول يمكن استخلاصه كالتالي: Len('SMITH') – 1. بتحويل هذه النتيجة إلى أحرف صغيرة، تحصل على التالي:

```
LCase('MITH') = 'mith' = LCase(Right('SMITH', Len('SMITH' – 1))) = LCase(Right('SMITH', 4))
```

لم يتبق لنا الآن إلا لصق النتيجتين السابقتين معاً، ويمكن ذلك باستخدام المعامل & الذي يستخدم للصق النصوص بعضها إلى جانب بعض، كالتالي:

```
UCase(Left('SMITH',1)) & LCase(Right('SMITH', Len('SMITH')-1)) = 'S' & 'mith' = 'Smith'
```

لاحظ أن العمليات السابقة يتم تنفيذها مع كل صف من صفوف الاستعلام المستعادة، وهي 14 صفاً في مثالنا، لكن هذه الدوال لحسن الحظ مكتوبة بشكل جيد، فلا تأخذ الكثير من الوقت. كما ينبغي أن تلاحظ أن واحداً على الأقل من معاملات الدوال النصية هو نص بالطبع، وهذا يعني وضعه (في حالة كتابته يدوياً، وليس حفظه في متغير) بين فاصلتين علويتين أو علامتي تنصيص.

المطلوب الثاني يختلف قليلاً عن المطلوب الأول، حيث نحتاج إلى الحصول على ثلاثة أجزاء من النص (رقم الهاتف) ثم لصقها معاً بعد فصلها بالحرف "-". من أجل الحصول على الحروف الثلاثة الأولى من أقصى اليسار، نستخدم الدالة Left() كما سبق مع تحديد 3 كمعامل ثانٍ. من أجل استقطاع ثلاثة أحرف أخرى من الوسط نحتاج إلى دالة جديدة لديها القدرة على استقطاع جزء من نص. هذه القدرة ينبغي أن تشمل إمكانية تحديد بداية للاستقطاع، ومقدار للاستقطاع (عدد الحروف المطلوبة). لدينا في هذا الصدد الدالة Mid()، التي تأخذ بالفعل ثلاثة معاملات: النص الكامل، رقم الحرف حيث يبدأ الاستقطاع، وعدد الحروف المطلوبة للاستقطاع. هذا يعني أن شيئاً مثل Mid('0123034880', 4, 3) يعيد '303'. المعامل الثالث لهذه الدالة اختياري وليس إجبارياً، وهذا يعني أنك إذا لم تزود الدالة بعدد الحروف المطلوبة للاستقطاع، فإن كل أحرف النص الكامل ابتداء من الموقع المحدد بالمعامل الثاني يتم استقطاعها. من أجل توضيح هذا، تم استخدام هذه الدالة من أجل استقطاع الجزء الأخير (الأيمن) من رقم الهاتف عوضاً عن الدالة Right(). بالتطبيق على رقم الهاتف الأول، نحصل على:

```
Left('0123034880',3) & '-' & Mid('0123034880',4,3) & '-' & Mid('0123034880',7) =
'012' & '-' & '303' & '-' & '4880' = '012-303-4880'
```

لاحظ أن هذه الطريقة لا تأخذ في الاعتبار حالة حقول رقم الهاتف الخالية؛ لذلك تجد الشرطات الصغيرة في الناتج ولو لم تكن هناك أرقام هاتف. واحد من الحلول الممكنة استخدام الدالة Format. هذه الدالة مرنة للغاية، وتستخدم لتنسيق الخرج، ولها عدة تنويعات مثل FormatNumber، وFormatCurrency، وFormatDateTime وغيرها، وكل واحد منها يستقبل العديد من المعاملات أغلبها اختياري. أرجو أن ترجع إلى ملفات المساعدة أو أي مرجع من أجل الصيغ الدقيقة للاستخدام. ما يهمنا هنا هو استخدام الشكل العام الأول للدالة، وبالرجوع إلى نوع التعبير المعطى كمعامل أول للدالة، يمكن تنسيق الخرج بواسطة تنسيق محدد في المعامل الثاني. هناك عدد من الطرق القياسية في تنسيق الأرقام والتواريخ، لكن فيما يتعلق بالنصوص، تستطيع استخدام تنسيق خاص بك كالتالي:

```
Format('0123034880', '0##-###-####') = '012-303-4880'
```

لاحظ استخدام الهاش# مكان أي رقم، أما الصفر في البداية فهو من أجل أن يحافظ الأكسس على أول صفر في الخرج ولا يحذفه. لاحظ أيضاً استخدام عدد محدد من الأقام معلوم مسبقاً، ووضع الشرطة – في المكان المطلوب. أرجو كذلك أن تجرب مع هذه الدالة، مثلاً باستبدال هاش مكان الصفر في البداية، وتقليص أو زيادة عدد الهاشات، ومعاينة النتيجة.

هناك بعد الكثير من الدوال المتوفرة للاستخدام مع النصوص، حتى إنك تستطيع معرفة الآسكي كود ASCII Code لحرف، والعكس، وهو الحصول على الحرف من رقم الآسكي الخاص به. عندما تواجه مشكلة في الحياة العملية، فلا تنس أن تقوم باستكشاف سريع لقائمة الدوال المتاحة، حتى لا تعيد اختراع العجلة من جديد.

الدوال العددية

أشهر الدوال العددية هي الدوال الرياضية. منها الدوال المثلثية trigonometric، ودالة القيمة المطلقة absolute value، والجذر التربيعي square root، ودالة التقريب rounding، وغيرها. من أجل مثال على بعض هذه الدوال، دعنا نفترض مثالاً خيالياً يعيد الجذر التربيعي للأقام من 1 إلى 10، مع تقريب الناتج إلى أقرب رقمين عشريين:

```
SELECT TOP 10 (SELECT COUNT(*) FROM tblEmployee AS e1 WHERE e1.EmpNo <= e2.EmpNo) AS
No, Sqr((SELECT COUNT(*) FROM tblEmployee AS e1 WHERE e1.EmpNo <= e2.EmpNo))
AS [Square Root of No],
Round(Sqr((SELECT COUNT(*) FROM tblEmployee AS e1 WHERE e1.EmpNo <= e2.EmpNo)), 2)
AS [Square Root Rounded To 2 Decimal Places]
FROM tblEmployee AS e2
```

ناتج هذه العبارة هو:

	No	Square Root of	Square Root Rc
▶	١	١	١
	٢	١,٤١٤٢١٣٥٦٢٤	١,٤١
	٣	١,٧٣٢٠٥٠٨٠٧٦	١,٧٣
	٤	٢	٢
	٥	٢,٢٣٦٠٦٧٩٧٧٥	٢,٢٤
	٦	٢,٤٤٩٤٨٩٧٤٢٨	٢,٤٥
	٧	٢,٦٤٥٧٥١٣١١١	٢,٦٥
	٨	٢,٨٢٨٤٢٧١٢٤٧	٢,٨٣
	٩	٣	٣
	١٠	٣,١٦٢٢٧٧٦٦٠٢	٣,١٦

الاستعلام الفرعي يعيد رقماً تسلسلياً للموظف ضمن الاستعلام الأساسي (راجع من فضلك الحلقة السابقة). اخترنا عشرة موظفين فقط بواسطة TOP 10، وفي الحقل الثاني استخدمنا الدالة Sqr من أجل استعادة الجذر التربيعي للرقم المتسلسل، ثم قربناه في الحقل الثالث باستخدام الدالةRound التي تأخذ عدد الأماكن العشرية المطلوب التقريب إليها، في معاملها الثاني. لاحظ أن هذا المثال مصطنع، ولكنه يخدم الغرض في توضيح استخدام الدوال العددية.

دوال التاريخ والوقت

لك أن تتوقع أن هناك العديد من الدوال التي تعالج التواريخ والأرقام كذلك. هناك حساب خاص بالتواريخ والأوقات يختلف عن حساب الأعداد كذلك. ربما من أشهر الدوال في هذه المجموعة الدوال Date()، وTime()، وNow()، التي تستخدم للحصول على التاريخ الحالي، والوقت الحالي، وكليهما على التوالي (حسب تقويم وساعة الجهاز). هنالك دوالّ تتيح لك جمع فترة زمنية محددة إلى تاريخ معين، هذه الفترة قد تكون سنة أو شهراً أو يوماً أو دقيقة وهكذا، ودالة أخرى تتيح لك طرح فترة من تاريخ معين، ودوال لاستخراج أجزاء السنة والشهر والأسبوع واليوم والساعة والدقيقة وحتى الثانية من تاريخ محدد. كما أن هناك معامل الطرح العادي (-) الذي يعيد الفرق بين تاريخين بعدد الأيام. كمثال على استخدام بعض هذه الدوال من SQL، دعنا نفترض أننا نريد الاستعلام عن تاريخ تعيين الموظفين مفصلاً إلى سنة وشهر (بالحروف) ويوم، ثم نريد كذلك أن نحدد تاريخ تقاعد كل موظف بإضافة ثلاثين سنة (مثلاً) إلى تاريخ تعيينه:

```
SELECT EmpNo AS [رقم الموظف], EmpName AS [اسم الموظف], EmpHireDate AS [تاريخ التعيين],
Year(EmpHireDate) AS [سنة التعيين],
Format(EmpHireDate, 'mmm') AS [شهر التعيين],
Day(EmpHireDate) AS [يوم التعيين],
DateAdd("yyyy", 30, EmpHireDate) AS [تاريخ التقاعد]
FROM tblEmployee
```

الناتج هو:

	رقم الموظف	اسم الموظف	تاريخ التعيين	سنة التعيين	شهر التعيين	يوم التعيين	تاريخ التقاعد
▶	٧٣٦٩	SMITH	١٧/١٢/١٩٨٠	١٩٨٠	ديسمبر	١٧	١٧/١٢/٢٠١٠
	٧٤٩٩	ALLEN	٢٠/٠٢/١٩٨١	١٩٨١	فبراير	٢٠	٢٠/٠٢/٢٠١١
	٧٥٢١	WARD	٢٢/٠٢/١٩٨١	١٩٨١	فبراير	٢٢	٢٢/٠٢/٢٠١١
	٧٥٦٦	JONES	٠٢/٠٤/١٩٨١	١٩٨١	ابريل	٢	٠٢/٠٤/٢٠١١
	٧٦٥٤	MARTIN	٢٨/٠٩/١٩٨١	١٩٨١	سبتمبر	٢٨	٢٨/٠٩/٢٠١١
	٧٦٩٨	BLAKE	٠١/٠٥/١٩٨١	١٩٨١	مايو	١	٠١/٠٥/٢٠١١
	٧٧٨٢	CLARK	٠٩/٠٦/١٩٨١	١٩٨١	يونيو	٩	٠٩/٠٦/٢٠١١
	٧٧٨٨	SCOTT	١٩/٠٤/١٩٨٧	١٩٨٧	ابريل	١٩	١٩/٠٤/٢٠١٧
	٧٨٣٩	KING	١٧/١١/١٩٨١	١٩٨١	نوفمبر	١٧	١٧/١١/٢٠١١
	٧٨٤٤	TURNER	٠٨/٠٩/١٩٨١	١٩٨١	سبتمبر	٨	٠٨/٠٩/٢٠١١
	٧٨٧٦	ADAMS	٢٣/٠٥/١٩٨٧	١٩٨٧	مايو	٢٣	٢٣/٠٥/٢٠١٧
	٧٩٠٠	JAMES	٠٣/١٢/١٩٨١	١٩٨١	ديسمبر	٣	٠٣/١٢/٢٠١١
	٧٩٠٢	FORD	٠٣/١٢/١٩٨١	١٩٨١	ديسمبر	٣	٠٣/١٢/٢٠١١
	٧٩٣٤	MILLER	٢٣/٠١/١٩٨٢	١٩٨٢	يناير	٢٣	٢٣/٠١/٢٠١٢
*							

في الدالة ()DateAdd، يحدد المعامل الأول نوع الفترة (سنة yyyy، أو شهر m أو يوم d، وهكذا...)، ويحدد المعامل الثاني عدد الفترات المطلوب إضافتها (يمكن أن تكون بالسالب من أجل الحصول على تاريخ في الماضي)، ثم يحدد المعامل الثالث تاريخ بدء الاحتساب. أرجو الرجوع إلى المراجع من أجل تفاصيل بقية الدوال.

المزيد؟

نعم، هناك العديد من الدوال المتنوعة التي قد لا تندرج بالضبط تحت واحد من الأنواع السابقة، لكنها لا تقل أهمية أو كثرة في الاستخدام؛ مثال على ذلك الدالة ()IIf، ودوال الاختبار مثل ()IsNull، ودوال التحويل مثل ()CStr. مرة أخرى، ليس المقام هنا مقام مرجعية لهذه الدوال، بقدر ما هو مقام إشارة إليها، مجرد إشارة، لتعلم أن هناك عدة مفيدة جداً في جعبتك عندما تواجه بعط المتطلبات. كتعويض عن التفصيل في هذه الحلقة، أزودك هنا بتمرين مفيد، ستستفيد منه إن شاء الله فائدة عظيمة، لو أنك كنت تريد...

تمرين:

اذهب إلى محرر VBA، ثم اكتب الدوال التالية دالة دالة (في أي مكان)، في كل مرة اكتب فقط اسم الدالة، ثم ظلله، ثم اضغط المفتاح F1. إذا كنت لا تعرف كيف تذهب إلى محرر VBA، فإنها لمشكلة! لكن حتى نحل هذه المشكلة، اذهب إلى تبويب الوحدات النمطية ثم أنشئ واحدة جديدة.

Asc, Chr, Format InStr, LCase, Left, LTrim, Mid, Replace, Right, RTrim, String, StrReverse, Trim, UCase
Abs, Cos, Exp, Log, Rnd, Round, Sgn, Sin, Sqr
Date, DateAdd, DateDiff, DatePart, DateSerial, Day, Hour, Minute, Month, Now, Second, Time, Year

إضافة، تعديل، وحذف السجلات

إضافة السجلات

من الواضح أن الاستعلام عن البيانات لا يتم إلا في وجود بيانات. في تطبيقات قواعد البيانات، تضاف البيانات باستمرار، وقد تعدل وتحذف، بخلاف عمليات الاستعلام من أجل استخلاص معلومات مفيدة. إضافة البيانات إلى قاعدة بيانات تتكون من جداول، تتم على أساس الصفوف. في كل مرة تضيف فيها بيانات جديدة إلى جدول، فإنك تضيف صفاً أو أكثر إلى هذا الجدول. قد تكون مجبراً على ملء كل أو بعض الحقول، وقد لا تكون مجبراً (في أسوأ حالات التصميم) إلى ملء أي حقل، لكن صفاً قد أضيف بقيم خالية. لا نتحدث هنا عن إضافة حقول إلى الجداول، لأن هذا يعني تعديل بنية الجدول، وهذه العملية تدخل ضمن ما يسمى (تعريف البيانات Data Definition)، وليس معالجة البيانات (Data Manipulation) التي نتحدث عنها.

توفر Jet SQL عبارة خاصة هي عبارة الأمر INSERT INTO من أجل إضافة السجلات إلى جدول. لهذه العبارة القدرة على إضافة سجل واحد أو أكثر من سجل مرة واحدة، ولهذا فإن لها شكلين:

1. واحد تحدد فيه القيم الجديدة يدوياً، ويستخدم لإضافة سجل واحد فقط في المرة الواحدة،
2. وشكل آخر يستخدم العبارة SELECT لإحضار صف أو أكثر من جدول (أو استعلام) آخر، يسمى المصدر، إلى الجدول (أو الاستعلام) الوجهة (نعم، يمكن أن تضيف إلى استعلام؛ إن كان الاستعلام مبنياً على أكثر من جدول، ولم تنس حقول المفاتيح الأساسية والأجنبية، فإن الصف ستتم إضافته إلى الحقول المناسبة في جداول الاستعلام).

دعنا الآن نر هذا الأمر وهو يعمل. من أجل إضافة صف واحد إلى جدول الموظفين، يمكن استخدام العبارة التالية:

```
INSERT INTO  tblEmployee (EmpNO, EmpName, EmpManager, EmpHireDate, EmpSal, EmpPhone,
EmpDept) VALUES (9999, "AHMAD", NULL, #28/10/2009#, 3000, "0123034880", 50)
```

لاحظ التالي:

- بعد تحديد اسم الجدول، تم سرد كل حقول الجدول بين قوسين. هذا ليس ضرورياً إذا لم تكن عازماً على إدخال قيم في كل الحقول؛ يمكن فقط أن تحدد الحقول التي ستستقبل بيانات. طبعاً كل الحقول المفتاحية والمطلوبة يجب أن تكون من ضمن الحقول المحددة. في الحقيقة، هذا الجزء بين القوسين ليس ضرورياً إن كنت ستحدد قيماً لكل الحقول، بالترتيب الصحيح في الجزء التالي.
- الجزء التالي ضروري، وهو الكلمة VALUES، وبعد ذلك بين قوسين، قيم كل الحقول المذكورة في الجزء السابق، أو قيم كل الحقول بالترتيب الصحيح في حالة عدم تحديد أسماء الحقول المطلوبة.
- في حالة عدم وجود قيمة حاضرة لحقل ما، لاحظ كيف تم تحديد القيمة الخالية NULL للحقل.
- القيم النصية تكتب بين علامتي تنصيص كالعادة، كذلك قيم التاريخ تحدد بين علامتي هاش #.

يصبح جدولنا بعد العبارة السابقة كالتالي:

EmpNO	EmpName	EmpManager	EmpHireDate	EmpSal	EmpPhone	EmpDept
٧٣٦٩	SMITH	٧٩٠٢	١٧/١٢/١٩٨٠	٨٠٠		٢٠
٧٤٩٩	ALLEN	٧٦٩٨	٢٠/٠٢/١٩٨١	١٦٠٠	٠١٢٣٠٣٤٨٨٠	٣٠
٧٥٢١	WARD	٧٦٩٨	٢٢/٠٢/١٩٨١	١٢٥٠	٠١٧٨٨٩٦٢٠١	٣٠
٧٥٦٦	JONES	٧٨٣٩	٠٢/٠٤/١٩٨١	٢٩٧٥		٢٠
٧٦٥٤	MARTIN	٧٦٩٨	٢٨/٠٩/١٩٨١	١٢٥٠	٠١٢٢٥٥٠٠٥٧	٣٠
٧٦٩٨	BLAKE	٧٨٣٩	٠١/٠٥/١٩٨١	٢٨٥٠	٥١٢٣٠٥٦٨٨٧	٣٠
٧٧٨٢	CLARK	٧٨٣٩	٠٩/٠٦/١٩٨١	٢٤٥٠		١٠
٧٧٨٨	SCOTT	٧٥٦٦	١٩/٠٤/١٩٨٧	٣٠٠٠		٢٠
٧٨٣٩	KING		١٧/١١/١٩٨١	٥٠٠٠		١٠
٧٨٤٤	TURNER	٧٦٩٨	٠٨/٠٩/١٩٨١	١٥٠٠	٠	٣٠
٧٨٧٦	ADAMS	٧٧٨٨	٢٣/٠٥/١٩٨٧	١١٠٠		٢٠
٧٩٠٠	JAMES	٧٦٩٨	٠٣/١٢/١٩٨١	٩٥٠		٣٠
٧٩٠٢	FORD	٧٥٦٦	٠٣/١٢/١٩٨١	٣٠٠٠		٢٠
٧٩٣٤	MILLER	٧٧٨٢	٢٣/٠١/١٩٨٢	١٣٠٠		١٠
٩٩٩٩	AHMAD		٢٨/١٠/٢٠٠٩	٣٠٠٠	٠١٢٣٠٣٤٨٨٠	٥٠

يمكن كذلك إضافة مجموعة من الصفوف إلى الجدول باستخدام الصيغة الثانية للعبارة INSERT INTO، وذلك باستخدام استعلام لجلب الصفوف المدخلة من جدول أو استعلام آخر. بافتراض أن هنالك جدول آخر (tblTemp) يحوي قيماً لأرقام وأسماء وأقسام موظفين، يمكن كتابة العبارة التالية لإضافة هؤلاء الموظفين إلى جدولنا:

```
INSERT INTO tblEmployee ( EmpNo, EmpName, EmpDept )
SELECT EmpNo, EmpName, EmpDept
FROM tblTemp
```

بلغة الأكسس، يسمى هذا القسم من أقسام معالجة البيانات باستعلام الإلحاق Append Query.

تعديل السجلات

يستخدم الأمر UPDATE في عبارة خاصة من أجل تعديل بيانات موجودة في قاعدة البيانات. يسمى هذا النوع باستعلام التعديل Update Query. مثال بسيط يوضح كيفية استخدام هذه العبارة هو تعديل راتب آخر موظف قمنا بإضافته في المثال السابق. لنقم من باب التفاؤل بزيادة راتبه إلى 5000 مادام يعمل في قسم تقنية المعلومات:

```
UPDATE tblEmployee SET EmpSal = EmpSal + 2000 WHERE EmpNo = 9999
```

من المهم جداً بالطبع التنبه إلى تقييد التعديل بشرط، وإلا تم التعديل على كل صفوف الجدول (لن يكون المدير ممنوناً لك). كما ترى، بعد الأمر UPDATE تحدد اسم الجدول أو الاستعلام (أو الجداول كما سنرى بإذن الله قريباً) التي تريد تعديل بيانات فيها. بعد ذلك الأمر SET، ثم الحقول التي تريد تعديل قيمها مع القيم الجديدة. في حالة تعديل أكثر من حقل، افصل كل حقل مع قيمته الجديدة عن الآخر بفاصلة. لاحظ كيف قمنا بتعديل الراتب بإضافة 2000 إلى الراتب الحالي، كما قمنا بتحديد موظف بعينه بواسطة رقمه.

كمثال أوسع قليلاً، دعنا نجرب تعديل اسم قسم مع زيادة رواتب موظفيه في نفس الوقت في عبارة واحدة. أسماء الأقسام موجودة في جدول الأقسام، في حين أن رواتب الموظفين موجودة في جدول الموظفين. نحتاج كالعادة إلى الربط بين الجدولين، بواسطة حقل رقم القسم (مفتاح أساسي في جدول الأقسام، ومفتاح أجنبي في جدول الموظفين). لا نريد بطبيعة الحال تعديل كل صفوف الدولين، لذا سنختار فقط القسم رقم 50 (IT)، فنغير اسمه إلى (Information Technology)، ونزيد رواتب موظفيه بمقدار 500. لاحظ ان الصف المتأثر هو فقط صف واحد لأننا لا نملك إلا موظفاً واحداً في هذا القسم:

```
UPDATE tblDepartment INNER JOIN tblEmployee ON tblDepartment.dptNo=tblEmployee.EmpDept
SET dptName = "Information Technology", EmpSal = EmpSal+500
WHERE dptNo=50
```

ملحوظة مهمة: انتبه إلى عدد مرات تنفيذ استعلام التعديل من النوع السابق، لأن الراتب سوف يزيد في كل مرة تنفذ فيها الاستعلام بالمقدار المحدد، لذا ينبغي تقييد تنفيذ الاستعلامات من هذا النوع ضمن عملية مدروسة. مثال على ذلك حركات احتساب وترحيل الأرصدة والفواتير ينبغي تأطيرها في عملية غير قابلة للتكرار إلا بعد التراجع عن الترحيل السابق.

حذف السجلات

حذف السجلات هي أسهل عمليات المعالجة. بسبب هذه السهولة وبسبب ما قد ينجم عنها من أثر وخيم (فقدان البيانات إلى الأبد) فهي من أخطر العمليات. ينبغي الاحتراز عند استخدام هذه العملية، كما ينبغي الاحتياط بأخذ نسخ احتياطية من البيانات بشكل دوري. صيغة عبارة الحذف مباشرة للغاية، وتحوي الأمر DELETE (كلمة معبرة)، مع اسم الجدول المراد حذف أحد صفوفه (أو أكثر)، ثم بالتأكيد أي شرط لتحديد الصف أو الصفوف المراد حذفها. لا تهمل شرط المقطع WHERE لأن ذلك يعني حذف كل صفوف الجدول. لأن الحذف يتم على مستوى صفوف كاملة، فإنك لا تحتج إلى تحديد أي حقول من أجل الحذف، كل الصف سيتم حذفه، وهذا يعني أنه من أجل حذف الموظف الذي قمنا بإضافته، وزيادة راتبه للتو (هذا المسكين)، يمكن ان نكتب التالي:

```
DELETE * FROM tblEmployee WHERE EmpNo = 9999
```

إذا أردت التخلص من قيمة محددة في حقل محدد (وليس من كامل الصف)، فإن بإمكانك بالطبع استخدام الأمر UPDATE مع تحديد قيمة الحقل بـNULL. كما قد تجدر الإشارة إلى أن الأمر DELETE يحذف البيانات من الجدول، ولا يحذف الجدول.

خاتمة

بهذه اللمحة السريعة على قسم معالجة البيانات من اللغة SQL أختم هذه الدورة. لست أنوي الخوض في قسم تعريف البيانات حالياً (ربما في جزء لاحق)، كما أنني قد أغفلت بعض أنواع الاستعلامات، والاستخدامات المتقدمة للغة SQL، لكن القصد من هذه المقدمة هو تقديم الأساس، وأرجو أن تكون قد لمست هذا وع وصولك إلى هذا السطر. لقد طالت هذه الدورة (السريعة) أكثر مما يجب، لكنني أرجو أن يكون ذلك قد أضاف شيئاً ما إلى المصادر العربية.

أسأل الله أن يجعل هذا العمل نافعاً لكاتبه وقارئه، وأن يصلح نيتي ويزكيني ويهديني سواء السبيل، ويرزقني العلم والعمل جميعاً، وأن يعيذني من عذاب القبر ومن عذاب النار، وأن يؤمنني من هول يوم القيامة، ووالدي وأحبابي أجمعين، والحمد لله رب العالمين.