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.

مذكرات حول تصميم قواعد البيانات وتطبيقاتها
مقدمة إلى تحليل وتصميم قواعد البيانات

(الجزء الأول)

أصل هذا الكتاب 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

1430 هـ - 2009 م

المحتويات

تمهيد

هل أنت مبرمج قواعد بيانات؟...

سؤال أختاره لبدء هذه السلسلة من النقاشات التي تهدف إلى بث الروح في المنتدى بقليل من التأصيل حين نلاحظ أن المنهجية تكاد تختفي.

نعم؛ لست بالأهل لتقديم هذه المفاهيم في وجود من هو أخبر، لكنه إسهام مني لعل الله يكتب له القبول ويفتح به الباب الذي علا مفاصله الصدأ أو كاد. الكثير من الكلام قد يقال، وقد تتداخل النقاط، لكنني أرى العنوان هو الأصلح والأقرب إلى المخطط العام للنقاش. نعود إلى السؤال:

هل أنت مبرمج قواعد بيانات؟

لكن، هل السؤال صحيح؟!

هذه نقطة مفصلية، معبرة عن غياب التأصيل الذي أشرت إليه آنفاً غير بعيد...

جرت العادة على إطلاق هذا اللقب على من يعد تطبيقات تعتمد على قواعد البيانات. هل تُبرمج قواعد البيانات؟ قواعد البيانات تصمم (ومن ثم تبنى)، لكن البرامج هي التي تبرمج. من ناحية التدقيق الصارم، يمكن أن نقول إنك (ربما) تكون مبرمجاً (لـ) ـقواعد البيانات...

(ربما)؟ لماذا الغلط؟ أنا بالتأكيد مبرمج لقواعد البيانات لأنني بالفعل أقوم بذلك باستخدام الأكسس ليل نهار...

ما شاء الله، السؤال هنا هو: ما الذي تقوم به بالضبط؟ وكائناً ما كان ذلك، هل تقوم به بالشكل الصحيح؟

هل هذا يستفزك؟ إذاً فقد وصلنا لنقطة البداية، وأنت على الرحب والسعة في هذا النقاش... لا تغادر الآن، لأن لدينا هاهنا المزيد لنقوله...

بعض الأسئلة الاستفزازية الأخرى:

تصفح أسئلة المنتدى وأخبرني، لماذا هناك الكثير من المبرمجين والبرامج، ولكنها لا تعمل بالشكل المرضي؟

لماذا قلّت الأسئلة عن المفاهيم والمبادئ وكثرت طلبات التعديل في الملفات المرفقة؟

لماذا كثرت الأسئلة عن خصائص زر ونموذج وقلّت الأسئلة عن تصميم جدول (اللبنة الأساسية لقاعدة البيانات؟)

لماذا لم يسأل أحد هذه الأسئلة من قبل؟!...

إذا كان الكلام يبدو سمجاً أو غير مقبول، فأنا أضع الأمر بين يديك أيها القارئ... هل من المحبذ البدء بسلسلة من هذا القبيل، حسب ما يأذن به الله من وقت وقدرة، أم أنه موضوع (ثقيل الدم)، غير مقبول؟ للمساعدة على الرد، أطرح هذه الأسئلة للإجابة:

ما هو الأكسس بالضبط؟ هل هو لغة؟ برنامج؟ قاعدة بيانات؟

ما الفرق بين المصطلحات السابقة؟

ما دخل الـ SQL في الموضوع؟ ما هي أصلاً؟

هل أنت مبرمج قواعد بيانات؟!...

القسم الأول: مبادئ التحليل وتصميم قاعدة البيانات

رقم الحلقة (1) تعريفات أساسية: قواعد البيانات والأكسس

المشكلة هي غياب البداية الصحيحة، وغياب البداية الصحيحة يعني غياب النهاية الصحيحة. كيف تريد أن تبرمج من أجل قواعد البيانات وأنت لم تتعلم البرمجة ولا قواعد البيانات؟ هاهنا أيضاً مشكلة عدم الاعتراف بقواعد البيانات كعلم وفن مستقل له كتبه ومناهجه وتقنياته ومنتجاته وشهاداته. بالنسبة للكثير، قواعد البيانات هي الأكسس ونماذجه وتقاريره، والأكسس ونماذجه وتقاريره هي قواعد البيانات.

ولكن ما هي قواعد البيانات؟

قاعدة الجنود هي المكان الذي يحوي الجنود، وقاعدة البيانات هي المكان الذي يحوي البيانات. في حالة الأكسس (وما شابهه) قاعدة البيانات هي بشكل أساسي الجداول التي تحوي البيانات.

هذا يبدو تافهاً، لكنه يتضمن نقطة مهمة: الأكسس بحد ذاته ليس قاعدة بيانات، ولا النماذج ولا التقارير... الجداول (والاستعلامات) التي تصممها أنت من أجل مخزن ما أو صيدلية هي قاعدة بيانات، Northwind هي قاعدة بيانات أخرى؛ وهكذا لدينا عدد غير محدود من قواعد البيانات، لكن لدينا أكسس واحداً فقط.

إذاً، ما هو الأكسس بالضبط؟

في الزمن القديم الجميل، كانت البيانات تحفظ في ملفات تتم إدارتها من قبل نظام التشغيل كبقية أنواع الملفات. أنت تطلب من نظام التشغيل إنشاء الملف، وفتحه وإغلاقه، وحفظه في القرص الصلب، لكنك المسؤول تماماً عن محتوياته وتنسيقه... في حالة قواعد البيانات هذا يعني صداعاً لا يطاق للمبرمج الذي ينبغي أن يعلم تفاصيل التنسيق ويتعامل معها بدقة شديدة عند العمليات المختلفة من إضافة وتعديل وحذف واسترجاع البيانات، لكن هذا ليس هو السبب الأول لترك نظام الملفات كتطبيق محترم لقواعد البيانات... هناك مشاكل التكرار والمحافظة على التكامل مما ليس الآن أوان التفصيل فيه...

الحل؟... كالعادة، أنشئ طبقة أخرى... حكاية الطبقات هذه تشكل هوساً غير طبيعي في عالم الحاسوب وعلومه؛ كل شيء عبارة عن طبقات بعضها فوق بعض، إذا أخرج يده لم يكد يراها... اختفت التفاصيل تحت أكوام الطبقات، ومن يومها لم نعد نفهم شيئاً، المهم أن نستخدمه (بسهولة)... يا للأسى؛ من الممكن أن تتقبل هذا لمستخدم الحاسوب العادي، لكن أن يصبح المتخصص في علوب الحاسوب هو أول من يصاب بهذه اللعنة.....

أمثلة؟... من الصعب عدم العثور على مثال! أنت برمجت بالسي؟ هي طبقة تخفي عنك (وتريحك) من تفاصيل لغة التجميع (assembly)، المترجم (compiler) هو البرنامج الذي يجعل هذا ممكناً، ولغة التجميع نفسها هي طبقة تعزلك عن لغة الآلة، المجمع (assembler) له البطولة هاهنا. تعرف نظام التشغيل؟ لا أقصد الإهانة بالطبع، الويندوز هو طبقة تفصلك عن تعقيدات العتاد الصلب وتعريفاته. نعم، نعم، أعلم أن البعض قد درس الشبكات وهو يتذكر طبعاً المرجع الشهير OSI، وطبقات TCP/IP الأربع (أو الخمس حسب بعض المراجع). أصبح الإنسان مرفهاً هذه الأيام، ومستعداً للتضحية بكثير من الفهم من أجل القليل من الراحة... لكن هذا موضوع آخر.

لماذا إذاً لا نستعين بطبقة بيننا وبين نظام التشغيل وملفاته تزيح عنا بعض العبء، وفي نفس الوقت نزودها بالكثير من الميزات التي تخلصنا من المشاكل القديمة؟ هذه الطبقة الجديدة هي، بالضبط كحالة المترجمات وأنظمة التشغيل، مجرد برامج؛ لكنها برامج خاصة ومعقدة جداً. كلمة جداً هنا تعني في الواقع جداً جداً، لأن هذه البرامج من المفترض أن تعد مرة واحدة لتقوم بمهمة شاقة، ثم يستخدمها الجميع ممن ليس لديه الوقت ولا الإمكانية لأداء هذه المهام الشاقة؛ ليس كل مبرمج مستعداً لأن يتعلم تفاصيل التعامل مع القرص الصلب حتى من خلال طبقة التعريف المزودة من مصنع القرص الصلب، هو يريد فقط أن يقول (افتح يا...) باستخدام دالة من دوال نظام التشغيل على سبيل المثال، ويقوم نظام التشغيل بالمهمة. الفكرة مشابهة في حالة قواعد البيانات. ولكي تدرك صعوبة هذه البرامج الخاصة التي أعدت خصيصاً من أجل قواعد البيانات، عليك أن تتذكر أن القليل من الشركات فقط أنتجت القليل من هذه البرامج، بعضها أنت تعرف أسماءها جيداً. نعم نعم، Oracle، Microdsoft، Sybase، IBM والقائمة لا أظنها تطول كثيراً. كإجابة على السؤال الأول (لمن لم يزل ينتظر الإجابة): أكسس هو واحد من هذه البرامج من Microsoft طبعاً.

ما الذي تقوم به هذه البرامج؟... إنها باختصار نظام تشغيل قواعد البيانات كما أن الويندوز مثلاً هو نظام تشغيل الملفات. وظائفها كثيرة، هي في المجمل إدارة قاعدة البيانات من عدة نواحي، لذلك تسمى هذه البرامج (برامج إدارة قواعد البيانات)، وهذا هو التعريف الدقيق للأكسس (والأوراكل والـ MS SQL Server بطبيعة الحال)، وهذه هي الإجابة الدقيقة للسؤال: ما هو الأكسس بالضبط؟

رقم الحلقة (2) المزيد من التعريفات الأساسية: DBMS وتطبيقات قواعد البيانات

إذاً، فالأكسس هو DBMS آخر... ذلك اختصار DataBase Management System، مصطلح ربما مر عليك مراراً، ويعني كما أسلفت من قبل، نظام (أو برنامج) إدارة قواعد بيانات. لم تكن أشكال قواعد البيانات (كيفية ترتيب البيانات) وبالتالي برامج إدارتها كما نعرفه الآن في الأكسس مثلاً، لكن هذا النوع الذي يرتب البيانات على شكل جداول ذات صفوف وأعمدة ويبني العلاقات بينها بواسطة عمود مشترك أو أكثر، يدعى بقواعد البيانات العلائقية (Relational Databases)، وبرامج إدارته تدعى RDBMS، الـR في البداية من أجل كلمة (Relational). كل ما تعرفه من قواعد بيانات الآن ومن برامج إدارتها من أمثلة الأوراكل والأكسس ينتمي لهذا النوع.

الأكسس ليس لغة، وليس قاعدة بيانات، ولكنه برنامج إدارة قواعد بيانات... هذه نتيجة جيدة. المزيد عن برامج إدارة قواعد البيانات ربما يصادفنا لاحقاً...

الفرق بين قواعد البيانات وأنظمة إدارتها يشبه الفرق بين الملفات وأنظمة إدارتها (نظم التشغيل)... هذه نتيجة أخرى جيدة... لن يضرك أن تتذكر ذلك.

حسن، قاعدة البيانات هي المكان الذي تحفظ فيه البيانات، وهذا يعني مجموع الجداول (وكائنات أخرى قليلة أذكرها لاحقاً إن شاء الله)، لكن هذا يستبعد تماماً النماذج والتقارير وما شاكلها... هذه من أهم النقاط التي تقودنا إلى مواضيع أخرى متشعبة، لكن دعني أولاً أقدم مصطلحاً آخر...

قواعد بيانات، برنامج إدارة قواعد بيانات... تطبيق قواعد بيانات.

هذا المصطلح الأخير مهم جداً بالنسبة لنا، وهو ما نعنيه عادة عندما نقول إنك مبرمج قواعد بيانات؛ أنت تبرمج تطبيقاً يتعامل مع قاعدة بيانات. أنت تعد برنامجاً (التطبيق ممكن أن نتفق على أنه برنامج كبير يتكون من عدة أجزاء كل منها برنامج أصغر)، يعطي ويأخذ من قاعدة بيانات، وهذا من أكثر أنواع التطبيقات طلباً في السوق. لماذا؟ لأن كل منشأة صغرت أم كبرت هذه الأيام تحتاج إلى قاعدة بيانات، أظن هذا واضحاً.

الفقرة السابقة تثير بعض الأسئلة:

هل هذا يعني أن النماذج والتقارير تشكل ما تسميه تطبيق قاعدة بيانات؟
نعم، المزيد من التوضيح يلحق إن شاء الله فيما بعد.

إذا كانت قواعد البيانات تختلف عن تطبيقات قواعد البيانات، وكل منشأة تحتاج إلى قاعدة بيانات، لماذا إذاً تشتري المنشآت هذه التطبيقات؟ لماذا لاتتعامل مباشرة مع قاعد البيانات؟

الطبقات يا صديقي، الطبقات... لا تنس الطبقات! ذكرت أن برنامج إدارة قواعد البيانات هو الذي يدير قاعدة البيانات، والتعامل مع قاعدة البيانات يتم عبر هذا البرنامج، لكن ليس كل أحد قادراً على التعامل معه، الناس تريد تعاملاً سهلاً، سريعاً، مريحاً، فعالاً، لا نريد أن نعلم الكثير عن التفاصيل، شكراً لك. على كل حال، لو لم يكن الأمر كذلك لما كنت أنت الآن هنا تقرأ هذا، لأن أحداً لن يحتاج إليك كمبرمج تطبيقات من أجل قواعد البيانات! الناس تريد طبقة سهلة كواجهة تسمح لها بالتعامل مع قاعدة البيانات أخذاً وعطاءً. هذا الكلام يشير بوضوح إلى أن هنالك طبقات في عالم قواعد البيانات وتطبيقاتها... نتيجة جداً مهمة نعود (أيضاً) إليها لاحقاً بإذن الله.

لحظة... تقول لي الآن: أنت ذكرت أنه ليس في مقدور الجميع التعامل مباشرة مع برنامج إدارة قواعد البيانات... هل أنا أيضاً لا أتعامل مع قاعدة البيانات إلا عبر برنامج إدارتها؟ ثم هل أنا قادر على التعامل مع هذا البرنامج؟ وكيف يتم التعامل معه أصلاً؟ لا تحاول أن تخدعنا، لا تغفل التفاصيل المهمة رجاءً...

رقم الحلقة (3) تعريفات أساسية أخرى: SQL

كنت تسأل أيها القارئ الكريم عن التعامل مع برنامج إدارة قواعد البيانات... السؤال جيد وفي محله، ويقود إلى المزيد من المصطلحات والمفاهيم المهمة.

أنت تريد أن تقول شيئاً، وأنا أعلم ما هو: هل علينا أن نعلم كل هذا؟ لماذا لا نتجه مباشرة إلى الناحية العملية، ونتعلم (كيف) نصنع الأشياء؟ الوقت لا يتسع لكل هذا التنظير، والناس يسبقوننا... أنت تضيع وقتنا!...

سأخبرك الآن بقاعدة عامة، إن لم تصدقها مني، فعلى الأقل قد تصدقها من البخاري الذي بوب في صحيحه تحت عنوان (باب العلم قبل العمل). هناك مثل يقول: البخيل يشتري مرتين. والبخيل بوقته في ساعة العلم سيدفع الثمن مضاعفاً مرات كثيرة في ساعة العمل. اسمع يا أخي نصيحة أخ مشفق: إذا لم ترد أن تقضي بقية حياتك تسأل في المنتديات عند كل خطوة في برامجك، فاعلم أولاً (ماذا) و(لماذا)، وستأتي (كيف) بإذن الله تبعاً.

لا أريد أن أطيل في هذا الكلام، لكنني سأعود بإذن الله إليه مراراً، لأنني ألمس من كثير من الإخوان أحد أمرين: إما أنه الاستصعاب للمنهجية السليمة، وهذا لا ينبغي لأن حل المشكلة لا يكون بالهروب منها، بل بمواجهتها. وكل صعب يسير إذا يسره الله، ولكن سنة الله اقتضت أن لكل مسبب سبباً، وإنما العلم بالتعلم. الأمر الثاني هو الاستخفاف، وهذه مشكلة حقيقية، لأن استخفاف المرء بالمجال الذي اختاره يؤدي إلى استخفاف الناس به أيضاً، وهذا ما نلاحظه بالفعل؛ أصبح الناس يعتقدون أن أي شخص من الممكن أن يبني تطبيقاً ويسيّر العمل، وأنه لا حاجة بالفعل لمن يدعون أنفسهم متخصصي حاسوب. نقطة.

برامج إدارة قواعد البيانات هي التي تقوم عنك بحفظ بياناتك على القرص الصلب، وهي التي تعرف بالضبط كيف تم حفظها وأين، أنت فقط تخبرها بالبيانات التي تريد حفظها (أو استرجاعها أو معالجتها) وبهيكلها المنطقي، وهي تتولى معالجتها فيزيائياً. هذا الكلام المجمل سيصبح واضحاً بعد قليل بإذن الله. هذه البرامج هي التي تفرض أي شروط تحددها أنت من أجل أمن وسلامة قاعدة البيانات وما فيها من بيانات، وتوفر لك الكثير من الخدمات التي كانت ستقع على عاتقك، مثلاً السماح بالوصول المتزامن لقاعدة البيانات من عدة مستخدمين.

هذا إلى الآن لا يجيب عن السؤال بعد: إذا كنت أنا من سيحدد البيانات وبنيتها المنطقية، وأنا من سيضع شروط السلامة والأمان، فلا بد من وجود وسيلة للتواصل مع برنامج إدارة قواعد البيانات لإيصال كل هذا. صحيح؟... صحيح. ماهي الوسيلة؟... SQL طبعاً، ومن غيرها؟

بالنسبة لبرامج إدارة قواعد البيانات العلائقية، هناك لغة واحدة مفهومة: لغة قواعد البيانات العلائقية. هذه اللغة تم تصميمها من البداية مع قواعد البيانات العلائقية ولها اسم غير دقيق تماماً ولكنه اسمها على أية حال. SQL: Structured Query Language، يترجمونها أحياناً إلى العربية بـ(لغة الاستعلامات البنيوية). طبعاً المقصود أنها لغة، وأن لها بنية ما (لا أظن أن هناك لغة بدون بنية)، وأنها تستخدم في الاستعلامات. هذه التسمية ربما خرجت مخرج الغالب، لكن ما تقوم به SQL هو أكثر بكثير من مجرد الاستعلام. ها هنا سؤال قبل الجلسة القادمة: هل SQL لغة برمجة؟

رقم الحلقة (4) هل SQL لغة برمجة؟

أكرر السؤال: هل SQL لغة برمجة؟

الإجابة القصيرة (لمن كان وقته ضيقاً): لا...

الإجابة الطويلة (لمن كان ينوي أن يكمل القراءة): لا. ليست SQL بصورتها الأصلية من لغات البرمجة بالمعنى التقليدي للغات البرمجة؛ لماذا؟ لأنها في الأصل لا تملك خصائص لغات البرمجة. هي لغة تحديد (ماذا) تريد من بيانات أكثر من (كيف) تعالج البيانات، وهي في الأصل لا تدعم الإجرءات ولا البنى البرمجية التي تدعمها أية لغة برمجة تحترم نفسها (مثل التكرارات والجمل الشرطية). أقول في الأصل لأن العديد من الإضافات قد تمت على هذه اللغة من قبل المؤسسات المعيارية (التي تضع المعايير standards)، ومن قبل كل منتج لبرنامج إدارة قواعد بيانات. على سبيل المثال Oracle أنشأت لغة البرمجةPL/SQL وتعني SQL الإجرائية، والتي لا تعدو أن تكون SQL مع دعم الإجراءات والبنى الأخرى المشار إليها آنفاً.

على كل حال، SQL لغة قوية في مجالها كلغة قواعد بيانات، وهي إلى حد ما بالنسبة إلى مبرمجي ومديري قواعد البيانات تساوي بشكل أو بآخر لقمة العيش! هذه نقطة جوهرية، ومهمة إلى أبعد حد، لا أستطيع إعطاءها حقها من التشديد، إذا لم تستفد غيرها من كل هذا النقاش فكفى بها: لا بد أن تتقن التكلم بلغة SQL. لا أريد هنا أن أخوض في تفاصيل اللغة وما تقوم به من وظائف، هذا مجال كتب، وربما تكون لنا جولات معها فيما بعد، لكن ينبغي أن تفهم جيداً أن كل تعاملك مع قاعدة البيانات (مباشرة أو غير ذلك، علمت أم لم تعلم) هو عبر هذه اللغة. ربما أنت الآن في الأكسس تستخدم الكثير من المعالجات لكن SQL هي هناك دائماً خلف الكواليس تعطي الكلمة الأخيرة. حتى نماذج الكائنات من أمثال ADO على سبيل المثال لمن كانت عنده فكرة، توفر طرقاً (مثل الإضافة والتعديل) هي في النهاية غطاء لأوامر SQL. من يبرمج باستخدام PL/SQL مع الأوراكل يعرف معنى هذا جيداً. إذا كنت تبرمج برامج حقيقية، فأنت ستحتاج إلى الاستخدام (البارع) المباشر لهذه اللغة عاجلاً أم آجلاً بإذن الله.

العبرة المستفادة هنا: لا بد أن تتعلم SQL، يتفاضل الكثير من المبرمجين لأجل قواعد البيانات بينهم في مدى إجادتهم للغة، الأفصح في التحدث بـSQL، هو المبرمج الأكثر مهارة بشكل عام.

ذكرت آنفاً أنك تخبر برنامج إدارة قواعد البيانات بالهيكل المنطقي للبيانات وهو يتولى إدارتها فيزيائياً، سوف نتكلم عن هذا لاحقاً بإذن الله عندما نتكلم عن تصميم قواعد البيانات. من الجيد أن تتذكر الآن أنك في النهاية تخبر DBMS بذلك باستخدام SQL (مباشرة أو عبر معالج ما).

مبرمج الشبكات يجب أن يعرف الكثير عن الشبكات بالإضافة إلى البرمجة، ثم ينبغي أن يعرف الكثير عن برمجة الشبكات باستخدام لغة محددة أو أكثر. صحيح؟... بنفس المنطق، المبرمج لقواعد البيانات ينبغي أن يعرف الكثير عن قواعد البيانات (و) عن البرمجة (و) عن استخدام لغة محددة أو أكثر لبرمجة تطبيقات قواعد البيانات. أنت تعلم ما يعنيه هذا: يعني أن أمامك الكثير لتتعلمه كمحترف. أنا لست هنا لأحيل حياتك إلى جحيم، لكن هناك أشياء في هذه الحياة لا تستطيع اختصارها. لا تختصرها، لكن أدّها باختصار إن أردت. أعني لا تحذفها، لكن خذ منها ما يتيسر لك. نحن هنا في نقاش أخوي، وهذا يقتضي عدم التكلف، والصراحة.

هل تعلم القدر الكافي عن قواعد البيانات؟ عن البرمجة؟ عن لغة برمجة واحدة على الأقل (ولتكن VBA، لا مانع)؟ أجب عن ذلك، ثم عد إلى السؤال الأول في هذه السلسلة: هل أنت مبرمج (من أجل) قواعد البيانات؟

سأتركك لتجيب عن السؤال، لكنني سأبدأ إن شاء الله من المرة القادمة في الحديث عن أول جزء وهو الأهم في سلسلتنا هذه... تحليل وتصميم قواعد البيانات... أظن أننا من الممكن أن نستصحب مثالاً عاماً (ومثيراً للاهتمام في الوقت ذاته) لتجسيد المفاهيم، وربما اقترحتم ما ترونه مناسباً، وإن كنت شخصياً أرشح قاعدة بيانات تطبيق مخزن مبيعات لما لها من طلب واسع...

حتى ذلك الحين: هل أنت.....؟

رقم الحلقة (5) الطبقات في عالم قواعد البيانات وتطبيقاتها

المفاهيم الأساسية مهمة جداً في أي مجال، وفي عالم الحاسوب هنالك الكثير من المفاهيم التي ينبغي الإلمام بها، والمشكلة أنها تهمل بشكل غريب، خصوصاً من قبل الواردين الجدد على هذا العالم؛ قد يكون السبب اختفاء هذه المبادئ خلف أكوام من التقنيات و(الطبقات)، التي غلفت المفاهيم بألف غلاف وصورتها بألف صورة حتى (تكاثرت الظباء على خراش... فما يدري خراش ما يصيد)! لعلك أيها القارئ الفاضل تلاحظ أنني ألمس الموضوع الشديد العمق مجرد لمسة، ولا أعطيه حقه من التفصيل، لأن ذلك بالفعل يطول، ويخرج عن نطاق هذه السلسلة، وعن إمكانيات كاتبها...

لنعد الآن إلى تلك (اللمسات)، ونتحدث قليلاً عن البرمجيات بشكل عام، وعن تطبيقات قواعد البيانات بشكل خاص...

تأمل الكلمة (تطبيق قواعد بيانات)، هذا يعني عادة أن ثمة تطبيقاً ما (برنامجاً كبيراً)، يستند إلى قاعدة بيانات، أي أن وظيفته الأساسية هي الأخذ والعطاء من قاعدة البيانات مع ما بينهما من المعالجة. هذا يقودك إلى النموذج الشهير: إدخالات، معالجة، إخراجات. بالمناسبة، يحلو للناس عادة تعريف المخرجات على أنها (المعلومات)، وعلى هذا فإن قاعدة البيانات هي مخزن للمادة الخام (البيانات)، والمعلومات هي هذه البيانات بعد معالجتها. نحصل على المعلومات عادة في شكل تقرير أو استعلام.

مرة أخرى، تطبيق قاعدة بيانات يحوي جزءين: قاعدة البيانات التي عرفنا أن برنامج إدارة قاعدة البيانات (يمكن أن ندعوه باسمه DBMS) يديرها وفيها تجد كل البيانات، والتطبيق الذي يتعامل المستخدمون مع قاعدة البيانات من خلاله، فهو يشكل واجهة أمامية يبنيها المبرمج من أجل المستخدم العادي. لماذا يبني المستخدم مثل هذه الطبقة الأمامية؟ قلنا من قبل إن ذلك يجنب المستخدم العادي أي تعقيدات للتعامل مع الـ DBMS، مثل ضرورة استخدام الـSQL أو أي غلاف خارجي لها. كذلك فإن هذه الواجهة من المفترض أن تخدم عدة أغراض بكفاءة؛ باختصار مخل، يجب أن تجعل الواجهة عملية إدخال البيانات (واسترجاعها بعد المعالجة):

1. دقيقة (خالية من الأخطاء) قدر الإمكان.
2. سهلة قدر الإمكان.
3. سريعة قدر الإمكان.
4. (ممتعة قدر الإمكان)؟ نحن لا نصمم ألعاباً هاهنا... لا أحد يتمتع بإدخال البيانات، صدقني. بعض المبتدئين يحاول أن يجعل برنامجه حديقة ملاهي على حساب العمل الحقيقي...

الترتيب المذكور ليس حصرياً، ويختلف باختلاف نوع التطبيق. أحياناً تكون السرعة هي الأولى على حساب أي عامل آخر، وهكذا. بالمناسبة أيضاً، من أجل ربط ما تقرأه هنا بالمصادر التي من الممكن (من الضروري؟) أن تراجعها فيما بعد، تسمى هذه الواجهة عادة بالطرف الأمامي (Front-end)، بينما تسمى قاعدة البيانات بالطرف الخلفي (Back-end). نحن في هذه السلسلة يهمنا أكثر هذا (الطرف الخلفي)، من الواضح أن الخلف لا يعني دائماً الأقل أهمية (أذكر أيام الجامعة، كان يحلو لبعض أذكى الطلبة الجلوس في الخلف دائماً)...

نحن الآن نعود إلى ما أشرنا إليه من قبل حول وجود طبقات في عالم قواعد البيانات وتطبيقاتها، وسأحاول الآن إيجاز بعض المفاهيم الهامة وتقديم بعض المصطلحات الجديدة (جديدة في هذه السلسلة، لكن من المفترض أنك سمعت بها كثيراً من قبل إن كنت حقاً مبرمج قواعد بيانات). أرجو التركيز هاهنا لأن السرد مقتضب نوعاً ما:

لنسمي مجموع قاعدة البيانات مع التطبيق نظام قواعد بيانات... إذا كانت مكونات هذا النظام (الطرف الأمامي والخلفي) تشكل وحدة واحدة، وبالطبع تقع في جهاز واحد، فإن هذا النظام يسمى نظام قواعد بيانات مكتبياً (Desktop Database System). نعم، كما تقول بالضبط، الأكسس ينتمي إلى هذا النوع من الأنظمة، أنت تجد الجداول مع النماذج مع التقارير في نفس الملف وتدار بنفس البرنامج. نعم نعم، يمكنك الآن تقسيم قاعدة البيانات ووضع الطرف الخلفي (الجداول) في ملف منفصل على جهاز ما، في حين تقع بقية المكونات في ملف آخر (على جهاز آخر ربما). لكن هذا مجرد تطوير لا يغير من حقيقة أن الأكسس في الأصل هو نظام قواعد بيانات مكتبي. لاحظ أنك في حالة التقسيم يجب أن تحتفظ بارتباطات للجداول في ملفك الثاني.

إذا كان الطرف الخلفي مستقلاً تماماً ببرنامج إدارته، والطرف الأمامي له برامجه الخاصة التي تشغله، فإن هذا النظام يسمى نظام قواعد بيانات زبون – خادم (أو عميل – ملقم)، وبالفصيح (Client – Server Database System). لاحظ أن الطرفين قد يجتمعان في نفس الجهاز، وإن كان الأصل أن يكونا منفصلين على جهازين مختلفين على شبكة محلية أو خارجية. مثال؟... من أشهر الأمثلة على ذلك أنظمة Oracle. على الرغم من توفر نسخ من أوراكل للاستخدام المكتبي على جهاز واحد (Oracle Personal)، لكن الغالب هو استخدام نسخة Oracle Server على جهاز، وتحميل Oracle Developer على بقية الأجهزة لتشغيل النماذج والتقارير. أنا الآن أتكلم عن الصورة الكلاسيكية، على الشبكات المحلية. الزبون يرسل البيانات عبر الشبكة إلى الخادم (حيث الـ DBMS) أو يطلب البيانات من الخادم عبر الشبكة، والخادم (يخدم) بالطبع، أعني يستجيب للطلبات، وبما أنه

DBMS، فهو يقوم أيضاً بالوظائف الأخرى التي لم نسهب فيها كثيراً من قبيل التأكد من أمان وسلامة قاعدة البيانات عبر قواعد وشروط السرية والتكامل التي تم تحديدها مسبقاً.

إذاً، فإن لدينا في هذا النظام الثاني طبقتين منفصلتين، ولأن متخصصي الحاسوب (كغيرهم من المتخصصين في كل المجالات) يحبون المصطلحات التي تجعل الأمور تبدو معقدة، فإنهم رأوا هذا الشكل وقالوا: هذا Two-Tier Architecture، أي بنية ذات طبقتين. أحياناً تكون معظم العمليات الحسابية والمنطقية، التي تشكل جزء معالجة البيانات الذي تكلمنا عنه سابقاً، في جانب الـclient (أي الزبون)، وهنا يسمى هذا الزبون سميناً (لا تضحك رجاءً، اسمه في المصادر الرسمية Fat-client)، أو أحياناً يسمونه Rich-client أي الزبون الغني (دعنا نسمه الزبون الدسم). إذا كانت معظم المعالجة تتم في جانب الخادم (server)، فإن الزبون يسمى نحيفاً (Thin-client). لا أدري من أين يأتون بهذه التسميات، لكن من المعروف أن متخصصي الحاسوب عندهم هواية أخرى غير التعقيد، وهي التسميات الطريفة.

من أجل تكملة الموضوع فقط (على الرغم من أن هذا لا يهمنا هنا)، هنالك الآن Three-tier بل و N-tier Architectures، أي ثلاث أو عدد غير محدود من الطبقات، ومن أمثلته وجود الواجهة على جهاز مستخدم الإنترنت (صفحة في مستعرض الوب)، التي تتعامل مع خادم وب (web server) على جهاز آخر، وهذا الأخير بدوره يطلب البيانات من خادم قواعد بيانات (database server)، حيث الـDBMS، وقد يكون على جهاز ثالث. لاحظ أن المسافة الفعلية بين هذه الطبقات غير محددة؛ قد تكون سنتيمترات قليلة في نفس الجهاز، مترات معدودة في نفس الغرفة، أو كيلومترات كثيرة في مدن مختلفة من العالم.

حسناً، قد أخذنا الكلام قليلاً، لكن حتى العودة بإذن الله، حاول أن تفكر مرة أخرى: هل أنت.....؟

رقم الحلقة (6) الفرق بين البنى المنطقية والفيزيائية

كنت قد أشرت قبلاً إلى البنية المنطقية للبيانات والتفاصيل الفيزيائية عند الحديث عن برامج إدارة قواعد البيانات. حان الوقت الآن لمزيد من التوضيح. خذ بعض المصطلحات: Physical design - Logical design - View design

كبداية، أحب أن أوضح أن هذه المفاهيم هي عبارة عن مستويات مختلفة، يمثل كل منها منظوراً معيناً إلى البيانات من زاوية مختلفة. لكن ليس المقصود بهذا التقسيم أن يمثل خطوات يتبعها المصمم أو المبرمج.

من أجل توضيح هذه الفكرة، أحتاج إلى الكلام عن مفهوم عام مهم يصادفك دائماً في علوم الحاسوب. ستسمع دائماً عن البنية الفيزيائية والبنية المنطقية. المقصود بالبنية الفيزيائية هي التطبيق الفعلي الذي يتم على مستوى الآلة، والمقصود بالبنية المنطقية هي طريقة تفكيرنا نحن بالموضوع بغض النظر عن كيفية تطبيقه فيزيائياً على مستوى الآلة. مثلاً، أنا أكتب هذا الكلام الآن في مستند وورد قبل أن أنسخه إلى المنتدى، أنا أفكر بهذا المستند على أنه وحدة واحدة تسمى ملفاً، وهذا الملف يحوي هذه الحلقة التي هي مجموعة كلمات عربية. هذه هي طريقة تفكيري كمستخدم للحاسوب. لكن الحقيقة أن ملفي هذا يحفظ في القرص الصلب على هيئة جزيئات ممغنطة صغيرة كل واحدة منها تسمى بت (bit) وتمثل إما صفر وإما واحد. مجموع الأصفار والواحدات هذه يمثل كوداً يمكن أن يترجم فيما بعد على أنه الكلمات العربية التي نفهمها. الحقيقة أيضاً أن هذه الـ(بتات) قد لا تحفظ معاً كوحدة واحدة، بل قد تبعثر في أماكن مختلفة تسمى (clusters)، لكن نظام التشغيل يعرف كيف يجمعها معاً لأنه يحتفظ بالمعلومات اللازمة عن موقع وأجزاء كل ملف في جدول خاص يسمى (FAT File Allocation Table). هذا هو الملف على المستوى الفيزيائي، في حين أن التمثيل الأول هو البنية المنطقية التي أفكر بها أنا.

بشكل مشابه، قاعدة البيانات لها أكثر من مستوى تبعاً لمن ينظر إليها: برنامج إدارة قواعد البيانات، أم المصمم والمبرمج (مرة أخرى، أنت ولا فخر)، أم المستخدم النهائي. هذه المستويات هي:

1. المستوى الفيزيائي (physical level).
2. المستوى المنطقي (logical level).
3. مستوى المعاينة (view level).

سبب وجود هذه المستويات المختلفة هو مفهوم الطبقات الذي أفضنا فيه قبل. البنية الفيزيائية للبيانات في قاعدة البيانات معقدة للغاية، على مستوى ملفات نظام التشغيل، ونظام التشغيل بدوره يتصرف بحكاية الـ(بتات) إياها. على كل حال، سبق أن قررنا أن ننشئ طبقة تتولى هذه التعقيدات، وسمينا هذه الطبقة (من يقول...؟) برنامج إدارة قواعد البيانات. ثم جاء شخص من IBM (عام 1970 للميلاد) يدعى E. F. Codd يقول: ما رأيكم أن نفكر بالبيانات على شكل جداول (relations بلغة Codd) بصفوف وأعمدة تربط بينها علاقات بواسطة أعمدة مشتركة؟ تفكيرنا بالبيانات بهذه الطريقة منذ ذلك الحين يسمى البنية المنطقية لقاعدة البيانات. بالطبع لا توجد جداول بصفوف وأعمدة على القرص الصلب، لكن هذا لا يهمنا مادمنا في طابق المستوى المنطقي. أصحاب الطابق السفلي عليهم أن يتعاملوا مع مشكلة النفايات التي نلقيها من الشرفة دون أن يرانا أحد. لكن، من قال إن هناك حدود لترف الإنسان؟ الغالبية وجدت أن حكاية الجداول هذه وما يرافقها من خزعبلات ما زالت معقدة وغير راقية. هم يريدون أن (يعاينوا) البيانات على مستوى يليق ببني آدم، وكما تعود بنو آدم أن يفهموه ويتعاملوا معه: نماذج مرتبة، وتقارير منمقة أو ما شابه.

كما هو واضح من التقسيم السابق، برامج إدارة قواعد البيانات هي اللاعب الأساسي في المستوى الفيزيائي، في حين نقبع نحن كمصممين ومبرمجين لقواعد البيانات في الطابق الثاني، أقصد المستوى المنطقي. المستخدمون النهائيون هم سكان الطابق الثالث. هناك استثناء يسير بالنسبة لمديري قواعد البيانات (DBA Database Administrators)، وهو أنهم على الرغم من وجودهم معنا في المستوى المنطقي، لكنهم أحياناً ينبغي أن يتعاملوا مع المستوى الفيزيائي لإنجاز بعض المهام. في الأكسس من الصعب أن ترى ذلك، لكن المتعاملين مع برامج مثل الأوراكل على سبيل المثال يعلمون أن مدير قاعدة البيانات (وهو الشخص الذي ينشئ قاعدة البيانات، لكنه ليس بالشرط أن يكون الذي صممها) يحدد بعض الخصائص مثل الملفات التي ستحوي (tablespaces)، وفي أي قرص صلب، وحجم البلوك وما إلى ذلك مما لا تحتاج إلى أن تشغل به بالك الآن.

ملحوظة أخرى تتعلق بالمستوى الأخير. مستوى المعاينة لا يخفي فقط تعقيدات المستويات الأدنى، ولكنه يلعب دوراً مهماً في مجال أمان وحفظ سرية المعلومات. أوضح مثال لهذا المستوى هو الاستعلامات التي يمكن توليدها من الجداول. لا تظهر الاستعلامات كل حقول الجداول، لأنه ينبغي أحياناً ألا يرى المستخدم العادي كل الحقول. المزيد من التفصيل تجده إن شاء الله في حلقات تالية حين نتكلم أكثر عن تصميم قواعد البيانات.

لذا، فإننا كمصممين ومبرمجين نبدأ من عند المستوى المنطقي حين نحلل ونستخدم أي أدوات من قبيل مخطط الكائنات والعلاقات، إلى حين الوصول إلى التصميم الأخير للجداول. ثم نبدأ في إعداد منظور المستخدم النهائي من استعلامات ونماذج حين نعد التطبيق الذي سيستخدمه هذا المستخدم. ولكن، هل أنت.....؟

رقم الحلقة (7) دورة حياة النظام: أهمية التحليل

دعونا الآن نتقدم قليلاً ونلمس الماء بأطراف أصابعنا... كفى لهواً على الشاطئ...

خذ الصورة الشائعة التالية: أحمد طالب حاسوب في السنة النهائية، وربما يكون قد تخرج منذ بضع سنين، لا فرق، اختار برمجة تطبيقات قواعد البيانات كسبب للقمة العيش، أخبره صديقه أن جاره صاحب محل (الأمانة) يريد برنامجاً لإدارة مبيعات المحل... يؤكد أحمد لصديقه أن البرنامج سيكون عنده بإذن الله في غضون أسبوعين، ويعود إلى منزله ذلك اليوم وكله حماس، لقد حان الوقت لأثبت نفسي... يجلس أمام الحاسوب، يفتح الأكسس ويبدأ في الطرق والسحب، أعني الضرب على لوحة المفاتيح، وسحب الفارة هنا وهناك...

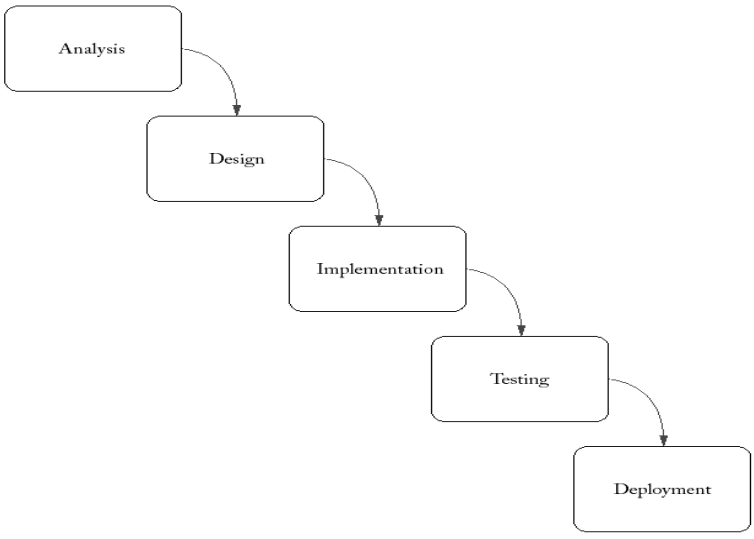
ماذا تفعل يا أحمد؟... أبني برنامج قواعد بيانات بالطبع، لا وقت للعب، اذهب واكتب مذكراتك السخيفة التي لن يقرأها أحد، ودع الخبراء يعملون!!...

على رسلك يابني، لو ذهبت تعد بيضة مقلية لقضيت وقتاً أطول قبل أن تصل إلى مرحلة الـ(طشش)، عندما تكسر البيضة على الزيت... (أوردها سعد وسعد مشتمل... ما هكذا يا سعد تورد الإبل)!

ما يفعله الكثيرون من أمثال أحمد، أنهم يكسرون دورة الحياة... لا أتكلم عن دورة حياة البعوضة هاهنا، ولكن عما اصطلح الناس عليه لوصف مراحل إعداد التطبيقات البرمجية. تميزت البرمجيات بأنها منتجات تعتمد إلى حد كبير على خبرة ومهارة المبرمج، ولا يمكن أتمتة صناعتها كما هو الحال في بقية المنتجات. إنتاج البرمجيات يتداخل فيه الفن مع العلم بشكل يصعب معه التمييز بينهما. اختلف الوضع قليلاً الآن، وحاول الناس (تقنين) هذه العملية باستنباط قواعد لإنتاج البرمجيات من واقع الخبرة المتراكمة في الماضي. هذه القواعد أثبتت إلى حد ما أنها تعمل بشكل أو بآخر، على الرغم من أنها ليست مطلقة، ومرنة إلى حد كبير. تم ترتيب هذه الأفكار وجمعها في شكل عمليات (processes)، وجمع هذه العمليات في شكل نماذج (models)، ولهذا تجد الآن ما يسمى بـ(Software Process Model) في مادة هندسة البرمجيات. لن أترجم المصطلح السابق لأنني لا أريد أن أضيف مشكلة أخرى إلى مشاكل تعريب المصطلحات التي تزخر بها المصادر العربية.

موضوع هندسة البرمجيات هذا موضوع مهم جداً، بالذات بالنسبة للمشاريع الكبيرة. صحيح أن مشاريعنا الصغيرة نسبياً، والتي تحتل الصدارة في نقاشنا في هذه السلسلة، لا تحتمل تطبيق نظريات هندسة البرمجيات الثقيلة، لكن هذا لا يعني أن المبادئ الأساسية لا تبقى صالحة للتطبيق (بل أقول، ضرورية التطبيق) في أي برنامج، بما في ذلك برامج قواعد البيانات. هذا يضيف إلى قائمة ما ينبغي دراسته بنداً آخر: اقرأ في هندسة البرمجيات إن سنحت لك الفرصة.

من أشهر النماذج الكلاسيكية التي تجدها في أدبيات هندسة البرمجيات، نموذج يرتب العمليات الأساسية في إنتاج البرامج تدفقياً (يدعونه النموذج الشلالي Waterfall Model). هذا النموذج يقترح أن دورة حياة إنتاج البرنامج تمر في مراحل محددة، مستقلة عن بعضها، ومتتابعة. اكتشفوا بالطبع أن هذا غير ممكن دائماً، وأدخلوا الكثير من التعديلات مما أنتج نماذج أخرى، كالنموذج المتكرر (Iterative) و النموذج المبني على المكونات (Component-based Model). هذا ليس مهماً الآن. سأتبنى في هذا النقاش النموذج الكلاسيكي لأنه أوضح وأقرب إلى منطق تسلسل العمليات. اخترت لكم هذا الرسم لأنه من أوضح ومن أكثر الأشكال تعبيراً عن دورة حياة البعو... أقصد البرنامج":



كما تلاحظ، يفترض النموذج أن هناك ما لا يقل عن خمس عمليات (كل منها يمثل مرحلة مستقلة)، ينبغي أن تتم من أجل إنتاج التطبيق. هذه العمليات على الترتيب هي:

1. التحليل.
2. التصميم.
3. التنفيذ.
4. الاختبار.
5. التسليم.

يفترض النموذج أيضاً أن كل مرحلة تنتهي تماماً (بمنتج ما)، قبل أن تبدأ المرحلة التي تليها. مثلاً، التحليل ينتهي بمستند SRS (System Requirements Specification) أو (مواصفات متطلبات النظام). هذا طبعاً في المشاريع الرسمية الكبيرة كما أسلفنا، على مستوى شركات برمجة وفرق مبرمجين وغير مبرمجين، لكنك مطالب بشكل أو بآخر بكل هذا ولو على مستوى مصغر. بالمناسبة، رسمياً، وفي الشركات الكبيرة، لكل عملية فريق متخصص من شخص أو أكثر، حسب حجم ونوعية التطبيق والشركة. لذلك تسمع عن وظائف مثل محلل أنظمة، ومصمم، ومبرمج. وكثيراً ما يشار إلى المبرمج في الوسط ببعض الاستهانة على أنه ليس أكثر من (مشفر coder). ما الذي يعنيه كل هذا بالنسبة لنا هنا ونحن نتحدث عن مشاريع صغيرة (على ضرب نظام مخزني أو نظام شؤون موظفين على مستوى محل أو شركة صغيرة)؟ يعني الكثير لو لاحظت... يعني أنك هنا تلعب دور الجميع ولا فخر: أنت المحلل والمصمم والمبرمج والمختبر والمسلم. ما شاء الله، الله يعطيك العافية. تذكر صديقنا أحمد في بداية الحلقة؟ لقد كان غافلاً عن هذه الحقيقة المروعة، ولذلك لم يحسن لعب الدور...

قبل أن نشرع في الكلام عن تطبيقات قواعد البيانات خاصة، دعني أمر مروراً سريعاً على المراحل السابقة حتى نأخذ فكرة عما يعنيه (إنتاج برنامج). أغلب المراحل تخفي الكثير من التفاصيل خلف تلك الكلمة المفردة التي توصف بها. مثلاً، هناك خطوة إضافية قبل التحليل (أحياناً تضم مع التحليل وأحياناً تفصل عنه في بعض المراجع)، ينبغي القيام بها في الأصل، وهي دراسة الجدوى (feasibility). لاحظ أن هذه الخطوة لن تكون أنت المطالب بها في الغالب، بل الجهة التي ستشتري منك الرنامج. هذه الخطوة ينبغي أن تكشف بوضوح إن كانت هناك حاجة حقيقية لبناء النظام، إن كانت هناك إمكانية حقيقية لبناء النظام، إن كان هناك مكسب حقيقي من بناء النظام. أحياناً قد تكون هناك حاجة، لكنك ستخسر أكثر مما تستفيد. مثلاً، لا ينبغي أن يكون السبب الوحيد لبناء النظام هو (لأن الحاج سعيد يستخدم واحداً في محله)، وبعض الأعمال الصغيرة لا تستحمل شراء الأجهزة وسعر التطبيق مع راتب عامل الحاسوب.

التحليل هو أساس النظام البرمجي (أنا الآن أستخدم كلمة النظام البرمجي لكي أتيح المجال للكلام عن قواعد البيانات فيما بعد). قلت إن التحليل هو الأساس الذي إن لم تعتن به، انهار النظام ولو بعد حين. الخطوة الأولى دائماً خطوة مهمة جداً، وقد تكون الأهم. ما بني على باطل فهو باطل، والعكس صحيح في الغالب. مرحلة التحليل يمكن تلخيصها في كلمتين: الجمع والفهم. الهدف منها هو فهم كل وظائف النظام بشكل واضح ودقيق، مع فهم كل القيود التي ينبغي أن يعمل النظام في حدودها. في سبيل ذلك، أنت تجمع كل ما يمكن أن يساعد على هذا الفهم. ذلك يشمل أي نماذج مطبوعة مستخدمة يدوياً، أي معلومات مستقاة من الأشخاص مباشرة في مقابلات أو عبر استبيانات، أي لوائح متبعة، و(ركز على هذه:) كل ما يمكن استنتاجه وملاحظته مما لم يمكن جمعه بالوسائل الأخرى سواء بالمراقبة أو (ركز على هذه أكثر:) بالتفكير!

التفكير؟... نعم. التفكير مهم جداً في كل المراحل وبالذات في المراحل الأولى من تحليل وتصميم وبرمجة. لماذا أذكر هذا؟ لأن الكثير يتهرب من التفكير! التفكير متعب ومرهق للذهن، أعلم هذا، ولكنه مطلب لا مفر منه في عالمنا. أنا أعني بالتفكير هاهنا الوقت الذي ستقضيه تدور في الغرفة، أو تقطع ذلك الممر جيئة وذهاباً لأكثر من ساعة تحاول أن تتخيل كيف يصنع المستخدم هاهنا أو كيف يمكن تنفيذ هذه الطريقة هنالك. من الممكن استخدام كرسي هزاز، لكنني لم أجرب ذلك بعد. إذا حاولت استخدام الكرسي العادي كهزاز، فإنك ستقطع الأرضية تحت الأرجل الخلفية إن كانت من المشمع، لذا كن على حذر!

الكلام عن التحليل يطول، لكنني سأؤجل بعض الملحوظات إلى حين حديثنا عن قاعدة بيانات نصممها معاً بإذن الله. وإلى أن أكمل التعليق على المراحل الأخرى، أتركك الآن مع سؤال مألوف: هل أنت.....؟

رقم الحلقة (8) دورة حياة النظام: نظرة على التصميم وبقية المراحل

قلنا إن التحليل هو الخطوة الأولى (بعد دراسة الجدوى) التي من المفترض، رسمياً، أن تنتج مستنداً يشرح كل متطلبات النظام. من المفهوم طبعاً أن من يشرح كل متطلبات النظام يفهم جيداً ما يتكلم عنه، ثم هو ينقل هذا الفهم بواسطة هذا المستند إلى من سوف يصمم النظام. في حالتنا، ذكرت أنك، ولا فخر، هو المحلل والمصمم، لذا قد تتجاوز عملياً كتابة هذا المستند بالشكل الرسمي لأنك سوف تكتبه لنفسك على أي حال. هذا طبعاً لا يعفيك من وجود توثيق للمتطلبات بشكل أو بآخر (بصورة غير رسمية).

على الرغم من وجود الكثير مما يمكن (وينبغي) أن يقال عن التحليل، إلا أنني كما أشرت في الحلقة السابقة سوف أوجل هذا قليلاً بإذن الله...

لنفترض الآن أن لدينا فهماً واضحاً وشاملاً (وصحيحاً بالطبع) لكل متطلبات النظام. الخطوة التالية هي التصميم. والتصميم كلمة عامة تتكون من عدة مستويات. أعني أنك لا تجد عادة تصميماً واحداً للنظام البرمجي ككل، لأن ذلك غير ممكن ببساطة. النظام البرمجي يعكس غالباً نظاماً يدوياً متشعباً. عند تمييز الأجزاء الأساسية الكبيرة لهذا النظام فأنت تصمم بنية النظام (Software Architecture Design). في نظام لفواتير الكهرباء على سبيل المثال، هناك جزء يتعامل مع القراءات، وآخر للتسديدات، وثالث للتسويات ربما، ثم هناك الجزء الخاص باحتساب الفواتير وطباعتها.

عند النظر إلى النظام البرمجي من زاوية المهام التي يؤديها، وتسلسل هذه المهام، وتدفق البيانات من مهمة إلى أخرى، فإنك تصمم ما يعرف بالنموذج الوظائفي (Functional Model)، الذي يعلم الأكثرون أحد أشهر أشكاله، وهو مخطط تدفق البيانات (DFD - Data Flow Diagram). هذا الشكل يوضح المهام التي يقوم بها النظام ممثلة بمربعات، يربط بينها أسهم، مع التركيز على تحديد البيانات التي تدخل كل مربع والتي تخرج منه إما إلى مربع (مهمة) أخرى، أو إلى مكان ما تحفظ فيه، أو إلى مخرجات تطبع.

دعني أقودك الآن إلى زاوية أخرى ننظر منها إلى النظام البرمجي، وهذه الزاوية تهمنا كثيراً هنا لأن تركيزنا سيكون عليها بإذن الله... المهم هنا هو بيانات النظام وبنيتها، وليس المهام التي سوف تؤدى عليها. هذا المنظور جزء لا يتجزأ من أنظمة قواعد البيانات. في هندسة البرمجيات، يسمون هذا التصميم (Semantic Data Model)، وفيه يبينون البنية المنطقية للبيانات في النظام. من أشهر أدوات هذا النوع من التصميم، مخطط الكائنات والعلاقات (ERD - Entity Relationship Diagram). مرة أخرى، سنؤجل الكلام عن هذا لما بعد، لأنه موضوعنا الأساسي، وأريد أن أنتهي من ذكر بقية المراحل على عجل حتى نعود إلى موضوعنا بمزاج رائق (أرجو ألا يكون مزاجك قد تعكر منذ الآن...).

من مهام المصمم أيضاً تصميم واجهة المستخدم، وهذه قصة طويلة (حزينة بعض الشيء)، وعلم وفن بحد ذاته ليس هاهنا مجال التفصيل فيه، لكن لا بد من القول بأن واجهة المستخدم جزء حيوي من النظام وقد يكون قاتلاً أحياناً. تذكر من نقاشنا السابق أن واجهة المستخدم من نماذج وتقارير تشكل الطرف الأمامي الذي يراه المستخدم ويتعامل معه، وبغض النظر عن المعالجة التي قد تكون في هذا الطرف أيضاً، أشرنا إلى أن واجهة المستخدم ينبغي أن تتمتع بعدة خصال، أبرزها سهولة ودقة وسرعة الاستخدام. أن تكون الواجهة جذابة، هذا أمر مطلوب بالتأكيد، لكن ما هو معيار (جذابة؟)، وما هو الثمن الذي سيدفعه النظام من أجل هذه الـ(جذابة؟)... هذه أسئلة يجب أن تسألها نفسك قبل أن تضيف نصف طن من الصور والألوان ومكونات ActiveX إلى واجهة برنامجك. تذكر أن سهولة وفعالية الاستخدام هي الأولى (الفعالية تشمل السرعة والدقة). أحياناً (الجاذبية) الشديدة تعرقل الاستخدام السهل. هناك نوع من الأناقة البسيطة التي تفوق في جاذبيتها الأناقة المتكلفة.

نقطة أخرى تتعلق بواجهة المستخدم لا بد من الإشارة إليها. للأسف، من الملاحظ أن بعض البرامج حين تفشل في توفير الفاعلية الحقيقية، تلجأ إلى تغطية ذلك بالواجهة المبهرة. أقول للأسف، لأن هذه مشكلة حقيقية ملموسة ومنتشرة، وفي طريقها لتصبح ظاهرة إن لم ننتبه إليها. معظم هذه البرامج تقبل في البداية، وقد رأينا وسمعنا عن الكثير منها، لكنها تختفي في النهاية، ولا تستطيع الصمود. المشكلة أنها تترك أثراً سيئاً للغاية. أصبح المستخدم العادي لا يثق في منتجات المبرمجين المحليين، وأصبحت برامج فعالة ترفض لمجرد افتقارها إلى تلك الواجهة الفنية، ولسان حال المستخدم يقول: إذا كان ذلك البرنامج (المحترف) قد فشل، فكيف بهذا البرنامج الحاف الجاف؟ أختم الحديث عن هذه النقطة بسؤال: كيف ترى واجهة البرامج العالمية الرصينة بالمقارنة مع واجهات البرامج المحلية؟ ربما يجعلنا هذا نفكر مرة أخرى في معنى تصميم واجهة المستخدم.

لا ننسى أن نذكر أن التصميم قد يمتد ليشمل التخطيط إلى مستوى الوحدات البرمجية من إجراءات ووظائف ومعاملاتها. وهنا يتداخل طور التصميم مع طور البرمجة، ويصبح المبرمج بالفعل مجرد مترجم لما يمليه عليه المصمم. هذا لا يعني أن مهمة المبرمج مهمة سهلة يمكن لأي موظف ليس لديه ما يقوم به أن يتولاه. لا. لو كان الأمر كذلك، لما كان المبرمجون ينالون واحداً من أعلى المرتبات اليوم... وظيفة المبرمج وظيفة حساسة للغاية، ومن المشاكل الحقيقية التي نعاني منها أننا وصلنا إلى قناعة بطريقة ما، بأن البرمجة يمكن القيام بها بالاستكشاف كما لو كنت تتعلم برنامج الوورد بالتقليب في قوائمه والعبث هنا وهناك.... لا يا صديقي لا، البرمجة

لها مبادئ وأصول وطرق وتقنيات يجب أن تتعلمها أولاً، ثم تزيد قيمتك كمبرمج مع ازدياد خبرتك... لا تحاول أن تتعلم البرمجة بطريقة (من كل بستان زهرة... ومن هنا وهناك). لا أريد الخوض في هذا حقاً، لأنني إن بدأت الكلام في هذا الموضوع ذي الشجون فلن أقف بسهولة، وهذا لن يعجب أحداً.

عند اكتمال تنفيذ التصميم بتقنية معينة (كلمة التقنية هنا عامة تشمل لغة البرمجة)، فإن عملية الاختبار الرسمية تبدأ. لاحظ أنه ليس هنالك مبرمج لا يختبر الكود الذي ينتجه، لكن اختباره لنفسه لا يعتمد هنا، ومن الضروري كذلك اختبار الكود ككل بعد تجميع ودمج الوحدات البرمجية التي تم تطويرها (ربما) من عدة مبرمجين. يمكن القول إن طور الاختبار هو من أكثر المراحل التي يتم تجاهلها والاستخفاف بها، وهذا واحد من أهم أسباب ضعف وركاكة كثير من المنتجات البرمجية: أنت تنشر غسيلك بين الناس، في حين كانت لديك الفرصة لتنشر هذا الغسيل في بيتك الخاص.

لا تستهن أبداً باختبار النظام البرمجي (الجاهز نظرياً). رسمياً، عملية الاختبار يتم التخطيط لها من قبل خبراء، وتتم بفريق مبرمجين غير الذي قام بالبرمجة الفعلية، ويتم اختيار نوع البيانات (ونوع الأخطاء) التي ستختبر النظام بعناية فائقة. العبرة الختامية هنا: بالطبع برنامجك يحوي الكثير من الأخطاء والثغرات؛ حاول اكتشاف أكبر قدر منها قبل أن يكتشفها المستخدم.

أخيراً، مرحلة التسليم تعني تركيب النظام في بيئة المستخدم النهائية، وربما التدريب على النظام. تقترن هذه المرحلة دائماً بطور صيانة النظام، وهو أطول طور في دورة حياة النظام البرمجي على الإطلاق (ما لم يفشل النظام من البداية بالطبع). صيانة النظام تشمل ثلاث مهام أساسية:

1. تصحيح الأخطاء.
2. تعديل بعض الأجزاء.
3. إضافة أجزاء جديدة.

المهمة الأولى أنت ملزم بها كمنتج للنظام، على الأقل خلال فترة أولية يتم الاتفاق عليها، ثم من الممكن فرض بعض الأتعاب بعد انقضاء هذه الفترة. المهمتان الأخيرتان تنبعان أساساً من تطور أو اختلاف متطلبات المستخدم انعكاساً لعمله، ومن الممكن تقديمهما مقابل ثمن إضافي. أحياناً، يوفر منتج النظام تعديلات دورية من باب تحسين وتطوير نظامه بغض النظر عن متطلبات المستخدم. وعلى كل حال، فإن هذه الصيانة قد تستمر مادام النظام مستخدماً.

قبل أن أنهي الكلام في هذه الحلقة، أريد أن أذكرك بأن السؤال ما زال قائماً: هل أنت.....؟

رقم الحلقة (9) مقدمة إلى مخطط الكائنات والعلاقات ERD

ذكرنا مراراً أن تطبيقات قواعد البيانات تتكون من شقين: التطبيق، وقاعدة البيانات. فلنتناس الآن التطبيق، ونهتم بقاعدة البيانات...

اسمح لي أن أقدّم بعض التوضيح لما نحن بصدده، وأرجو أن تتذكر هذا دائماً: قبل أن تخطو الخطوة ينبغي أن تدرك جيداً ما هو محلها من الإعراب، من أين تبدأ، وإلى أين تنتهي. هذا الوضوح في الأفكار هو ما يميز ما نسميه بالمنهجية... لكي تبني قاعدة البيانات، ينبغي أن تمر على مرحلتين، الأولى هي التحليل والثانية هي التصميم. لا تخلط بينهما. التحليل كما أسلفنا يعني جمع كل ما يمكن من فهم المتطلبات. هنا تشترك قاعدة البيانات مع التطبيق في الحاجة إلى التحليل، وفي مفهوم التحليل، وفي آليات التحليل (طرقه ووسائله لمن كان يكره المصطلحات السمجة). بعد أن تفهم النظام جيداً، تأتي مرحلة التصميم. بالنسبة لقواعد البيانات العلائقية، الهدف النهائي من التصميم هو تحويل الفهم الذي حصلت عليه في مرحلة التحليل إلى مجموعة من الجداول (السوية). ما معنى (سوية)، وكيف تصل إلى جداول (سوية)؟ هذا موضوع آخر في منتهى الأهمية، بل هو في قلب تصميم قواعد البيانات، لذا أنصحك أن تكون موجوداً عندما نتحدث عنه فيما بعد بإذن الله. في سبيل الوصول إلى هذا الهدف، هناك مجموعة من الأدوات التي قد تساعدك، من أبرزها وأكثرها انتشاراً مخطط الكائنات والعلاقات ERD (Entity Relationship Diagram).

عندما تصمم الجداول فأنت تصمم البنية المنطقية للبيانات في النظام، أنت تقول بعد التحليل: لقد فهمت المطلوب، وقد وجدت من هذا الفهم أنه لكي يعمل النظام بالشكل المرجو، فإنه يحتاج إلى كيت وكيت من البيانات، التي يمكن تقسيمها وتنظيمها بالشكل الفلاني، وهذا الشكل الفلاني هو تصميم قاعدة البيانات. هناك طريقان يتبعهما المصممون عادة. إما أن تبدأ مباشرة بتخطيط مجموعة من الجداول التي تمثل تصميم قاعدة البيانات، لكن يلزمك بعد هذا أن تقضي بقية الوقت في تطبيق قواعد التسوية على مجموعة الجداول هذه لتتأكد من أنها (سليمة). مرة أخرى، المقصود بالضبط من (سليمة) هاهنا سيكون بإذن الله واضحاً عندما نتحدث بتفصيل أكبر عن تسوية قواعد البيانات. الطريق الثاني هو أن تقضي الوقت في البداية تبني مخطط الكائنات و العلاقات ، وهذا ليس بالأمر السهل، لكنه الأكثر منهجية. إذا كانت خبرتك تمكنك من بناء مخطط سليم يعكس الواقع، فإنك تقريباً قد انتهيت؛ الباقي هو مجرد تطبيق قواعد آلية لتحويل المخطط إلى مجموعة من الجداول التي من المفترض أن تكون سوية بالفعل.

من هنا نفهم أن مخطط الكائنات والعلاقات أداة مهمة في تصميم قواعد البيانات، والتمكن منها يتطلب بعض الخبرة، والخبرة تعني الكثير من العمل. المبدأ وراء هذا المخطط واضح وسهل. أنت تميز في النظام الذي أمامك (مخزن، صيدلية، مدرسة، مستشفى، شركة...) مجموعة من الكائنات التي يمثل كل منها وحدة واحدة، قد تكون هذه الوحدة شخصاً أو شيئاً أو مفهوماً ما. مثلاً، بعض الكائنات التي قد تميزها في الأنظمة السابقة: طالب، مريض، موظف (أشخاص)، مادة أو صنف، فصل، عيادة (أشياء)، علامة، إجازة (مفهوم)، وهكذا... الصعوبة هاهنا هي في اكتشاف وتمييز الكائنات. الخطوة التالية هي اكتشاف وتحديد العلاقات بين هذه الكائنات (لا بد أن تكون هناك علاقات بينها لأنها تنتمي لنفس النظام). لاحظ أن كلاً من الكائنات والعلاقات قد يملك خصائصه الخاصة التي تصفه بدقة (الدقة هنا تعتمد على متطلبات النظام). هنا أيضاً صعوبة في اكتشاف العلاقات ومن ثم في تحديد نوعها. نعم العلاقات أنواع كما لا بد قد سمعت من قبل. بل إن هناك أنواع لأنواع العلاقات. المزيد فيما بعد بإذن الله.

على الرغم من أن مخطط الكائنات والعلاقات موضوع يدرس في علم قواعد البيانات، وله طرقه وقوانينه، إلا أنه كما أشرت من قبل يحتاج إلى بعض الخبرة للتمكن منه. الصعوبات التي قد تواجه المبتدئ تشمل، كما سبق أن أشرت، الاكتشاف الصحيح للكائنات والعلاقات والتفريق السليم بين الكائن وخاصية الكائن (هل الهاتف كائن بحد ذاته أم أنه خاصية من خصائص الموظف؟)، وكذلك الاختيار بين الكائنات والعلاقات (الفاتورة كائن أم علاقة بين كائنين؟). وبالرغم من هذه الصعوبة، إلا أن العائد كبير، وهو تصميم سليم للجداول، وتعميق لفهم النظام، وخبرة لا تقدر بثمن في تصميم قواعد البيانات.

هذا يحدد الطريق أمامنا بعض الشيء فيما يتعلق بقواعد البيانات. علينا أن نحلل النظام لنفهمه جيداً، وهذا يستلزم جمع قدر كاف من البيانات بمختلف أشكالها. ثم علينا أن نستخدم هذا الفهم في إنشاء مخطط للكائنات والعلاقات (الطريق الأكثر منهجية)، ثم يلزمنا أن نحول هذا المخطط إلى مجموعة من الجداول. هذا يلخص تحليل وتصميم قواعد البيانات. طبعاً من الممكن اختبار التصميم بتطبيق قواعد التسوية، وبعد ذلك يأتي دور إكمال التصميم بقواعد لضمان تكامل وسلامة القاعدة وبياناتها، وغيرها من المكملات التي تختلف من نظام إلى آخر. سأنهي الآن هذه الحلقة بإذن الله، لكن من الممكن أن تقضي الوقت حتى الحلقة القادمة في الإجابة عن السؤال الذي أصبح مألوفاً جداً: هل أنت.....؟

رقم الحلقة (10) التحليل من ناحية عملية: المقابلات

لنتحدث الآن عن التحليل من جانب عملي أكثر... لدينا الكثير لنتذاكره عن تصميم قاعدة البيانات، لكنني أتبع هاهنا أسلوب (اترك الحلو للنهاية)، وأنا أفترض أن موضوع التصميم هنا (حلو)...

يقولون إن فهم السؤال هو نصف الإجابة، والحقيقة أنه إن لم يكن هناك فهم للسؤال، فليس هناك حتى (نصف) إجابة، وهذا يعني أن فهم المشكلة جزء لا يتجزأ من الحل. من هنا تدرك أهمية التحليل. التحليل كما قلنا مراراً هو الفهم للنظام الذي تواجهه في الحياة العملية، وموضوعنا هاهنا هو كيف تحصل على هذا الفهم...

سنفترض هنا إن شاء الله أننا بصدد تصميم قاعدة بيانات من أجل نظام مبيعات مصغر لمحل يبيع مواداً... هذه المواد قد تكون قطع غيار، أو أدوات صحية، أو أدوية (صيدلية)، أو مواد غذائية... الفرق في حالة المثالين الأخيرين أن التصميم يزداد صعوبة بسبب وجود مبدأ مدة الصلاحية.

على مستوى برامجنا الصغيرة نسبياً، فإن أهم وسيلة لجمع المعلومات في مرحلة التحليل هي السؤال المباشر. والسؤال المباشر يكون غالباً (وهذا أفضل) عبر المقابلة الشخصية. نعم، هناك من يستخدم الهاتف لجمع المعلومات، بل إن الأمر يصل إلى استخدام الواسطة... ربما قد سمعت عن صديق لك يعد برنامجاً لصاحب محل، والصلة الوحيدة بينهما هي صديق يعمل في المحل، وينقل إليه المطلوب... كل شيء ممكن هذه الأيام...

المقابلة قد تكون رسمية أو غير رسمية. وقد تكون على مستوى واحد أو عدة مستويات. قد تقابل مدير المستوصف في مكتبه، وقد تقابل أحد مساعديه في المقهى المجاور. إذا كان مالك العمل ليس هو من يعمل على النظام، وهذا هو الحال غالباً إلا في بعض المحلات أو الصيدليات الصغيرة، فإن مقابلة المالك وحده (من يملك العمل، ويفرض قواعد اللعبة، ويدفع النقود) لا تكفي بالتأكيد. هو يملك العمل لكنه ليس من يقوم بالعمل، وهنا فرق كبير؛ هو يفرض القوانين لكن لا أحد يلتزم بها، وهو يدفع النقود لكنه لن يدفع في النهاية إذا لم يعجب النظام المستخدم العادي. هاهنا نقطة حيوية شديدة الأهمية: مدخل البيانات العادي أو المستخدم النهائي العادي للنظام، هذا الذي تزدريه وتنظر إليه شزراً، وتتساءل من أين أتى بكل هذا الغباء والبلادة، هذا هو بالتحديد من سيحكم على نظامك بالكامل إما بالنجاح، وإما بالفشل. برامج كثيرة ممتازة حكم عليها بالفشل لأن المستخدم الأخير قال بأنها لا تنفع، وإنها لا تؤدي المطلوب. برامج أخرى أقل مستوى وجدت لها مكاناً لأنها عرفت كيف تتعامل مع متطلبات المستخدم الأخير.

إذاً، عليك أن تقابل صاحب العمل، والمستخدم لنظامك. صاحب العمل يعطيك المطلوب النهائي والخطوط العامة، لكنه لا يعرف دائماً (فضلاً عن أن يعطيك) تلك التفاصيل الصغيرة التي تشكل في آخر الأمر الحياة اليومية للنظام. بالمناسبة، في المصادر الإنجليزية يذكرون مصطلحين هما (client) و(user)، والفرق بينهما هو ما نتكلم عنه الآن من الفرق بين المالك (العميل الذي يشتري منك النظام) والمستخدم الذي يعمل على النظام (مدخل بيانات أو مشغل). أحياناً سيحيلك المالك مباشرة إلى المستخدم الأخير (أو المستخدمين)، ويطلب منك مراجعته كلياً. هذا يوفر عليك الكثير، وإن كان خطيراً إن لم يكن هذا المستخدم معجباً بك بالقدر الكافي.

المقابلة ليست نزهة من أجل شرب كوب من الشاي أو الشعور ببعض الأهمية. المحلل الذي يقابل الناس من أجل جمع المعلومات وفهم النظام يجب أن يتمتع بعدة صفات (ليس من بينها التمتع بكوب الشاي إياه). أولاً، ينبغي أن تعرف ما تريد. ثانياً، ينبغي أن تستخرج ما تريد بذكاء. الذكاء يشمل إلى جانب الأسئلة والتعليقات الذكية، صفتين مهمتين جداً. الأولى حسن الاستماع، والثانية حسن الكلام. حسن الكلام يسمى اللباقة، وهي مهارة أساسية من مهارات التواصل مع الناس، خصوصاً عندما تحتاج إلى هؤلاء الناس من أجل عملك. حسن الاستماع يستحق أن نؤكد بشدة عليه، لأن كثيراً من الناس لا يملكون الصبر الكافي. عندما تصطاد السمك، مهما كنت خبيراً، فإنك لا بد أن تقضي بعض الوقت في الانتظار من أجل أن تحصل على ما تريد. أنت هناك تصطاد المعلومات، ولا تتوقع أن ينهال عليك السمك من أول وهلة.

هنا أيضاً نقطة تستحق التأكيد. لا تسأل أسئلة عامة جداً. لا تسأل أحداً مثلاً أن يصف لك عمله. كل شخص (يشعر) بعمله، ولكنه من الصعب أن ينقل هذا (الشعور) إلى الآخرين دون أن ينسى شيئاً. إذا سألتك أنت ماذا تعمل لأجبتني بكل ثقة: أنا أطور الأنظمة. وإذا أخبرتك أنني لا أفهم في تطوير الأنظمة ولا في البرمجة لكنني أريد منك أن تشرح لي عملك بالضبط، فعلى الأرجح أنك ستركلني من الباب، أو إذا كان مزاجك رائقاً، فقد تخبرني كيف أنك تحلل الأنظمة وتصممها ثم تبرمج، وقد تستفيض في الشرح، لكن هل تعتقد أنه سوف يكون بإمكاني أخذ هذا الشرح و(أتمتة) عملك بناء عليه؟ هذا ما تحاول أنت أن تفعله مع بقية الناس، لذا كن على حذر.

أدعك لتسأل نفسك مرة أخرى: هل أنت.....؟

رقم الحلقة (11) التحديات في وجه المحلل

أود أن أفصل أكثر في هذه الحلقة عن التحديات التي ينبغي أن يتوقعها المحلل (أنت ولا فخر)، خصوصاً من غير المالك، وهذا يشمل المستخدمين النهائيين، وقد يشمل أحياناً مستفيدين غير مباشرين من النظام. ثم من الممكن أن نعرج قليلاً على الوسائل الأخرى (بخلاف المقابلة الشخصية) التي تستخدمها عادة لجمع المعلومات في مرحلة التحليل.

استكمالاً لموضوع المقابلات، أرغب أيضاً في التأكيد على نقطة أخرى... احرص على أن تدير الحديث لا أن تقوم أنت بالحديث. هذه النقطة ذات شقين؛ الأول قد سبقت الإشارة إليه من أنك ينبغي أن تتحلى بالصبر، وأن تستمع أكثر مما تتكلم (طبعاً نستثني الأخوات، لأن هذا المطلب بالنسبة لهن يشبه محاولة الانتحار... ملحوظة مهمة: أنا أمزح!) لاحظ أن استماعك لن يقتصر في الغالب على موضوع النظام، خصوصاً إذا كنت تقابل عمالاً غير مالك العمل. خصوصاً إذا كانوا من النساء. ستحصل على كمية لا بأس بها من القيل والقال والشكوى والتذمر، وربما بعض المشاكل الشخصية، توقع أيضاً التطرق إلى السياسة، من سياسة الشركة وموازين القوى في الهيكل الإداري الجديد إلى سياسة الدولة. تحلى بالصبر. من بين ثنايا كل هذا الهراء أنت تحصل على الكثير. أول ما تحصل عليه هو ثقة وود الموظفين. ثم ستحصل على معلومات قيمة ليست مدونة في اللوائح، ولا يتطرق المدراء إليها غالباً. مثال؟... حسناً، أنت قد تعرف ما الذي يبغضه الموظفون في النظام السابق، لكن المدير قد طلب منك إضافته في النظام الجديد أيضاً. أنت قد تعرف بعض القيود التي يصر المدراء على إدراجها في الأنظمة، لكن أحداً لا يلتزم بها وتشكل في النهاية نقطة ميتة في النظام، لا تعمل، ومميتة، تعرقل عمل النظام ككل. أنت بالطبع تحصل في أثناء هذه الجلسات على صورة أقرب لبيئة العمل، وهذا مصدر ثمين بالنسبة لك كمحلل.

لكن، كل عملة لها وجهان. الشق الثاني من النقطة التي قدمتها في الفقرة السابقة هي أن (تدير) الحديث أيضاً. لا ينبغي أن تترك الحبل على الغارب. المسألة تحتاج إلى فن. ينبغي أن تقود الحديث من بعيد بحيث توجهه من حين لآخر ليصب في الهدف الذي تركت بيتك من أجله في الصباح. إذا لم تكن تتقن هذا الفن فإن ترمس الشاي سينتهي ولم تحصل على شيء ذا قيمة (بخلاف أن سعاد دائماً تحصل على الإكرامية لأنها أنشأت علاقة مع أسرة المدير الإداري، وأن المدير الحالي هو مجرد حمار لا يفقه شيئاً). هذا يعني أن تعد مسبقاً مجموعة من الأسئلة المختارة بعناية، بحيث تحصل في نهاية اليوم على إجاباتها (ولو طال اليوم وأردف أعجازاً وناء بكلكل... هل يعرف أحدكم ما هو الكلكل؟).

البعض يحب الفوضى، لأن الفوضى تخفي المخالفات. يقولون إن المال السائب يعلّم السرقة. محل أو شركة أو صيدلية بدون نظام هي –في الحصيلة- مال سائب. هذا صحيح للأسف. وهذا البعض توقع أن تقابله على عدة مستويات. شريك صاحب المحل قد لا يرغب في إدخال الحاسوب (ومشاكله) إلى العمل. أحد المدراء في الشركة قد يبذل ما في وسعه ليجعل مهمتك مستحيلة بحجة أن خبرتك مثلاً لا تكفي لبناء نظام متكامل. مدخل البيانات العادي قد يحاول بشتى الطرق أن يوحي لك بأن ما تحاول القيام به لا يمكن تطبيقه.

مدخل البيانات؟... مدخل البيانات مرة أخرى؟ وماذا قد يستفيد هذا من الفوضى؟ حتى أنت يا بروتس؟ نعم، (بروتس) هذا (أنا أكرر) ينبغي ألا تستهين به أبداً. أنا أضرب لك مثالين، مثال يبين كيف يمكن أن يستفيد المدخل العادي من ركاكة أو عدم دقة النظام، ومثال آخر أذكره من أيام الجامعة من دكتور قديم عمل في الماضي السحيق في تصميم بعض أنظمة قواعد البيانات. المثال الأخير كان لنظام ما يسرد حقوقاً ما (لا أذكرها الآن جيداً) لمجموعة من الأشخاص يتم ترتيب أولوية حصولهم على هذه الحقوق من قبل البرنامج حسب الترتيب الأبجدي لأسمائهم. لا أحد يهتم بآلية الترتيب فيما بعد، بل المهم التقرير الذي في اليد والأسماء المطبوعة. طبعاً يتم في البداية إدخال بيانات الأشخاص من قبل مدخلي البيانات. ماذا يستطيع عمله مدخل البيانات العادي الذي يصبح أحياناً غير عادي بالفعل؟ الشخص الذي يمت لمدخل البيانات بصلة قرابة أو معرفة يقوم بإدراج فراغ صغير (مسافة space) قبل اسمه الأول، وبهذا يضمن أن اسمه سيأتي في بداية القائمة ولو كان (يوسف) مثلاً...

المثال الذي يوضح كيف يمكن أن يستفيد مدخل البيانات من عدم صرامة النظام، يحدث في برامج الفوترة. في نظام لفواتير الكهرباء على سبيل المثال، إذا لم تكن هناك آلية قوية لاكتشاف أخطاء الإدخال، فإن بعض المدخلين يعمد إلى تغيير القراءة الشهرية لمنزله أو منازل الأصدقاء بعد انتهاء طور الإدخال، وقبل بدء الاحتساب الأخير للفواتير...

بقي أن أنبهك إلى واحد من أكبر التحديات التي يواجهها أي نظام برمجي جديد، وبالتالي تواجهها كمحلل. هذا الموضوع لا يمكن إغفاله، لأنه شائع جداً ومتوقع جداً. هو خاص كذلك بالأنظمة التي لا يكون فيها المالك هو مستخدم النظام، بل يكون هناك طاقم آخر لاستخدام النظام. هذا التحدي بكل بساطة هو عداء الطاقم. هذا العداء قد يكون سافراً ويجعل مهمتك مستحيلة، وقد يكون مستتراً ويجعل مهمتك في غاية الصعوبة. طبعاً أن يكون العداء مستتراً ليس سببه الخوف من شخصك الكريم، لكن على الأرجح من غضبة المدير. لماذا قد يعادي أي طاقم النظام الجديد؟ أليس من المفترض أن هذا النظام سيجعل حياتهم أسهل، وينقلهم من عصور القرون الوسطى حين تم اختراع الطباعة إلى عالم (النقرة والسحب والإفلات)؟ لا أدري لماذا خطر ببالي استخدام الفأرة ولوحة المفاتيح من بين كل تكنولوجيا الحاسوب...

أصحاب الخبرة يعلمون أن هذا الكلام مغرق في النظرية. سأذكر هاهنا سببين رئيسين من بين عدة أسباب تقود إلى عداء المستخدمين لك. الأول أن النظام الجديد يعني في كثير من الأحيان أن هناك من سيفقد شيئاً ما مهماً. هذا الشيء يتراوح من الوظيفة برمتها إلى المركز أو الأهمية. يمكن لأي نظام تمت (أتمتته) أن يؤدي إلى الاستغناء عن بعض الطاقم، أو تغيير مراكزهم. ينبغي أن تضع هذه الحقيقة نصب عينيك دائماً، وأن تخفيها قدر الإمكان عن الطاقم. هناك حل آخر، لكنه يقود إلى السبب الثاني من أسباب العداء بين طاقم العمل وبين المحلل. ذلك أن يتم استخدام الطاقم نفسه من أجل الإدخالات وتشغيل النظام دون الاستغناء عنهم، لكن هذا يقتضي أن يتعلم الطاقم استخدام الحاسوب، أحياناً من الصفر، ناهيك عن استخدام النظام. بعض الموظفين (خصوصاً القدامى وكبار السن)، يرادف هذا بالنسبة له كلمة مستحيل. الكثير من المواقف قد تواجههابسبب هذا، لكن لن أطيل هنا، فالفكرة قد وصلت على الأرجح. العبرة هاهنا: انت مكروه بالغريزة، فلا تجعل من نفسك مكروهاً أكثر من ذلك...

كنت أود التطرق في هذه الحلقة للمزيد من وسائل التحليل، لكن الكلام قد أخذنا (أرجو ألا يكون التطويل باعثاً على الملل). على كل حال أختم بتلميح إلى أن المحلل يستطيع دائماً بإذن الله أن يستخدم بعض الذكاء و(الدبلوماسية) واللباقة من أجل تسهيل مهمته وتخفيف التحديات التي تواجهه بطبيعة عمله. مثلاً، تأمل في محلل يأتي إلى موظف من عند المدير ليقول له راسماً على وجهه أمارات الأهمية: "السلام عليكم... أنت رياض؟ المدير يكلفك بأن تصف لي تفاصيل إدخال القيود، ولكن اختصر رجاء، لا داعي لإضاعة وقتي ووقتك، عندي خلفية كافية عن الموضوع."... الخلفية هذه يا صديقي من الممكن استخدامها في سطح المكتب، لكن ليس عند التحليل. ثم تأمل في أسلوب محلل يدخل المكتب سائلاً عن (الأستاذ) رياض، ثم يقدم عليه راسماً على وجهه ابتسامة ودودة، ويقول: "السلام عليكم، الأستاذ رياض؟ أرجو أن تسمح لي ببعض الوقت، لأنني علمت من المدير أنك أنسب شخص ممكن أن يعطي فكرة عن إدخالات القيود... وأنا بالفعل متلهف لمعرفة هذه المهمة التي يقولون أنها ليست بالأمر اليسير...". لو كنت (الأستاذ) رياض، من من الاثنين ستقدم له كوباً من الشاي من الترمس الخاص الذي أحضرته من بيتك؟

لا تنس في ختام الحلقة أن تسأل نفسك السؤال المعتاد: هل أنت.....؟

رقم الحلقة (12) وسائل التحليل الأخرى

إذا كنت ما تزال في مزاج لمواصلة القراءة، فدعنا نمر سريعاً على بعض الوسائل الأخرى التي تستطيع من خلالها جمع المعلومات في مرحلة التحليل في سبيل الوصول إلى فهم شامل للنظام الذي بين يديك. سؤال المستخدم قد لا يكون مباشرة عبر المقابلة الشخصية أو المكالمة الهاتفية. قد يكون، نظرياً على الأقل، عبر استبيانات مكتوبة توزع على المستخدمين المعنيين من أجل تعبئتها. هذه من أقل الأساليب استخداماً في حياتنا العملية، وأذكرها فقط من باب أننا كنا نرسم المزهرية دائماً حين نرسم طاولة ونحن أطفال. لماذا المزهرية؟ هل توجد مزهرية على الطاولة؟ لا، ولكن الطاولة تأتي في اللوحات الشهيرة مع مزهرية.

الوسيلة المهمة جداً التي لا تستطيع أن تعمل بدونها، هي كل التقارير والنماذج والمستندات المطبوعة المتداولة في النظام. التقارير هي مخرجات النظام. النماذج والمستندات تمثل مدخلات النظام. النماذج هي عادة حقول تملأ بالبيانات، في حين أن المستندات هي عبارة عن وثائق مثل سندات مالية وخلافه. التقارير هنا تلعب دوراً مميزاً في غاية الأهمية. الطريقة الشهيرة في التحليل هي البدء من المخرجات (output) لاستنتاج المدخلات (input) وفهم النظام. حتى عند السؤال المباشر، أنت عادة تسأل عن (المطلوب) ثم تسأل كيف يمكن الحصول على هذا المطلوب. هذا يشمل حالة تصميم نظام لأتمتة نظام يدوي أو بدلاً عن نظام حاسوبي سابق. ادرس المخرجات بعناية، لأنك يجب أن توفر هذه المخرجات في النهاية. الخلاصة هنا: اجمع كل ما يمكنك من أوراق.

يتعلق بالنقطة السابقة نقطة أخرى. بخلاف الوثائق المستخدمة في حركة النظام، تجد بعض الأحيان مستندات توثيقية لنشاط النظام، خصوصاً في الأنظمة الرسمية الكبيرة نوعاً ما. مثال ذلك اللوائح التي تنص على سياسة الشركة أو النظام الإداري أو المالي. أحياناً تجد توثيقاً لنشاط الشركة أو الجهة الطالبة للنظام في شكل مجلدات أو حتى مطويات تستخدم في الدعاية. ألق نظرة هنا وهناك، إذا وجدت وقتاً في مرحلة التحليل للقراءة، فالأفضل أن تقرأ هذه المطبوعات بدلاً عن قراءة الصحف.

نقطة أخرى أرى أنه كان ينبغي أن أقدمها لما تحتله من أهمية. هذه الطريقة مفيدة جداً، وأحياناً تكون هي الوسيلة الوحيدة العملية لاستكمال التحليل. تلك هي المراقبة. ينبغي أن تذهب بنفسك إلى بيئة العمل بعد المقابلات، من أجل مراقبة العمل وهو يمضي. تراقب عن كثب (في صمت، لا أحد يحتاج نصائحك القيمة هنا)، وترصد ما لم يتم إطلاعك عليه، بسبب النسيان أو بسبب (التناسي). هذا لا يكون متوفراً دائماً، خصوصاً في الأماكن الحساسة، لكن في غالبية الأنظمة التي نقوم بتصميمها من أجل محلات صغيرة أو مرافق عامة، فإنه يمكنك التنسيق للمداومة مع الموظفين لفترة محددة. يقولون إن ما يتم تعلمه بالرؤية أبلغ مما يتم بالقراءة.

مصدر جيد آخر من مصادر الفهم في التحليل، الاطلاع على الأنظمة الجاهزة الأخرى. أحياناً بصفتك متخصص حاسوب لاتفهم لغة التخصصات الأخرى، لكنك تفهم لغة برنامج أو نظام حاسوبي آخر.

بالمناسبة، في بعض الأنظمة القياسية، أعني التي لها أصول متفق عليها وتشكل أحياناً علماً قائماً بحد ذاته، فإنه يلزمك أن تتعلم على الأقل أساسيات أصول هذه الأنظمة. أعلم أنك ستقفز الآن وتقول: أنا أعلم ماذا تعني. نعم، واحد من أشهر الأمثلة على ذلك أنظمة المحاسبة القائمة على أصول المحاسبة المنهجية المتعارف عليها. من الصعب أن تفهم هذه الأنظمة إلا بعد أن تلم بمبادئ علم الحاسبة. من أجل ذلك أنا أرى الآن أن أكبر خطأ ارتكبوه في حقنا أيام الدراسة القديمة أنهم لم يدرجوا علم المحاسبة في منهج الحاسوب.

أختم بآخر وأهم مصدر من مصادر فهم متطلبات أي نظام. أهم مصدر؟ في الأخير؟ نعم... وهو العقل! استخدم عقلك! أنا لا أعني أي إهانة من أي نوع. فقط أعني أنك يجب ألا تقوم بكل عملية التحليل بآلية. الآلات غبية. فكر جيداً فيما هو غث وما هو سمين، فيما هو خطأ وما هو صواب، فيما هو صدق وما هو كذب (نعم، حتى في تحليل أنظمة قواعد البيانات). من أهم الأوقات التي ستحتاج فيها إلى التفكير هي عند استكمال الفراغات. ستجد غالباً أجزاء لم يخبرك بها أحد، وغالباً لا يستطيع أن يخبرك أحداً. لا تستغرب أن المستخدم نفسه أحياناً لا يعرف ما هو المطلوب بالضبط في نقطة معينة، خصوصاً عندما يتعلق الأمر بمعالجة المطلوب باستخدام الحاسوب. هنا يمكنك أن تكمل الفراغات. والعامل الرئيس هنا هو الخبرة. كلما زادت خبرتك في تحليل الأنظمة أصبحت قراراتك صائبة أكثر.

هل نسيت شيئاً؟ على الأرجح. لكنك هنا معي أيها القارئ الكريم من أجل ذلك، ذكرني إن نسيت. وعلى كل حال هذا ما يحضرني في باب التحليل، وإلى لقاء قادم أسألك فقط من باب الفضول: هل أنت.....؟

رقم الحلقة (13) مدخل إلى التصميم: نظرة عامة

وصلنا إلى مرحلة التصميم. وكما ذكرنا، فإن مهمتنا هنا أن نصمم قاعدة البيانات خاصة. لسوء حظك، هنالك العديد من المفاهيم التي لا بد من عرضها، لذا لا بأس من أن تقوم الآن وتعد لنفسك كوباً من الشاي، فإن الجلسة قد تطول. أنا أفضله بمزيد من السكر طبعاً، ولكنني سأكون مشغولاً بالكتابة على الأرجح، شكراً.

أفضل وسيلة لفهم موضوع هو أن تقترب منه بما يسمى (منظور الطائر). هذا يعني أن لا تقفز في الموضوع مباشرة وتخبط بيديك محاولاً النجاة. إذا لم تكن تجيد السباحة جيداً فقد تغرق. منظور الطائر (bird view) يعني أن تقترب من الموضوع من بعيد ومن أعلى، كما يقترب الطائر، وتلقي نظرة شاملة على الموضوع من الخارج (قبل أن تدخل فيه)، مما يمنحك فرصة اكتشاف أبعاد الموضوع الحقيقية، حجمه، علاقته بغيره من المواضيع، من أين جاء، وإلى أين يقود. هذا المفهوم مهم جداً، ولا أعرضه من باب الفلسفة، ولكن من باب نصيحة عملية لمواجهة أي موضوع تود فهمه. بغير هذه الطريقة ستظل تشعر بنقص في الفهم وبعض الضياع حتى بعد مرور سنوات من تعاملك مع الموضوع، إلا أن يفتح الله عليك فتحاً من عنده.

عندما كنت طالباً، فإنك كنت داخل قوقعة، من النادر أن تفهم (حقاً) المادة المعروضة. المواد تلقى عليك وليس العكس، بمعنى أن المادة هي التي تفرض عليك وتجد نفسك في خضمها ولست أنت من يقترب من المادة. أنت تشعر في تلك الحال أن الدنيا كلها هي هذه المجموعة من الكتب (أو الملازم كما هو الحال عندنا) التي يجب أن تدرسها (من أجل الامتحان بطبيعة الحال)، وقد تشعر أحياناً بسبب الوقت والجهد الرهيب الذي بذلته فيها أنك ملم بالمادة جيداً. بعد الخروج من القوقعة، وغالباً ما يكون هذا بعد التخرج كلية، تبدأ في فهم ماهية الأمور، وبعد أن تكتمل عندك عدة خيوط بعضها يقود إلى بعض تدرك أنك لم تكن تدرك شيئاً. تبدأ عندها في امتلاك القدرة ليس على فهم أعمق للمواد فحسب، بل وعلى الانتقاد في اختيار بعض المواد وعرض بعضها الآخر بالشكل الذي تم. تبدأ عندها في الشعور بأنك قد (خُدعت) بشكل أو بآخر...

لماذا هذه الفلسفة؟ لأنك قد تكون الآن في قوقعة أخرى... بدأت في التعرف على قواعد البيانات من اليوم الذي عرفت فيه الأكسس، أعني أنك دخلت في الموضوع من الوسط ولم تقترب منه كما سيفعل أي طائر ينوي أن يبرمج لقواعد البيانات... هذا يحرمك من التقدير الصحيح لكثير من المبادئ التي قد تتبعها ولكنك لا تعلم حقاً لماذا، أو التي لا تتبعها لأنك لم تسمع بها أصلاً، وهذه مشكلة... تقول كفى ثرثرة؟ معك حق، لكن يجدر بك حقاً أن تحضر لنفسك الآن كوب الشاي...

عوداً على نظرة الطائر، بدلاً من أن أجعلك تحلق عالياً (وهذا خطير... تذكر عباس بن فرناس؟)، لنبق على الأرض لكن نفترض أنك قد ولدت قبل أربعين عاماً، وأنك تعمل مبرمجاً في شركة كبيرة. هذه الشركة لديها قاعدة عريضة من العملاء، وبالطبع فإن كمية ضخمة من بيانات هؤلاء العملاء وتعاملاتهم يجب أن يتم حفظها. في ذلك الوقت، تسأل عن الأكسس، فيهمس لك أحدهم في أذنك بأن بيل جيتس نفسه لم ير بعد أول حاسوب في حياته وبالتالي ليس هناك أكسس ولا ويندوز حتى، وأنه يتساءل حقاً من أين جئت بالضبط كي لا تعلم ذلك! أوراكل؟ إنفورمكس؟ سايبيس؟ طيب، فوكسبرو؟ أي شيء؟ لاشيء...! في النهاية تتنهد في أسى وأنت تتذكر أيام زمان (الآن)، وتدرك أنه ينبغي لك أن تستخدم نظام الملفات التي يدعمها نظام التشغيل لتحفظ البيانات. ستختار تنسيقاً معيناً خاصاً بك للبيانات، ثم تكتب مجموعة من البرامج (ربما بلغة كوبول) لتنفذ العمليات الأساسية على البيانات في الملفات، مثل إضافة زبون أو عملية شراء، أو استخراج مجموع إيراد يوم، وهكذا حسب طلب الإدارة.

أنت الآن تفكر: لا بأس إطلاقاً، ما دامت هناك وسيلة لتنفيذ المطلوب، فإن المرء يستطيع العيش بدون أكسس. هذا صحيح تماماً، وهذا بالضبط ما كنت أقوله قبل أن أشتري الجوال الذي أصبح الآن قطعة أخرى من ملابسي. دعنا نمض أكثر واسمح لنا أن نختلس إليك النظر وأنت تعد ملفاً لبيانات مبيعات العملاء، ثم واحداً آخر لديونهم، ولكل منهما برنامج خاص يضيف بيانات العملاء ومعاملاتهم إلى الملف المناسب. أنت تعلم بالطبع أن بيانات العملاء ستكرر في الملفين، ليست هناك طريقة أخرى حالياً، لكنك قلق أكثر على الحجم الزائد الذي يسببه هذا التكرار بدون فائدة. نحن جميعاً نعلم أن هذا العبء يشكل عيباً حقيقياً إذا أخذنا بعين الاعتبار قيمة مساحة الخزن في تلك الأيام، لكن الذي ينبغي ان تقلق منه أكثر هو الأخطاء التي قد تنجم عن هذا التكرار الزائد. يسمون هذا النوع من الأخطاء inconsistency، وهو في هذه الحالة يعني أن نفس المعلومة قد يكون لها قيمتان مختلفتان في مكانين مختلفين. نحن لاحظنا ذلك لأننا نراقبك ونحن نرشف الشاي، لكنك لم تلاحظ أن العميل محمد سلمان اسمه محمد سليمان في الملف الثاني. بغض النظر عن سبب هذا الاختلاف (مثلاً، تم تعديل الاسم في ملف ونسيان تعديله في الملف الآخر)، أظن أنه لا العميل ولا المدير (وبالتالي ولا أنت أيضاً) سيكون سعيداً.

عيوب مثل التكرار redundancy والاختلاف inconsistency ليست هي الوحيدة التي تواجهها. بالأمس طلب المدير قائمة بالعملاء المتميزين، ووضع شروطاً غريبة لهذا التميز، تتضمن كميات الشراء ونوعياتها وطريقة الدفع. لكن أحداً لم يقل من قبل أن برنامجاً يقرأ هذه المعلومات ينبغي أن يكتب. على المدير أن يقبل بالبرامج الجاهزة التي تم إعدادها من قبل، أو بالطبع فإنه من الممكن جداً أن المدير يعلن بكل وضوح أنهم يدفعون لك مرتباً شهرياً من أجل هذا. أعلم أنك كنت قد غادرت المكتب بعد انتهاء الدوام، لكن لسوء حظك، كان قد تم اختراع الهاتف قبل هذا الوقت بتسعين عاماً، لأن جراهام بيل ليس هو بيل جيتس الذي لم ير أول حاسوب في حياته بعد.

التنسيقات المختلفة للملفات حسب المبرمجين، توزيع البيانات في ملفات مختلفة، المعرفة الدقيقة لتنسيق البيانات في الملفات، كل هذا يجعل حياتك جحيماً، لكنه ليس كل شيء. هناك أنواع أخرى من الـ inconsistency هذه التي تنشأ من الوصول المتزامن لأكثر من مستخدم لنفس الملف. مثلاً، تمت قراءة مديونية العميل محمد سلمان من ملف المديونية من قبل برنامجين مختلفين (لنقل إن مديونيته تبلغ 100000)، في البرنامج الأول تمت إضافة عملية شراء بالآجل، مما زاد المديونية (5000)، وفي العملية الأخرى تم تسديد بعض المديونية (40000)، لكن كلا من البرنامجين قد قرأ القيمة القديمة للمديونية (100000)، والبرنامج الأول أضاف إليها قيمة الفاتورة (105000)، أما البرنامج الثاني فقد خصم منها قيمة السداد (60000). الحاصل أن واحداً منهما سيكتب نتيجته ثم يكتب البرنامج الآخر نتيجته. النتيجة الأخيرة ستمسح الأولى لكنها خطأ في كلتا الحالتين (النتيجة الصحيحة 65000)! الـ inconsistency هنا تعني اختلاف قيمة المعلومة في الملفات عن قيمتها الحقيقية في الحياة الواقعية. أخطاء مثل هذه قد تنشأ أيضاً بسبب عدم توفر الذرية (ترجمة غريبة لـ atomicity). هذا يعني أن انقطاع الكهرباء مثلاً قد يؤدي إلى تمام بعض العمليات وعدم تمام البعض الآخر، في حين أن هذه العمليات كلها يجب أن تتم معاً أو لا يتم شيء منها على الإطلاق. طبعاً الافتراض في هذه التسمية أن الذرة هي أصغر جزء غير قابل للتقسيم.

هل اكتفيت من تجربتك؟ أنت تقول أن هناك مشاكل أخرى تتعلق بالأمان وخلافه، لكن الشاي قد برد، ونحن لن نمضي بقية اليوم هاهنا! لو أنك كنت منتبهاً قليلاً في حصص الرياضيات المنطقية ونظرية المجموعات لربما استطعت أن تسبق Codd الذي يعمل في IBM، ويبدو أنه قد سئم هو الآخر من كل هذه المتاعب. قرر Codd هذا أن يبني نموذجاً جديداً لقواعد البيانات، يختلف جذرياً عن النموذج المستخدم المعتمد على ملفات تتم معالجتها ببرامج إجرائية (مثل الكوبول) تتطلب معرفة تنسيق خزن البيانات. نريد طريقة سهلة نحدد فيها (ماذا) نريد من بيانات وليس (كيف) نقرأها ونعالجها. (كيف) هذه تتطلب معرفة دقيقة بالتنسيق الفيزيائي للملف، ولغة البرمجة مثل الكوبول (تسمى لغات جيل ثالث) تترجم هذه الكيفية إلى أوامر لقراءة ومعالجة البيانات. (ماذا) لا تتطلب أكثر من معرفة البنية المنطقية للبيانات. في المطعم، أنت تطلب من المحاسب حصة من الأرز مع الدجاج، المحاسب يصيح (نفر رز مع الدجاج)، فيصل إليه، لكن الطباخ يجب أن يغرف الأرز بنفسه من الوعاء المناسب ويحضر الدجاج من الشواية مثلاً. المحاسب استخدم (ماذا) يريد، والطباخ استخدم (كيف). أنت طبعاً المستخدم الذي سيدفع!

تفاصيل النموذج الذي جاء به Codd أتحدث عنها إن شاء الله في الحلقة القادمة. حتى ذلك الحين ما زلت حقاً أتساءل: هل أنت.....؟

رقم الحلقة (14) قواعد البيانات العلائقية: المفاهيم الأساسية

قلنا إن Codd قد جاء بنموذج جديد لقواعد البيانات. نعرض في هذه الحلقة إن شاء الله أصول هذه الفكرة التي لا أشك في أن الأكثرين إن لم يكن الكل على دراية جيدة بها (أو أنني أشك؟). لاحظ رجاءً أن هذا عرض ميسر ومختصر للغاية، ولا يتطرق البتة للتفاصيل الرياضية (من الرياضيات وليس كرة القدم) التي قام عليها النموذج...

الفكرة تقوم أساساً على أن قاعدة البيانات تتكون من مجموعة من الجداول غير المرتبة. المقصود بغير المرتبة هاهنا أن الترتيب غير مهم. المقصود بالجداول هو بنية منطقية من مجموعة من الأعمدة غير المرتبة، ومجموعة من الصفوف غير المرتبة كذلك. تقاطع صف مع عمود يمثل معلومة محددة عن شيء ما أو كائن ما في الحياة الواقعية. وعلى هذا فمجموعة الأعمدة تمثل وصفاً لشيء أو وحدة واحدة، ومجموعة الصفوف تمثل عدداً من هذه الأشياء (التي يجب أن تكون من نفس النوع لأنها تشترك في نفس الوصف عبر نفس الأعمدة). إذاً، فالجدول هو مجموعة من البيانات المتعلق بعضها ببعض related، ولذلك كان اسمه بلغة Codd علاقة relation وليس جدولاً. المتقدم لوظيفة في شركة ما على سبيل المثال يمثل شيئاً أو كائناً أو وحدة ما، ومجموعة بياناته من اسم وتاريخ ميلاد ومحل ميلاد إلى آخره، تمثل خصائص علاقة relation attributes بلغة Codd وأعمدة جدول بلغتنا.
بالمناسبة، اسم قواعد البيانات العلائقية (ترجمة لـrelational databases) مأخوذ أصلاً من اسم الجدول relation بتعبير Codd، وليس من العلاقات التي تربط الجداول كما يظن الكثيرون.

هذا المبدأ الذي يبدو في منتهى السهولة والبداهة مبني على أساس رياضي متين كما أشرنا، لذلك يعد من ميزات هذا النموذج العلائقي أنه من أوائل النماذج الموصوفة بلغة رسمية، مما يعني وضوح ودقة مبادئها، على خلاف ما قد يخطر للبال من أنه نموذج بديهي يمثل البيانات في صورة مرتبة كما ترتب لك زوجتك الأشياء التي ينبغي شراؤها في قائمة. هذا يقود إلى الحقيقة التي تروعنا دائماً: لهذا النموذج أصول وشروط، ينبغي أن ندرك على الأقل المهم منها عملياً. هذا هو الهدف من هذه السلسلة لو كنت قد لاحظت.

دعنا نستعرض بعض الأفكار فيما قام به Codd. من المهم أن نلاحظ أنه قد تم تخليص المبرمج من الحاجة إلى المعرفة الدقيقة بتنسيق البيانات على المستوى الفيزيائي من أجل التعامل معها إجرائياً، أي بلغة برمجة من الجيل الثالث (مثل كوبول) تحدد أوامرها كيفية قراءة البيانات. أصبح الآن بالإمكان التفكير على المستوى المنطقي (راجع الحلقة رقم 10). أصبح الآن من الممكن التفكير في مجموعة من الجداول والعلاقات فيما بينها. العمليات التي تتم على هذه الجداول تشبه العمليات الي تتم على المجموعات في الرياضيات، مثل اتحاد جدول مع جدول أو تقاطع جدول مع جدول آخر. هناك أيضاً بعض العمليات التي تتم على جدول واحد ونتيجتها جزء من هذا الجدول مثل عملية الإسقاط وتعني اختيار مجموعة من أعمدة جدول، أو عملية اختيار مجموعة من صفوف الجدول. كل هذه العمليات منطقية وسهلة نسبياً ولا تتطلب أكثر من معرفة ماذا يوجد من بيانات وأيها تريد. لاحظ رجاءً أن هذه العمليات ينبغي أن يعبر عنها بطريقة ما، وقد عرفنا سابقاً أن هذه الطريقة هي لغة خاصة اسمها SQL.

ذكرت في الفقرة السابقة عرضاً أن هناك علاقات بين الجداول، هذا مفهوم قطعاً لأن كل جداول قاعدة بيانات واحدة تنتمي لنفس النظام، ولا يوجد نظام يتكون من كائن واحد حتى لو كان نظاماً لتقشير البطاطس. لاحظ أن هذه الفكرة تسمح بالتخلص من بعض العيوب عند استخدام نظام الملفات لحفظ قاعدة البيانات، كالحاجة مثلاً لتكرار البيانات وما يتبع ذلك من أخطاء. أصبح من الممكن الآن، إذا ذكرنا مثال الحلقة السابقة، أن نربط بين العملاء ومشترياتهم ومديونياتهم من دون الحاجة إلى تكرار بيانات العملاء. لا توجد هناك مشاكل اختلاف التنسيق والاضطرار إلى الاستعانة بمبرمج من وزنك الثقيل لكتابة برنامج جديد بلغة برمجة إجرائية كلما خطرت للمدير فكرة سخيفة أخرى لتقرير جديد. الوصول إلى البيانات أسهل ويتم بطريقة قياسية عبر لغة قياسية... كل هذا جميل، لكن مازالت هناك مشاكل ماثلة، ثم إن هناك حلقة مفقودة في كل الموضوع...

بالطبع... هذه الحلقة المفقودة هي التي تكمل الفكرة، وتجعل الموضوع كله جميلاً. لابد من طبقة تكمل الفراغ بين المستوى المنطقي الذي أصبحنا نفكر فيه، وبين العالم الفيزيائي الذي لا يعرف الحاسوب سواه. المشكلة أنك كنت تتثاءب طول الوقت عندما كنا نتحدث في الحلقات السابقة عن برنامج إدارة قواعد البيانات، وأنت تهمس في غيظ: ألم ينته بعد من هذا الهراء؟... هذا الهراء بالضبط هو الذي يجعل النموذج الجديد قابلاً للتطبيق، فبرامج إدارة قواعد البيانات هي التي تتولى ترجمة البنية المنطقية المتمثلة في شكل جداول إلى الحقيقة الفيزيائية على القرص الصلب. وهي التي تتولى بعض المهام التي كانت تسبب لنا صداعاً مزمناً أيام نظام الملفات إياه. الوصول المتزامن بدون مشاكل inconsistency، والذرية atomicity، والأمان security، هذا علاوة على تنفيذ كل عمليات إنشاء واستعلام قواعد البيانات (مجموعات الجداول) التي تتم بلغة SQL كما عرفنا. كانت Oracle من أوائل الشركات التي أنتجت برنامجاً من هذا النوع لتطبيق أفكار Codd.

برامج إدارة قواعد البيانات العلائقية يجب أن تكون علائقية. هذا يعني أن أصول ومبادئ النموذج العلائقي تطبق على قاعدة البيانات وعلى برنامج إدارته. هذا ما كافح Codd من أجل فرضه، خاصة بعد محاولة بعض الشركات مجرد تعديل برامجها الموجودة من قبل لتصبح برامج إدارة قواعد بيانات علائقية. وضع Codd مجموعة من القواعد (ترقيمها من صفر إلى 12) لتصبح معياراً يحكم على برنامج إدارة قواعد البيانات إن كان حقاً RDBMS (Relational Database Management System). نحن هنا لاتهمنا هذه القواعد، وسنتركها لمصممي برامج إدارة قواعد

البيانات، لكننا سنتعرض لبعض القواعد العامة التي تنطبق على قاعدة البيانات نفسها، والتي تشكل مبادئ أساسية لأي قاعدة بيانات علائقية. هذه المفاهيم هي فكرة المفاتيح وفكرة المجال و وأنواع العلاقات وقواعد التسوية، وأخيراً قواعد التكامل. كل واحدة من هذه الكلمات لا أشك في أنها على الأقل تمر عليك كثيراً (أو أنني أشك؟). بالمناسبة، بعض القواعد التي وضعها Codd كانت من الصرامة بحيث أنها لم تطبق حتى في أشهر برامج قواعد البيانات العلائقية، لكن الحياة تمضي كما تعلم...

يجب أن تستوعب تماماً فكرة الجدول وكيف أنه مجموعة من خصائص تتعلق بوحدة واحدة، حيث، مرة أخرى، تمثل الأعمدة هذه الخصائص، وكل صف يعني حالة واحدة (instance) من هذه الوحدة. الجدول الخاص بالمتقدمين لوظيفة applicants تمثل أعمدته خصائص المتقدم، وكل متقدم فعلي يتم تمثيله بسطر جديد. مباشرة بعد هذا المفهوم يجب أن تستوعب جيداً مفهوم المفتاح الأساسي. اشترط Codd في قاعدته الثانية ضمان الوصول إلى كل معلومة في أي خلية (تقاطع عمود مع سطر). أنت تحدد اسم الجدول واسم العمود (أحياناً يسمى الحقل) ثم ... هنا النقطة؛ يجب أن تملك وسيلة تحدد فيها سطراً معيناً من مجموعة صفوف الجدول. لن تستطيع ذلك ما لم يكن لديك مفتاح. المفتاح هو قيمة خاصة بصف محدد ممكن استخدامها للوصول إلى ذلك الصف. هذا يتضمن أن تكون هذه القيمة:

1. موجودة.
2. فريدة.

القيمة قد تقع في عمود أو أكثر لكنها لا تتكرر في أكثر من صف واحد، وإلا لم يمكن استخدامها لتحديد صف واحد بالضبط. هذه القيمة (التي قد تكون مركبة composite من عمودين أو أكثر) تسمى المفتاح الأساسي للجدول Primary Key. هذا شرط أساسي في النموذج العلائقي: لا بد لكل جدول من مفتاح أساسي.

دعنا الآن نوضح مفهوم المفتاح الأساسي بمثال. في جدول المتقدمين لوظيفة ممكن أن نعطي كل متقدم رقماً فريداً (بطريقة ما) بحيث نستطيع فيما بعد تحديد هذا المتقدم برقمه. هذا الرقم سيكون عموداً من أعمدة الجدول بالطبع، لكن يجب عدم تركه فارغاً وضمان عدم تكراره. كل برنامج إدارة قواعد بيانات يحترم نفسه سيفرض هذين الشرطين. الأكسس يفعل ذلك أيضاً، والمفتاح الأصفر الصغير الذي تضعه بجانب أحد الحقول عند تصميم الجدول يرمز إلى مفتاح الجدول الأساسي. لاحظ أننا لم نختر اسم المتقدم مثلاً أو تاريخ ميلاده كمفتاح أساسي للجدول لأنه يمكن جداً تشابه متقدمين في الاسم أو تاريخ الميلاد، وهذا لا ينفع في قاعدة بيانات علائقية. المفتاح الأساسي لا يمكن أن يتكرر في صفين مختلفين... هل من الممكن أن نختار إذاً مفتاحاً مركباً من عمودي الاسم وتاريخ الميلاد؟ ممكن، على افتراض أنه من المستبعد جداً تشابه متقدمين اثنين في الاسم وتاريخ الميلاد، لكن من المحبذ دائماً في المفتاح الأساسي أن يكون:

1. صغيراً قدر الإمكان.
2. ثابتاً قدر الإمكان.
3. بسيطاً قدر الإمكان.
4. مألوفاً قدر الإمكان.

هذا يعني أن المفتاح ذا الحقل الواحد أفضل من المفتاح ذي الحقلين أو أكثر. هذا على الأقل يسهل كتابة البرامج والاستعلامات فيما بعد علاوة على تخفيف عبء المعالجة. الثباتية تعني معدل احتمالية تغير المفتاح. على سبيل المثال، تغير اسم المتقدم احتماله أكثر من تغير رقمه الذي يعطيه إياه قسم الشؤون الإدارية أو النظام الآلي. لا حاجة البتة لتغيير الرقم، في حين أن الاسم قد يتغير على الأقل بسبب اكتشاف خطأ في إدخاله من البداية. ويمكنك أن تتفق معي في أن المفتاح الذي لا يتغير خير من المفتاح الذي يتغير. من المستحب أيضاً أن تكون معالجة المفتاح خفيفة وسريعة كما أشرنا في النقطة الأولى. معالجة الأرقام بشكل عام أسرع من معالجة سلاسل الحروف. في الأخير، كلما كان المفتاح مألوفاً للمستخدم وسهل التذكر كان هذا أفضل في حالة كتابة الاستعلامات وتحديد المراد. هذا يتحقق إذا كان للمفتاح معنى في الحياة العملية. لذلك تجد من ينصح بعدم استخدام أرقام مولدة تلقائياً من النظام كمفتاح أساسي. في حالة كحالة أرقام المتقدمين، من حيث هل لها معنى في الحياة العملية أم لا، تستطيع موازنة الإيجابيات والسلبيات واختيار ما تراه مناسباً. في النهاية أنت المصمم والمبرمج، والخطأ الذي ترتكبه عند التصميم ستدفع غالباً ثمنه عند البرمجة. حظاً سعيداً.

فقط من باب إكمال موضوع المفتاح الأساسي، قد تجد أحياناً أن أكثر من حقل (أو مجموعة حقول) تصلح كمفتاح، تماماً كما هو الحال في رقم المتقدم أو اسمه مع تاريخ ميلاده. كل الحقول المفردة أو المركبة التي تصلح لأن تكون مفاتيح أساسية (لأنها لا تتكرر في أكثر من سطر) تسمى مفاتيح مرشحة Candidate Keys ، والمفتاح الذي يقع عليه اختيارك كمصمم يسمى المفتاح الأساسي Primary Key، أما المفاتيح الباقية فتسمى بعدئذ مفاتيح بديلة Alternative Keys. نعم، هو مجرد هوس آخر بالمصطلحات التي تجعل الموضوع يبدو رهيباً.

أظن أنني قد أطلت قليلاً في هذه الحلقة. على كل حال، نستكمل إن شاء الله في الحلقة القادمة الكلام عن أهم مفاهيم ومبادئ قواعد البيانات العلائقية من أجل الوصول إلى طريقة التصميم السليمة لهكذا قواعد، ومن أجل الحصول على الجواب الذي لم تجب عنه بعد: هل أنت.....؟

رقم الحلقة (15) قواعد البيانات العلائقية: المزيد من المفاهيم الأولية

عرفنا أن فكرة قواعد البيانات العلائقية تقتضي تقسيم بيانات النظام في الحياة الواقعية إلى بنى منطقية أسميناها الجداول، وأسماها relations Codd. عرفنا أيضاً (أعلم أنك كنت تعلم ذلك منذ دهر، وأنك ربما اكتشفته قبل Codd نفسه، ولكن تجاوز عني قليلاً) أن كل جدول لا بد أن يحتوي على حقل واحد (عمود واحد) على الأقل لا تتكرر فيه نفس القيمة في صفين أو أكثر (مثل رقم قيد الطالب، الرقم الوظيفي لموظف، رقم فاتورة، رقم ملف لمريض، مجموع رقمي المشترك والشهر لقراءة استهلاك الكهرباء، وهكذا). هذا الحقل (أو مجموعة الحقول) أسميناه المفتاح الأساسي. بدون هذا المفتاح لا يمكنك الوصول إلى صف بعينه.

بعد ذلك ستقضي وقتك على الأرجح تعيد تجميع البيانات من هذه الجداول المنفصلة لتحصل على معلومات مفيدة من قاعدة بياناتك. صحيح أن الجداول مستقلة من حيث تعبيرها عن وحدات أو كائنات أو مفاهيم مستقلة، لكن النظام يتكون من جميع هذه الوحدات بعضها مع بعض، ولا بد من وجود ترابط بين هذه الوحدات المنفصلة. مثلاً، لا فائدة من بيانات العميل بدون الفواتير التي توثق مشترياته، و لا بد من معرفة درجة الموظف عند احتساب راتبه، وهكذا. المشكلة ليست في وجود هذا الترابط لأنه موجود بالفعل، لكن المشكلة في القدرة على التعبير عنه وإعادته بعد التقسيم إلى جداول... هنا يأتي دور العلاقات.

ميزة قواعد البيانات العلائقية أنها تعبر عن العلاقات بين الجداول المختلفة باستخدام البيانات نفسها. هذا يعني أنك لست مضطراً لاستخدام وسائل إضافية لحفظ العلاقات غير البيانات الموجودة أصلاً في الجداول. هذه الوسائل الإضافية هي على سبيل المثال المؤشرات pointers أو أي هياكل أخرى تربط بين الجداول. الثمن الذي تدفعه هو القليل من تكرار البيانات المقبول، وهو ثمن يسير لو تعلم. الآن ما معنى هذا كله؟ معناه أن العلاقات بين الجداول تحفظ باستخدام حقول مشتركة بين هذه الجداول. العلاقة بين جدول العملاء وجدول الفواتير هي حقل (عمود) رقم العميل الذي تكرره في الجدولين بالطبع؛ العلاقة بين جدول بيانات الطلاب وبين جدول العلامات هي حقل رقم قيد الطالب؛ العلاقة بين جدول المواد الدراسية وبين جدول العلامات أيضاً هي حقل رمز المادة؛ العلاقة بين جدول القراءات الشهرية وبين جدول بيانات المشتركين هي حقل رقم المشترك؛ العلاقة بين جدول الفحوصات وبين جدول بيانات المرضى هي حقل رقم المريض؛ العلاقة بين جدول الكتب وجدول الناشرين هي حقل رقم الناشر، وهكذا.

في كل الأمثلة السابقة، تلاحظ أنه قد تم حفظ العلاقة بين جدولين بواسطة المفتاح الأساسي في أحد الجدولين. رقم العميل، رقم قيد الطالب، إلى آخره، كل هذه مفاتيح أساسية (قيم لا تتكرر) في جدول واحد على الأقل. هذا طبيعي، لأنه لا بد من تمييز الصفوف المرتبطة بين الجدولين، وقد عرفنا أن هذا التمييز هو وظيفة المفتاح الأساسي. المفتاح الأساسي في جدول لا تتكرر قيمه، لكنه في الجدول الثاني قد تتكرر قيمه. مثلاً، رقم العميل لا يتكرر في جدول العملاء، لكنه من الممكن جداً أن يتكرر في جدول الفواتير، لأن العميل الذي يحترم نفسه كعميل سيشتري منك أكثر من مرة، وسيحصل على عدد من الفواتير. إنك لن تسميه عميلاً بغير ذلك على كل حال. لا يمكن لطالبين أن يملكا نفس رقم القيد في جدول بيانات الطالب، لكن نفس رقم قيد طالب سيتكرر في جدول العلامات لأن الطالب سيأخذ أكثر من مادة (لا أظنهم يقسطون المواد أيضاً هذه الأيام). رقم المريض لا يتكرر في جدول بيانات المرضى لكنه من الممكن أن يتكرر في جدول الفحوصات (سيحرص أصحاب المستشفى على أن يتكرر، لا تقلق)، وهكذا.

إذاً، فإن المفتاح الأساسي في جدول لم يعد أساسياً في الجدول الثاني، لأنه من الممكن أن يتكرر، هذا التكرار ضروري كما رأينا من أجل عكس العلاقات الفعلية في الحياة الواقعية. هذه الضريبة يدفعها المفتاح الأساسي لأنه هاجر من جدوله إلى جدول آخر، فأصبح مفتاحاً أجنبياً في الجدول الثاني. رقم العميل في جدول العملاء مفتاح أساسي له هيبته ولا يمكن أن يتكرر، لكنه في جدول الفواتير مجرد مفتاح غريب لا يملك نفس الميزات، ويمكن أن يتكرر. هل ذاق أحدكم من قبل مرارة الغربة؟ إن مفتاحنا الغريب هذا يتجرعها في كل مرة تضيف فاتورة من أجل عميل. هذا هو مفهوم المفتاح الأجنبي أو الغريب foreign key الذي يرتبط ارتباطاً وثيقاً بمفهوم العلاقات بين الجداول. الخلاصة أن المفتاح الأجنبي (أو الغريب أو الخارجي، تعددت الأسماء والغربة واحدة) هو مفتاح أساسي في جدول آخر، وهو وسيلة الربط بين الجدولين. لاحظ أن هذا الحقل (أو مجموعة الحقول) مكرر في جدولين، لكن لا بد من هذا لحفظ العلاقة كما أفضنا.

دعني الآن أقدم لمصطلح آخر قد تصادفه عند القراءة في هذه إلـ (خزعبلات) على حد تعبيرك، وهو مصطلح المجال domain. وهو هاهنا يتعلق بقيم المفتاح الأساسي والأجنبي. عرفنا أن المفتاح الأجنبي هو ذاته المفتاح الأساسي في جدول آخر لكن مع إمكانية التكرار. هذا لا يعني أن كل قيم المفتاح الأساسي يجب أن تتكرر على الأقل مرة في المفتاح الأجنبي، في البداية قد لا يكون لكل موظف إجازات مثلاً لذلك لن تجد كل قيم الرقم الوظيفي في جدول الإجازات. أيضاً، لا يجب أن يكون المفتاح الأجنبي بنفس اسم المفتاح الأساسي. لكن المهم أن تكون قيم المفتاح الأساسي وانعكاسه في الجدول الثاني وهو المفتاح الأجنبي من نفس المجال. المجال هنا يعني مجموعة القيم التي يمكن أن يأخذها المفتاح الأساسي. مثلاً، رقم العميل مجاله كل أرقام العملاء، الرقم الوظيفي مجاله كل الأرقام الوظيفية المحتملة، وهذا أضيق من مجموعة الأعداد الصحيحة على سبيل المثال.

مجال اسم الطالب من أجل التوضيح هو مجموعة الأسماء وليس مجموعة الحروف. مجال حقل نوع المتقدم هو القيمتين (ذكر أو أنثى) فقط. المزيد بإذن الله عن كيفية التأكد من هكذا قواعد فيما بعد.

اسمحوا لي أن أقف هنا مختصراً هذه الحلقة على مفهوم المفتاح الأجنبي ومجاله فحسب، على أمل استكمال بقية المفاهيم قريباً بإذن الله... لكن السؤال الذي أصبح مألوفاً جداً الآن ما يزال ينتظر الإجابة: هل أنت.....؟

رقم الحلقة (16) العلاقات

أخذنا في الحلقة السابقة نظرة على المفاتيح الأجنبية... هل كان العرض واضحاً؟ هل هناك أية استفسارات، تصحيحات، تعليقات؟ لست أدري... ولماذا لست أدري؟ لست أدري... أعني أن أحداً لم يهز حتى رأسه لكي يعطيني أي فكرة. على كل حال، موضوع المفاتيح الأجنبية يقودنا مباشرة إلى موضوع أنواع العلاقات. عرفنا أن الجداول هي انعكاس للكائنات في الحياة الواقعية، وأن هذه الكائنات يتعلق بعضها ببعض، والمفاتيح الأجنبية هي الوسيلة التي نعكس بها هذه العلاقات في الجداول. بقي أن نشير إلى أن هذه العلاقات على أنواع اعتاد الناس على تقسيمها من حيث عدد الكائنات المساهمة في العلاقة (الصفوف) من الطرفين (الجدولين) إلى ثلاثة أقسام، نمر عليها بإذن الله سريعاً فيما يلي.

علاقة واحد إلى واحد one-to-one (1-1) بين جدولين: وتعني أن لكل صف في الجدول الأول، هناك على الأكثر صف واحد فقط في الجدول الثاني يرتبط به. بلغة المفاتيح، هذا يعني أن المفتاح الأساسي في الجدول الأول إذا وجد في الجدول الثاني كمفتاح أجنبي فإنه لا يتكرر أيضاً. هذا النوع من العلاقات نادر في الحياة العملية؛ تخيل جدولين يقابل كل صف في الجدول الأول صفاً واحداً بالضبط (أو لا صف) في الجدول الثاني. من الممكن جداً إلصاق أعمدة هذين الجدولين وإنشاء جدول واحد بصف طويل. لاحظ أن المفتاح الأساسي في الجدول الأول، هو نفسه المفتاح الأساسي في الجدول الثاني، وهو نفسه يلعب دور المفتاح الأجنبي هنا وهناك. كل هذا انعكاس لحقيقة أن كائنين في الحياة الواقعية، مثلاً كائن مواطن وكائن بطاقة شخصية، إذا ارتبطا بعلاقة واحد إلى واحد فإنهما في النهاية الشيء نفسه، ويشكلان كائناً واحداً خصائصه (أعمدته) هي مجموع خصائص (أعمدة) الكائنين. وجود جدولين بينهما هذه العلاقة يعني وجود نفس المفتاح الأساسي للجدولين.

لماذا إذاً قد نجد مثل هذين الجدولين أصلاً إن كان الأمر كذلك؟ نحن لا ينقصنا المزيد من العلاقات وأنواعها، حاولوا اختصار المادة ولا تعقدوها، ليس من الضروري أن تكون هنالك دائماً ثلاثة أنواع... الحقيقة أن ثمة أسباباً أكثر من مجرد تعبئة منهاج قواعد البيانات لوجود جداول بينها علاقة واحد إلى واحد، يمكن ذكر ما يلي:

1. قد يحدث أن تكون علاقة واحد إلى واحد تعكس بالفعل ولو نادراً العلاقة في الحياة الواقعية بين كائنين يتم معاملتهما بشكل مستقل أحياناً. مثلاً، قد تتم معاملة البطاقة الشخصية في نظام السجل المدني بشكل منفصل عن بيانات المواطن. بالطبع يمكن دمج الاثنين في جدول واحد، لكن تصميم الجدولين يعكس الواقع بشكل أفضل. قد تسمع بمثل هذا عند النصارى مثلاً، حيث يرتبط جدول الزوج بجدول الزوجة بعلاقة واحد إلى واحد على سبيل المثال.
2. لأسباب تتعلق بأداء قاعدة البيانات، قد تختار فصل جدول واحد إلى اثنين وتربط بينهما بواسطة نفس المفتاح الأساسي بعلاقة واحد إلى واحد، لأن الجدول القديم كان كبيراً جداً أو (هل يتصور أحدكم حدوث هذا؟) يكون عدد حقول الجدول قد جاوز الحد المسموح به في الأكسس مثلاً (255 حقل! ستحتاج لشاشة بيل جيتس المزدوجة لمعاينة أفضل لهذا الجدول).
3. قد يكون السبب في فصل جدول إلى اثنين وربطهما بعلاقة واحد إلى واحد هو السرية. فمثلاً، قد تفصل بيانات الموظف الخاصة في جدول منفصل وتجعل صلاحية الوصول إلى هذا الجدول محدودة.
4. سبب آخر هو وجود بعض العمليات المتكررة (مثل نسخ احتياطي على سبيل المثال) على جزء محدود من الجدول (بعض الحقول فقط)، فيمكن فصل هذه الحقول في جدول منفصل لتسهيل هذه العمليات، ولا تنس طبعاً ربط هذا الجدول بالجدول الأصلي بواسطة المفتاح الأساسي ذاته.

الخلاصة أن هذه العلاقة تقتضي وجود نفس المفتاح الأساسي في جدولين...

علاقة واحد إلى متعدد one-to-many (1-M) بين جدولين: تعني أن لكل صف في الجدول الأول عدداً من الصفوف المرتبطة به في الجدول الثاني قد يكون صفراً أو واحداً أو أكثر، ولكن كل صف في الجدول الثاني يرتبط بصف واحد بالضبط من الجدول الأول. هذا هو نوع العلاقات الأكثر شيوعاً، وعادة لا تجد غيره في تصميم قاعدة بيانات، لأن النوع الثالث من العلاقات كما سنرى حالاً بإذن الله يتم تحويله إلى هذا النوع. لاحظ أن هذه العلاقات تقرأ من الجهتين بصورة مختلفة؛ فمثلاً، العلاقة بين جدول العملاء وجدول الفواتير هي علاقة واحد إلى متعدد، أما العلاقة من جدول الفواتير إلى جدول العملاء فهي علاقة متعدد إلى واحد. هذا يشبه أن العلاقة بينك وبين أبيك هي أنك ابنه، والعلاقة بين أبيك وبينك هي أنه أبوك! هذا ليس لغزاً بالطبع، لسنا في برنامج فوازير، ولكن هذه العلاقات تسمى بالفعل أحياناً علاقات أب-أبناء، لأن الأب قد يكون له ابن أو أكثر (أو ليس له أبناء، وعندها لا أدري لماذا يبقى اسمه أباً، ربما من باب التفاؤل)، أما الابن فليس له إلا أب واحد.

لكل عميل فاتورة أو أكثر، ولكل فاتورة عميل واحد فقط؛ كل قسم قد يحوي عدداً من الطلاب، لكن كل طالب ينتمي إلى قسم واحد فقط؛ لكل مريض عدداً من الفحوصات، لكن الفحص الواحد لا يكون إلا لمريض واحد؛ للموظف عدد من الإجازات، لكن الإجازة الواحدة لا تصرف إلا لموظف واحد فقط. في كل هذه الأمثلة نطبق العلاقة بتكرار حقل بين الجدولين، الحقل في جهة الواحد هو المفتاح الأساسي، وفي جهة المتعدد هو المفتاح الأجنبي،

وهذا الحقل يربط بين الجدولين من حيث أنه بإمكاننا فيما بعد استخراج فواتير العميل، طلاب القسم، فحوصات المريض، إجازات الموظف، وهكذا عبر تتبع هذا الحقل في الجدولين.

علاقة متعدد إلى متعدد many-to-many (N-M) بين جدولين: تعني أن كلاً من صفوف الجدولين قد يرتبط بصف أو أكثر (أو ولا بصف) من صفوف الجدول الثاني. الحقيقة أن هذا النوع من العلاقات يحدث أيضاً كثيراً في الحياة العملية، لكنك لن تجده في أي قاعدة بيانات علائقية. العلاقة مثلاً بين الكتب والمؤلفين هي علاقة متعدد إلى متعدد، لأن كل كتاب قد يكون له مؤلف أو أكثر، وكل مؤلف قد يكون ألف أكثر من كتاب. كذلك العلاقة بين الطالب والمادة؛ كل طالب يدرس أكثر من مادة، وكل مادة يدرسها أكثر من طالب. العلاقة بين المشترك والشهر علاقة متعدد إلى متعدد، لأن كل مشترك له استهلاك (وبالتالي قراءة عداد مثلاً) في أكثر من شهر، وكل شهر فيه استهلاك لأكثر من مستهلك بالطبع. لنأخذ أيضاً المثال الشهير للعلاقة بين جدول الأصناف وجدول الفواتير؛ كل صنف قد يظهر في تفاصيل أكثر من فاتورة، والفاتورة الواحدة قد تحوي أكثر من صنف. لا يمكن تطبيق هذه العلاقة بين الجدولين مباشرة لأن وسيلة الربط كما أفضنا هي المفتاح الأساسي في أحد الجدولين، الذي نضعه في الجدول الآخر كمفتاح أجنبي. علاقة متعدد إلى متعدد تعني أن المفتاح الأساسي سيتكرر، وهذا يكسر أبسط قواعد قواعد البيانات العلائقية. مثلاً، إذا جعلنا حقل رقم الصنف في جدول الفواتير للربط بين الأصناف والفواتير، فهذا يعني أننا سنضيف صفاً جديداً في جدول الفواتير مع كل صنف، لكن هذا الصف الجديد سيحمل نفس رقم الفاتورة، في حين أن رقم الفاتورة هو المفتاح الأساسي ولا يمكن أن يتكرر.

الحل؟ لا بد لتطبيق هذه العلاقة من كسرها إلى علاقتين من نوع واحد إلى متعدد. عالم الحاسوب مليء باستخدام قاعدة فرق تسد. وهي تثبت فعاليتها في الحقيقة. هذا يعني أنك ستنشئ جدولاً ثالثاً وسيطاً وظيفته هي الربط بين الجدولين. لذلك ننشئ دائماً جدول تفاصيل الفواتير. العلاقة بين جدول الأصناف وجدول تفاصيل الفواتير هي علاقة واحد إلى متعدد، لأن كل صنف قد يظهر عدداً كثيراً من المرات في هذا الجدول، كذلك العلاقة بين جدول الفواتير وجدول تفاصيل الفواتير هي علاقة واحد إلى متعدد، لأن الفاتورة قد تملك أكثر من بند (صنف) واحد. الحقل الذي يربط جدول الأصناف مع جدول التفاصيل هو رقم الصنف مثلاً، في حين أن الحقل الذي يربط جدول الفواتير مع جدول التفاصيل هو رقم الفاتورة. حقلي رقم الصنف ورقم الفاتورة هما مفتاحان أجنبيان، لكن مجموعهما هو المفتاح الأساسي في جدول التفاصيل. هذا يعني أن رقم الصنف يتكرر في جدول التفاصيل لأن الصنف عادة يتم شراؤه في كثير من الفواتير، وكذلك يتم تكرار رقم الفاتورة في جدول التفاصيل لأن هناك أكثر من صنف في نفس الفاتورة، وهكذا لا تستطيع استخدام أيّ منهما كمفتاح أساسي في جدول التفاصيل، لكن رقم الفاتورة مع رقم صنف محدد لا يتكرران معاً؛ كل صنف يظهر في الفاتورة الواحدة مرة واحدة فقط. إذا بدا كل هذا مربكاً فاقرأه مرة ثانية، وإذا كان ما يزال مربكاً فلا عليك، إنه مربك بالفعل في أول مرة.

كمثال آخر على العلاقات من نوع متعدد إلى متعدد خذ مثال الحساب والقيد في أنظمة المحاسبة... الحساب (مثلاً حساب الصندوق) يظهر في أكثر من قيد، وكل قيد يحوي أكثر من حساب (على الأقل حسابين: دائن ومدين). لتطبيق هذه العلاقة في قواعد البيانات العلائقية يتم إنشاء جدول تفاصيل القيد بالإضافة إلى جدولي الحسابات والقيود. المفتاح الأساسي في جدول الحسابات هو رقم الحساب مثلاً، والمفتاح الأساسي في جدول لقيود هو رقم القيد، أما المفتاح الأساسي في جدول تفاصيل القيود فهو مجموع حقلي رقم الحساب ورقم القيد، وهما كما تلاحظ مفتاحان أجنبيان لربط الجدوليم معاً.

أختم هذه الحلقة بملحوظة ... كما ترى يا أخي، نظريات قواعد البيانات كثيرة، وأنا لا أنوي الخوض في تفاصيلها، لكن هناك من نظرياتها ما لا يسوغ تجاهله. لست أعلم إلى أي مدى يرغب إخواني القراء في الخوض في هذا النقاش، وهل هو يسير بشكل مرضٍ أصلاً؟ أعتذر عن الإثقال على أي قارئ، لكن من باب الفضول فقط، أخبرني: هل أنت.....؟

رقم الحلقة (17) التسوية Normalization: مدخل

كنت أنوي الحديث في هذه الحلقة عن بعض الخواطر المتعلقة بالعلاقات، لكن قد قررت البدء في مناقشة التسوية normalization، وهو موضوع كما أشرت فيما سبق جداً مهم. سأحاول بإذن الله تنويع الأمثلة لأن التطبيق العملي للأفكار هو غاية منشودة من التعلم، وفي الوقت ذاته وسيلة فعالة للتعلم.

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...

موضوع التسوية يقع في قلب تصميم قواعد البيانات العلائقية، وعلى الرغم من أنك من الممكن أن تتجاوزه باستخدام بعض الأدوات كمخطط الكائنات والعلاقات ERD (سنرى هذا فيما بعد إن شاء الله)، إلا أن الإلمام به لا يمكن تعويضه ولا تفويته. من الجدير التنويه بأنك قد تجد عدداً من الترجمات لهذا المصطلح في العربية. الاسم الأصلي إنجليزي بالطبع للأسف، لأن أحداً من العرب لم يعد متفرغاً لمثل هذه التفاهات منذ بضع قرون، وقد تركوا المهمة للآخرين، واكتفوا بالاختلاف حتى في الترجمة. يدعى هذا المفهوم في الإنجليزية normalization، وقد تجد من ترجماته التسوية والتقييس، وهناك ما يدعى أيضاً normal forms، ومن ترجماتها الأشكال السوية والأنماط القياسية. لا تدع هذا يربكك، كل هذه المصطلحات تشير إلى مفهوم واحد أحاول أن أوضحه قليلاً في هذه الحلقة وما يليها بإذن الله من حلقة أو اثنتين.

في البداية دعنا نأخذ نظرة طائر عامة وسريعة على مفهوم التسوية. أين تقع قواعد التسوية؟ من الممكن أن ترى ذلك من بعيد، وهو سيساعدك على فهم دورها، ثم ستعرف ما هي بالضبط عندما تقترب أكثر. المفترض في تصميم قواعد البيانات العلائقية أنك ستصل إلى مجموعة من الجداول التي تعكس كائنات الحياة العملية والعلاقات بينها. من المحتمل جداً أن تصميمك لهذه الجداول قد يكون سليماً، خاصة إذا كانت قاعدة البيانات صغيرة وأنت تملك خبرة كافية. لكن من المحتمل جداً كذلك أن تصميمك قد يحوي العديد من الأخطاء، خاصة إذا كانت قاعدة البيانات ليست صغيرة جداً، ولم تكن أنت ذا خبرة كافية (حسناً، لا أقصد الانتقاص منك شخصياً، دعنا فقط نفترض ذلك من باب الجدل). ما المقصود بالضبط بأن تصميم الجداول قد يكون (سليماً)؟ من أجل ذلك قاموا بوضع تعريف رسمي لـ(سليماً) هذا. هذا التعريف هو عبارة عن مجموعة من الشروط يجب أن يستوفيها التصميم السليم. هذه الشروط لا يتم استيفاؤها جملة واحدة، وإنما تستطيع تطبيقها في خطوات. إذا طبقت بعض الشروط المحددة سلفاً كخطوة أولى، وأصبحت جداولك تقابل هذه الشروط، فإننا نقول إن جداولك أصبحت في الشكل السوي (أو النمط القياسي) الأول. إذا طبقت المزيد من الشروط، فإنك ترتقي بجداولك إلى الشكل السوي الثاني، وهكذا. هنالك العديد من الأشكال السوية (يمكن إيصالها إلى سبعة)، لكن معظم قواعد البيانات العملية لا تحتاج إلى أن تستوفي أكثر من الشكل السوي الثالث، وهذا ما أعتزم الوصول إليه إن شاء الله تعالى.

قبل أن نخوض في هذه الأشكال أو الأنماط السوية أو القياسية، دعنا نلق نظرة سريعة على نوع تلك الأخطاء التي وقعت فيها عندما لم تكن تملك الخبرة الكافية (تذكّر، من باب الجدل فقط!). هذا سيساعدنا على فهم الهدف من تلك الأشكال وكل هذه القوانين. من الصعب أن تدرك لماذا اخترعوا مفك العلب إذا لم تكن تعرف المعلبات والصعوبة في فتحها باستخدام السكين (هل سبق أن فتحت إحدى المعلبات باستخدام مفتاح باب عادي؟ لقد فعلتها أنا عدة مرات عندما كنت مضطراً، هذا يجعلني ماهراً عند البعض، ومتخلفاً عند الأكثرين). العدو الأول الذي قد يتسلل إلى تصميم الجداول، وهو الخطأ الذي يجب أن تواجهه بكل قواك، هو التكرار الزائد للبيانات redundancy، وأقول الزائد لأن هناك تكراراً لا بد منه، كما رأينا في حالة تكرار قيم المفتاح الأساسي في جدول كمفتاح أجنبي في جدول آخر من أجل إنشاء علاقة بينهما. هذا التكرار يؤدي إلى العديد من المشاكل لا ينبغي أن تسمح لها بالبقاء في قاعدة بياناتك. للأسف، كثير من التصاميم غير المحترفة تحوي بعض هذه المشاكل، ومن ثم ينتقل عبء معالجتها إلى طور البرمجة. هذا هو السبب في أنك تجد الآن كثيراً من اللف والدوران في الكود من أجل إتمام عمليات ما كان ينبغي لها أن تسبب كل هذا الألم للمبرمج (هذا هو أنت أيضاً في حالتنا). خذ هذا السر الآن، واجعله نصب عينيك: أغلب الصعوبات التي تواجهها في البرمجة مع قواعد البيانات يكمن حلها في تصحيح تصميم الجداول. الكثير من الترقيع في الكود سببه القليل (أو الكثير) من الأخطاء في تصميم الجداول.

من أول ما قد يتبادر للذهن من مشاكل التكرار الزائد للبيانات مشكلة إهدار مساحة القرص الصلب. على الرغم من أن هذه المشكلة لم تعد بنفس حدتها في الأيام الخوالي، إلا أن الفرق بين جداول تعاني من تكرار لا داعي له للبيانات، وأخرى جيدة التصميم قد يصل إلى حد غير مقبول خصوصاً في أنظمة قواعد بيانات سعتها القصوى محدودة مثل أنظمة الأكسس. على كل حال ليس هذا ما ينبغي أن يكون أكبر ما يقلقك. هناك مشاكل فنية أخرى تصل إلى أن تمنعك من التعبير عن معلومات معينة في قاعدة بياناتك، ومشاكل تولد ما يسمى بأخطاء التعديل والحذف. هذه الأخيرة يترجمونها دائماً في المصادر العربية بـ(شذوذ) التحديث و(شذوذ) الحذف عن المصطلح الإنجليزي update and delete anomalies. لكي نفهم هذه الأخطاء بصورة عملية، دعنا نفترض المثال التالي من عالم برامج المخازن والمبيعات.

هب أنك كنت بصدد تصميم قاعدة بيانات لبرنامج مبيعات مخزن ما، ولأنك قد قرأت في الحلقات السابقة أن التصميم يجب أن يبنى على التحليل، ومن مصادر التحليل ما يتوفر من مستندات وتقارير، طلبت عينة من الفواتير الصادرة في النظام الحالي (فواتير معدة يدوياً في مطابع)، وأخذت في دراستها... طبعاً لاحظت أن الفاتورة تحوي العديد من البيانات، من ضمنها بيانات رأس الفاتورة كرقم وتاريخ الفاتورة، وبيانات المخزن من عنوان وأرقام هاتف، واسم العميل وطريقة الدفع (نقداً أو بالآجل)، وبيانات الأصناف المباعة من نوع وكمية وسعر وحدة وخصم إن وجد، ثم إجمالي الفاتورة، بخلاف التواقيع وربما اسم البائع. أدركت سريعاً أنه لا داعي لحفظ بيانات الترويسة من اسم وعنوان المخزن وخلافه لأنها بيانات ثابتة لا تتغير، ويمكن تثبيتها في التقارير المطبوعة أو قراءتها من مصدر آخر لا علاقة له بحركة الفواتير. تجاهلت طبعاً التواقيع، لأنه لا حاجة أبداً لحفظها، ولو شئت لما استطعت إلا بجهاز خاص، طبعاً لم تكن متحمساً إلى هذا الحد. ثم قررت أن الفاتورة تشكل كائناً مستقلاً يمكن تمثيله في جدول واحد مستقل، سأختصر بنيته كما يلي لكي يكفي عرض الصفحة:

رقم الفاتورة	تاريخ الفاتورة	اسم العميل	هاتف العميل	رقم الصنف	اسم الصنف	الكمية	السعر	الإجمالي

نظرت إلى جدولك وأنت تشعر بالرضا... ثم تذكرت أنه يجب أن يكون لكل جدول في قاعدة بيانات علائقية مفتاح أساسي، هكذا حدد Codd بكل وضوح، إنها اختراعه وهو حر فيه. اخترع أنت نموذجاً لقواعد البيانات وأعلن بكل وضوح عدم الحاجة لمفتاح أساسي. لكن رقم الفاتورة لا ينفع هاهنا لأن الفاتورة الواحدة قد تشمل أكثر من صنف، وهكذا ينبغي إضافة صف جديد مع كل صنف بنفس رقم الفاتورة، في حين أن المفتاح الأساسي لا يتكرر في صفين. إنك الآن تبتسم بثقة الخبير وتقرر أن المفتاح الأساسي سيكون مجموع حقلي رقم الفاتورة مع رقم الصنف؛ إن الصنف الواحد لا يتكرر في الفاتورة الواحدة، وهكذا يكون رقم الفاتورة مع رقم الصنف مزيجاً فريداً لا يتكرر في صفين. كل هذا جميل، لكن اسمح لي الآن أن أنقل إليك الأخبار السيئة...

لحظة... خطرت لي الآن فقط فكرة وأنا أكتب هذا... ذكرت في الأعلى أنك ستصادف مشاكل في التعبير عن بعض المعلومات في قاعدة البيانات، بمعنى أنك لا تستطيع إدخال بعض البيانات، وأن هناك مشاكل في تحديث وعند حذف بعض البيانات... ما رأيكم أن أجعل هذا فرصة لبعض المشاركة العملية؟ ما رأيكم في أن أترك هذا السؤال كتمرين: ما هي الأخبار السيئة التي قد تصادفك فيما بعد مع هذا التصميم؟ هل تكره التمارين؟ لا تقلق، سأكتب الحلقة القادمة بإذن الله سريعاً، ثم إنني على الأقل لن أسألك السؤال الذي كنت تتوقعه: هل أنت.....؟

رقم الحلقة (18) الأشكال السوية: الشكل السوي الأول 1st NF

دعونا الآن نعد إلى تصميمنا السابق في الأعلى. أشرت إلى أن من مشاكل هذا التصميم عدم القدرة أحياناً على حفظ بعض المعلومات في قاعدة البيانات. مثال على ذلك أنك يجب أن تنتظر حتى يشتري منك عميل شيئاً ما ويحصل على فاتورة لكي تستطيع تسجيل اسمه ورقم هاتفه، بغير ذلك تظل عاجزاً عن حفظ بيانات العميل لأنك لا يمكن أن تحفظ صفاً في هذا الجدول بغير رقم فاتورة ورقم صنف (تذكر أن هذين هما معاً المفتاح الأساسي للجدول، ولا يمكن تركهما خاليين). نفس الكلام ينطبق أيضاً على بيانات الأصناف. هذه مشكلة تتعلق بإضافة البيانات، دعنا نفحص مشاكل التعديل والحذف.

من الواضح طبعاً أننا نعاني هنا من التكرار حتى النخاع. الاسم الكامل للعميل ورقم هاتفه سيتكرران (تكراراً زائداً بدون فائدة إضافية) مع كل فاتورة جديدة، بل ومع كل صنف في الفاتورة. مرة أخرى، ليست المشكلة الكبرى هنا في مقدار مساحة الخزن المهدرة، وإنما في المشاكل التي قد تواجهها عند القيام ببعض العمليات على الجدول. مثلاً، لنفترض أن هاتف عميل قد تغير يوماً ما (احتمال وارد جداً)، يجب عليك الآن تغيير رقم الهاتف في عدد كبير من الصفوف وليس في صف واحد فقط، لأن رقم الهاتف يتكرر مع كل صنف في كل فاتورة. إن هذا يعني أكثر من مجرد القيام بعدد أكبر من عمليات التعديل، إنه يعني احتمال الوقوع في وضع inconsistency الذي تكلمنا عنه من قبل عند الحديث عن مشاكل نظام الملفات لو تذكر، وهو أن تجد رقم هاتف نفس العميل بقيمتين مختلفتين في مكانين مختلفين كنتيجة لتعديل بعض أماكن وجوده وعدم تعديل البعض الآخر. هذا قد يحدث لسبب أو لآخر، لكن النتيجة هي عدم الوثوق في مصداقية النظام في أحسن الأحوال. أما في الأسوأ، فهو عدم اكتشاف هذا الاختلاف أصلاً إلا متأخراً جداً. هذا النوع من الأخطاء يسمى أخطاء (أو شذوذ anomaly) التعديل.

كيف يمكن أن نواجه مشاكل عند حذف البيانات من مثل هذا التصميم؟ كما توقعت أنت وكنت على وشك أن تقول، ليست المشكلة الوحيدة هي في الانتباه لحذف عدد من الصفوف عند إرادة حذف فاتورة مثلاً، لكن هناك احتمالية فقد بيانات عميل عند حذف آخر فاتورة له، أو فقد بيانات صنف عند حذف آخر فاتورة ظهر فيها هذا الصنف. يسمى هذا النوع من الأخطاء بشذوذ الحذف delete anomaly. في قاعدة بيانات تحترم نفسها، لا ينبغي أن نجد مثل هذه الإشكاليات. ومن أجل التخلص منها، تم تأسيس قواعد التسوية، عبر تحديد عدد من الأشكال (السوية) التي ينبغي أن تكون الجداول فيها. يتم الوصول بالجداول إلى هذه الأشكال عبر تحقيق شروط معينة، كل شرط منها يوصلك إلى شكل سوي. وينبغي أن تمر على الشكل السوي الأول حتى تصل إلى الثاني، وهكذا.

هل تقول إنني قد نسيت شيئاً؟ أنت تقصد أنك تتوقع أن أعطيك التصميم الصحيح لمثالنا الذي أشبعته ركلاً ولكماً، أقصد نقداً وتخطيئاً... أنا لم أنس ذلك في الواقع، ولا أحاول تناسيه أيضاً، لا تنظر إلي هكذا، وإنما هذا هو موضوع التسوية أساساً، أن تسوي التصميم ليحقق الأشكال السوية التي أسرد منها هنا الأشكال الثلاثة الكافية عملياً، ثم نمر بإذن الله عليها بشيء من التفصيل.

افتراض أولي: البيانات في قاعدة البيانات مقسمة إلى جداول كل منها له مفتاح أساسي.

1. يكون الجدول في الشكل السوي الأول First Normal Form (1NF) إذا كانت قيم كل أعمدته ذرية atomic، ولم تكن هناك مجموعات مكررة من الأعمدة.
2. يكون الجدول في الشكل السوي الثاني Second Normal Form(2NF) إذا كان في الشكل السوي الأول، وكل الأعمدة غير المفتاحية معتمدة تماماً على كامل المفتاح الأساسي.
3. يكون الجدول في الشكل السوي الثالث Third Normal Form (3NF) إذا كان في الشكل السوي الثاني، وكل الأعمدة غير المفتاحية مستقلة بعضها عن بعض.

هناك من يقول الآن: كل هذه بديهيات! لكن مع ذلك، اسمحوا لي أن أثرثر قليلاً حول هذه البديهيات لأن هذا هو موضوع السلسلة على كل حال، ولا يمكن أن أكمل الحلقة في تقشير البطاطا مثلاً... من أجل ذلك سأتعرض بإذن الله للعديد من الأمثلة، فمن كان يجد ذلك مملاً، بإمكانه تجاوز الأمثلة الإضافية لأنها مجرد تكرار لترسيخ المفهوم.

دعنا نعد إلى موضوع الفواتير مع الأصناف، ولنفترض أن طالباً مبتدئاً جداً (ربما في سنته الأولى) فكر ثم قدّر ثم جاء بالتصميم التالي لجدول الفواتير:

رقم الفاتورة	تاريخ الفاتورة	الصنف	السعر	الكمية	الصنف	السعر	الكمية	الصنف	السعر	الكمية

لأنه لاحظ أن هناك عدة أصناف في الفاتورة الواحدة، لكل منها سعر وكمية (تم اختصار بقية البيانات من أجل تبسيط النقاش ولكي يكفي عرض الصفحة). ينص شرط الشكل السوي الأول على أن مجموعات الأعمدة المكررة

ممنوعة. المجموعة (الصنف، السعر، الكمية) هي مجموعة مكررة من الأعمدة، وعلى هذا فهي تكسر قاعدة الشكل السوي الأول. ما الخطأ في مثل هذا التصميم؟ لاحظ من فضلك أنك في هذا التصميم بين الإفراط والتفريط؛ أعني أنك تضع حداً اصطناعياً لعدد الأصناف في الفاتورة الواحدة، وما يدريك حتى لو حجزت مكاناً لعشرين صنفاً ألا يأتي من يشتري واحداً وعشرين صنفاً في فاتورة واحدة يوماً ما؟ ثم إن غالبية الفواتير لا تزيد عدد أصنافها عن بضع أصناف، مما يجعل كل هذا العدد المحجوز بلا معنى للغالبية العظمى من الفواتير. أمر آخر: القصد من تصميم قواعد البيانات في المقام الأول هو الوصول الفعال للمعلومة، وهذا أبعد ما يكون في تصميم كهذا. تخيل استعلاماً بسيطاً عن الكميات المباعة من صنف محدد. يجب عليك أن تكرر البحث عن الصنف ليس في كل الصفوف فحسب، بل وفي أكثر من حقل، لأن ذلك الصنف قد يظهر في أي حقل من الحقول الثلاثة. قاعدة الشكل السوي الأول تقضي على مثل هذه المشاكل.

ربما أنك تفكر الآن بأنه من المستبعد أن يأتي أحدهم ويضع مثل هذا التصميم، لكنني أحب أن أؤكد لك أنهم لم يأتوا بهذه القاعدة من الفراغ، وأنك ستجد دائماً من يصنع أشياء غريبة، ربما كان مبتدئاً وربما لم يكن مبتدئاً جداً. سأضرب لك الآن مثالاً آخر من المحتمل أنك رأيته في مكان ما. كتصميم لقاعدة بيانات طلاب تم اقتراح الجدول التالي (كالعادة، أكتفي بالحقول المعبرة لغرض المساحة):

رقم الطالب	اسم الطالب	مادة1	مادة2	مادة3	مادة4	مادة5	مادة6	مادة7

على اعتبار أن لكل فصل دراسي جدول كهذا يسرد مواد ذلك الفصل. هذا التصميم لا يأخذ بعين الاعتبار إمكانية تغيير المنهج وزيادة أو نقصان عدد المواد في وقت ما، وهو احتمال أكبر من الصفر بكل تأكيد، دع عنك أن هذا يتطلب عدة جداول لنفس البيانات أصلاً، لأن لكل فصل مواد مختلفة. لاحظ أن الأعمدة هنا هي أسماء المواد، فإذا حاولت إضافة بعض المرونة بتسجيل اسم كل مادة (أو رمزها) مع علامة الطالب كالتالي:

رقم الطالب	اسم الطالب	مادة	علامة	مادة	علامة	مادة	علامة	مادة	علامة	مادة	علامة	مادة	علامة	مادة	علامة

فإن هذا يشبه بالضبط مثال أصناف الفواتير الأول، ويحمل نفس مشكلة الحدود الصناعية، ويكسر إلى أجزاء صغيرة قاعدة التسوية الأولى: لا مجموعات مكررة من الحقول.

مثال آخر لن يضر أحداً... بيانات كتاب في قاعدة بيانات مكتبة تشمل إلى جانب رقم وعنوان وعدد صفحات الكتاب وخلافه، ربما حقلاً لموضوع الكتاب أو تصنيفه (مثلاً برمجة، تصميم، إنترنت، قواعد بيانات...). المشكلة أن كتاباً واحداً ربما يتم تصنيفه تحت أكثر من موضوع. كتاب عن الأكسس مثلاً قد يصنف تحت تصنيف قواعد البيانات وتحت البرمجة على سبيل المثال. ربما قدّر المصمم أن أي كتاب لا يمكن أن يصنف تحت أكثر من ثلاثة تصنيفات مختلفة ولذلك وضع التصميم التالي لجدول الكتب:

رقم الكتاب	العنوان	الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	تصنيف1	تصنيف2	تصنيف3

أغلب الكتب لها تصنيف واحد فقط مما يعني فراغ مهدر، ومن الممكن أن أصنف كتاباً عن برمجة PHP مع MySQL على سبيل المثال تحت موضوع برمجة وقواعد بيانات وإنترنت ومصادر مفتوحة ولغات سكربت (خمس تصنيفات).

أمر آخر يعالجه الشكل السوي الأول، وإن كان احتمال مصادفته أقل. تنص قاعدة الشكل السوي الأول على أن كل حقل يجب أن تكون قيمته ذرية (ترجمة atomic، يجب أن تقبل ذلك). الفرض هنا أن الذرة غير قابلة للتقسيم، والمقصود أن في كل حقل (عمود) قيمة واحدة مفردة، لا مجموعة أو قائمة من القيم بأي شكل كان. مثلاً، في مثال جدول الكتب، قد يختار المصمم التصميم التالي:

رقم الكتاب	العنوان	الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	التصنيفات

ثم يكتب في حقل التصنيفات شيئاً مثل: (برمجة، قواعد بيانات، إنترنت). تخيل أن المطلوب منك هو استعلام يجمع الكتب على حسب تصنيفاتها. سيكون هذا تمريناً ثقيلاً، لكنني لن أطلب ذلك، لا تقلق. مثال آخر لمثل هذا التصميم تخصيص حقل واحد للعنوان الذي يشمل المدينة والبلد على سبيل المثال.

الآن، وبعد أن عقّدنا الأمور بما فيه الكفاية، كيف من الممكن تصحيح أمثال هذه الجداول لتحقق الشكل السوي الأول (لاحظ: لا أقول ليصبح تصميمها سليماً تماماً). نحن نسير في خطوات، ودائماً يتضمن الحل تقسيم الجدول إلى جداول أصغر من أجل التخلص من الإشكاليات. يسمى هذا التقسيم decomposition. في حالة متطلب (الذرية)، يمكن فصل القيم المتزاحمة في حقل واحد في حقول منفصلة، لكنك ستقع هنا في فخ مجموعات الأعمدة المتكررة. في هذه الحالة من الممكن تلافي هذا الخطأ بتحويل المجموعات المكررة أفقياً إلى مجموعات مكررة رأسياً. هذا يعني أن تحول المجموعات المكررة من أعمدة إلى صفوف. على سبيل المثال، في حالة الفواتير والأصناف، التصميم التالي يتخلص من مشكلة مجموعات الأعمدة المكررة:

رقم الفاتورة	تاريخ الفاتورة	الصنف	الكمية	السعر
000001	2009/05/01	صنف1	5	85
000001	2009/05/01	صنف2	1	300
000001	2009/05/01	صنف3	3	100

لاحظت التكرار؟ بالطبع، نحن لم نتخلص من كل المشاكل بعد، ما زلنا في الشكل السوي الأول. المهم أيضاً أن تلاحظ أن المفتاح الأساسي في هذا التصميم لا يمكن أن يكون رقم الفاتورة، لأسباب واضحة طبعاً، لذا من الممكن جعله مزيجاً من رقم الفاتورة والصنف. نفس الحل ينسحب على مثال المواد والكتب. يتم تعديل المفتاح الأساسي والتخلص من التكرار أفقياً إلى التكرار رأسياً.

أعلم أنني قد أثقلت عليك اليوم، لكن دعني أختم بمثال هو جزء من جدول المتقدمين لوظيفة، حيث نجد الحقول التالية كتصميم أولي:

الاسم	تاريخ الميلاد	محل الميلاد	العنوان	التليفونات 1، 2، جوال	المؤهل	خبرة1	خبرة2	خبرة3	اللغات

أعلم أنك لم تكمل كلامك بعد من أن التمارين غير مستحبة على الإطلاق، لكن يبدو أنني من النوع الذي لا يتعلم بسرعة، وإني لأزعم أن لديك الآن ما يكفي من المعرفة (أو أنني أظن ذلك؟) لتكتشف كلا النوعين من أنواع الخروقات للشكل السوي الأول. سأبدأ بإذن الله الحلقة القادمة بالتعليق على هذا الأمر قبل الانتقال إلى الشكل السوي الثاني من أشكال التسوية. لا، لم أنس السؤال بالطبع، ترى: هل أنت.....؟

رقم الحلقة (19) الأشكال السوية: الشكل السوي الثاني 2nd NF

لنعد إلى التصميم المعطى في آخر حلقة كتمرين على الشكل السوي الأول. بعد نظرتين أو ثلاث نكتشف أن هناك أكثر من موضع مرشح لكسر الشكل السوي الأول. حقل العنوان مثلاً، إن كان يشمل عدة تفاصيل على غرار المدينة والشارع والبناية والرمز البريدي، فيجب تقسيمه إلى عدة حقول لأن التصميم الحالي يكسر قاعدة الذرّية. بالنسبة للتلفونات، إذا عدنا إلى التحليل ووجدنا أن المطلوب هو بالضبط حقلين لأرقام الهاتف لا أكثر، ورقم جوال واحد، فإنه من الممكن تقسيم حقل التلفونات إلى ثلاثة حقول: اثنان لأرقام هواتف عادية، وواحد لرقم الجوال. بالنسبة للخبرات، فإنه ينبغي التخلص من هذه المجموعات المكررة من الأعمدة، واستبدال صفوف مكررة بها. بالطبع سوف يتغير المفتاح الأساسي، فلن يكون مجرد الاسم على سبيل المثال، بل الاسم مع الخبرة. فكر باللغات بشكل مشابه، لكن من أجل سهولة الشرح، دعنا نفترض أننا وصلنا إلى التصميم التالي من الشكل السوي الأول (مع الأخذ في عين الاعتبار ما ذكرناه عن أرقام الهواتف)، بعض البيانات الاعتباطية تساعد على فهم الفكرة:

الاسم	تاريخ الميلاد	محل الميلاد	العنوان	المدينة	هاتف1	هاتف2	جوال	المؤهل	الخبرة
اسم1	تاريخ1	مدينة1	عنوان1	مدينة1	هاتف1	هاتف2	جوال1	مؤهل1	خبرة1
اسم1	تاريخ1	مدينة1	عنوان1	مدينة1	هاتف1	هاتف2	جوال1	مؤهل1	خبرة2
اسم1	تاريخ1	مدينة1	عنوان1	مدينة1	هاتف1	هاتف2	جوال1	مؤهل1	خبرة3
اسم2	تاريخ2	مدينة2	عنوان2	مدينة2	هاتف3		جوال2	مؤهل2	خبرة4
اسم2	تاريخ2	مدينة2	عنوان2	مدينة2	هاتف3		جوال2	مؤهل2	خبرة5
اسم2	تاريخ2	مدينة2	عنوان2	مدينة2	هاتف3		جوال2	مؤهل2	خبرة6
اسم3	تاريخ3	مدينة3	عنوان3	مدينة4	هاتف4	هاتف5	جوال3	مؤهل1	خبرة7

لاحظ أن الاسم لا ينفع لوحده كمفتاح أساسي، لذا لا بد من اختيار الاسم مع الخبرة كمفتاح أساسي. مثلاً، الصف الأول يمكن تحديده حصرياً بالاسم (اسم1) والخبرة (خبرة1). هذا التصميم يشبه ما وصلنا إليه في حالة الفواتير والأصناف:

رقم الفاتورة	تاريخ الفاتورة	رقم العميل	اسم العميل	هاتف العميل	رقم الصنف	اسم الصنف	الكمية	السعر
000001	2009/06/07	1	عميل1	هاتف1	1	صنف1	8	100
000001	2009/06/07	1	عميل1	هاتف1	2	صنف2	1	250
000001	2009/06/07	1	عميل1	هاتف1	3	صنف3	100	50
000002	2009/06/07	2	عميل2	هاتف2	2	صنف2	5	250
000002	2009/06/07	2	عميل2	هاتف2	4	صنف4	1	1000
000003	2009/06/07	1	عميل1	هاتف1	2	صنف2	3	255

كما قلنا مراراً من قبل: المفتاح هنا هو رقم الفاتورة مع رقم الصنف. هذا الجدول أيضاً من الشكل السوي الأول. ماذا عن جدول الطلاب؟ خذ هذا الجدول المقتضب من الشكل السوي الأول، ويشمل الطلاب مع علاماتهم في المواد المختلفة:

رقم الطالب	اسم الطالب	رقم القسم	اسم القسم	رمز المادة	اسم المادة	عدد الساعات	العلامة
4000	طالب1	1	قسم1	A001	مادة1	3	3.3
4000	طالب1	1	قسم1	A002	مادة2	3	3.7
4000	طالب1	1	قسم1	A003	مادة3	2	3.8
4001	طالب2	1	قسم1	A001	مادة1	3	3.2
4001	طالب2	1	قسم1	A002	مادة2	3	3.0
4001	طالب2	1	قسم1	A003	مادة3	2	3.8

لا أحتاج للقول طبعاً بأن المفتاح هنا هو رقم الطالب مع رمز المادة (ها أنا قد قلت ذلك على أي حال). تصميم جدول الكتب في الشكل السوي الأول سيكون شيئاً كهذا:

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر	اسم الناشر	عنوان الناشر	التصنيف

المفتاح واضح من أسماء الحقول التي وضعت تحتها خطاً. هذه طريقة شائعة للتعبير عن المفاتيح في التصاميم والرسومات.

أنت الآن قد اطلعت على مجموعة من الجداول مصممة لأغراض مختلفة، لكنها كلها تحقق شرط الشكل السوي الأول. وأنت الآن كذلك تعقد حاجبيك في ريبة وتتمتم في شك: ما الذي فعلناه بالضبط؟ أنا أرى أطناناً من التكرار، والمشاكل التي كنت تثرثر عنها قبل حلقة ما زالت تترعرع في الجداول... أرى أنه من الأفضل أن تترك الحديث عن قواعد التسوية وتفتح محلاً لبيع الليمون البارد... هكذا تفيد الناس أكثر...

ولكن هذه ليست نهاية الحكاية، ما يزال أمامنا الشكل السوي الثاني، ثم يمكن أن أفكر في مشروع الليمون. ينص شرط الشكل السوي الثاني على أن الجدول يجب أن يكون في الشكل السوي الأول، وتكون كل الأعمدة غير المفتاحية معتمدة تماماً على كامل المفتاح الأساسي. الأعمدة غير المفتاحية تعني كل الأعمدة عدا العمود (أو الأعمدة) المكونة للمفتاح الأساسي. كلمة (كامل) المفتاح الأساسي هي كلمة السر هنا. تطبق هذه القاعدة على الجداول التي تحتوي على مفاتيح مركبة، أي تتكون من أكثر من حقل واحد. وهي تعني أن كل حقل غير الحقول المكونة للمفتاح يجب أن يعتمد على المفتاح الأساسي كاملاً بكل أجزائه، ولا يعتمد على جزء محدد منه فقط. بعبارة أخرى، يجب أن نستخدم كل حقول المفتاح الأساسي للوصول إلى حقل معين في سطر معين، وينبغي ألا يكون جزء فقط من المفتاح الأساسي كافياً لتحديد بعض الحقول. دعنا نركز على مثال الفواتير والأصناف لتوضيح هذا الكلام.

المفتاح الأساسي في جدول الفواتير هو مجموع رقم الفاتورة ورقم الصنف. هل نحتاج لكامل المفتاح، أي كلا حقليه معاً، للوصول إلى باقي الحقول؟ إذا أردنا أن نصل إلى كمية ما (مثلاً، الكمية 8 في الصف الأول)، فإننا بالفعل نحتاج إلى تحديد شيئين: رقم الفاتورة أولاً التي ظهرت فيها هذه الكمية، ثم رقم الصنف الذي بيعت منه هذه الكمية. لكن الحقيقة أن أغلب الحقول تكسر قاعدة الشكل السوي الثاني. مثلاً، لمعرفة تاريخ الفاتورة، أنت لا تحتاج لأكثر من رقم الفاتورة فقط. كذلك بمعرفة رقم الفاتورة 000001 على سبيل المثال، أنت تعرف أن العميل الذي اشترى الفاتورة هو العميل رقم 1 (وليس العميل 007 مثلاً). هذا يعني أن هناك حقولاً لا تعتمد على كامل المفتاح الأساسي بجزأيه. نفس الكلام يقال عن اسم الصنف؛ أنت تعرف اسم الصنف من رقم الصنف فقط، وهو نصف المفتاح الأساسي. هذا هو موضوع الشكل السوي الثاني باختصار، ينبغي ألا تكون هناك اعتمادات جزئية partial dependencies؛ إذا لم يتم استخدام المفتاح الأساسي كاملاً لتحديد الصف بكل حقوله، فإن هذا المفتاح لا يستحق هذا المنصب، وينبغي تقسيمه. إذا قسمت المفتاح فأنت تقسم الجدول، لأنه لا يكفي أيضاً استخدام نصف المفتاح فقط كمفتاح أساسي كما لاحظنا سابقاً. الحل؟ انقل الحقول التي تعتمد على جزء من المفتاح إلى جدول مستقل، واستخدم هذا الجزء كمفتاح أساسي في الجدول الجديد. في حالة مثالنا، هذا يعني توليد المجموعة التالية من الجداول:

جدول الفواتير:

رقم الفاتورة	تاريخ الفاتورة	رقم العميل	اسم العميل	هاتف العميل

جدول الأصناف:

رقم الصنف	اسم الصنف

جدول تفاصيل الفواتير:

رقم الفاتورة	رقم الصنف	الكمية	السعر

لاحظ أنه تم نقل الحقول التي تعتمد على رقم الفاتورة فقط إلى جدول مستقل (جدول الفواتير)، وبالطبع فإن المفتاح الأساسي في هذا الجدول هو رقم الفاتورة. وتم نقل الحقل الذي يعتمد على رقم الصنف فقط، وهو حقل اسم الصنف إلى جدول آخر (جدول الأصناف)، ومفتاحه رقم الصنف. وأخيراً تم الإبقاء على بقية الحقول التي تعتمد على كل المفتاح الأساسي (رقم الفاتورة مع رقم الصنف) في جدول باسم تفاصيل الفواتير. لاحظ كذلك أنه بالرغم من تقسيم الجدول إلى ثلاثة جداول، إلا أننا احتفظنا بالعلاقة بين هذه الجداول بحيث يمكننا فيما بعد ربطها من جديد عند الاستعلام عن البيانات. هذه العلاقة حافظنا عليها عبر المفاتيح الأجنبية. رقم الفاتورة في جدول تفاصيل الفواتير هو مفتاح أجنبي (مفتاح أساسي في جدول الفواتير)، ووظيفته الربط بين جدول الفواتير وجدول تفاصيل الفواتير. وبالمثل، رقم الصنف يربط بين جدول الأصناف وجدول تفاصيل الفواتير. بالنسبة للسعر فإنه على شاكلة الكمية، يمكن معرفته بمعرفة مجموع رقم الفاتورة مع رقم الصنف. أعلم أن البعض يعد السعر من خصائص الصنف، أي أنك تعلم السعر بمجرد معرفتك لرقم الصنف، لكن الافتراض هنا هو الافتراض العملي بأن سعر الصنف متغير في كل فاتورة (لاحظ سعر صنف2). ربما يكون للصنف سعر مبدئي افتراضي، لكن من الممكن جداً أن يتم البيع بسعر مختلف حسب الزبون، كما أن هذا يسمح بالاحتفاظ بتاريخ سعر الصنف.

يمكنك أن تطبق هذا المبدأ على بقية الأمثلة... بالمناسبة، ليست كل الجداول في أفضل تصميم بعد، لكنها ستصبح بإذن الله كذلك في الحلقة القادمة، بعد أن نطبق شرط الشكل السوي الثالث. أنا الآن سأصل بيني وبينك إلى حل وسط: سأضع إن شاء الله حل مثالين، وأترك لك المثال الأخير (مثال الكتب، وهو الأسهل).

جدول المتقدمين (للوظائف):

الاسم	تاريخ الميلاد	محل الميلاد	العنوان	المدينة	هاتف1	هاتف2	جوال	المؤهل

جدول الخبرات:

الاسم	الخبرة

جدول الطلاب:

رقم الطالب	اسم الطالب	رقم القسم	اسم القسم

جدول المواد:

رمز المادة	اسم المادة	عدد الساعات

جدول العلامات:

رقم الطالب	رمز المادة	العلامة

أنت ترى، كل التسهيلات الممكنة تقدم لتدفعك للمحاولة والتجربة، فحاول من فضلك وجرب، فربما تصل إلى إجابة عن السؤال: هل أنت.....؟

رقم الحلقة (20) الأشكال السوية: الشكل السوي الثالث 3rd NF

أين وقفنا في المرة السابقة؟ بالنسبة لي، لا أكاد أذكر، لذا لا أتوقع منك الكثير. على أي حال، دعنا نمر سريعاً على المثال من قاعدة بيانات المكتبة لنسترجع الخطوات اليسيرة التي مررنا بها في طريق التسوية. لنتخيل أن التصميم الأولي لجدول الكتب كان كالتالي:

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر	اسم الناشر	عنوان الناشر	التصنيفات

بسبب شرط الشكل السوي الأول، لا يمكن الاحتفاظ بالتصنيفات التي قد تتعدد للكتاب الواحد في حقل واحد (من يذكر اسم هذا الشرط؟ لا أحد؟ لا بد أنكم تمزحون، إذا لم تذكر الذرية atomicity، فماذا يمكن أن تذكر؟). الحل هنا هو تقسيم هذا الحقل إلى عدد (كافٍ) من التصنيفات، وقد تم هنا افتراض أن هذا العدد الكافي هو ثلاثة تصنيفات:

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر	اسم الناشر	عنوان الناشر	تصنيف1	تصنيف2	تصنيف3

بسهولة يمكن اكتشاف أن هذا التصميم ما يزال خارج إطار الشكل السوي الأول لأنه يحوي مجموعات مكررة من الأعمدة. الحل هنا أن نحول التكرار في الأعمدة إلى تكرار في الصفوف، أي أن تصنيفات كتاب ستمتد رأسياً في صفوف عوضاً عن الامتداد أفقياً في أعمدة، كالتالي (البيانات لتوضيح التصميم):

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر	اسم الناشر	عنوان الناشر	التصنيف
11111	كتاب1	1	2001	500	5	1	ناشر1	عنوان1	برمجة
11111	كتاب1	1	2001	500	5	1	ناشر1	عنوان1	إنترنت
22222	كتاب2	1	2006	690	2	1	ناشر1	عنوان1	قواعد بيانات
33333	كتاب3	2	2007	800	1	2	ناشر2	عنوان2	تصميم
33333	كتاب3	2	2007	800	1	2	ناشر2	عنوان2	برمجة
44444	كتاب4	3	2005	1000	3	3	ناشر3	عنوان3	قواعد بيانات
44444	كتاب4	3	2005	1000	3	3	ناشر3	عنوان3	برمجة
44444	كتاب4	3	2005	1000	3	3	ناشر3	عنوان3	وب

أصبح المفتاح الأساسي الآن هو مجموع رقم الكتاب مع التصنيف. (بالمناسبة، الكتاب رقم 11111 قد يكون في البرمجة بـPHP أو ASP على سبيل المثال، هل تستطيع أن تذكر مثالاً للكتاب 33333 و44444؟). نلاحظ هنا أننا مازلنا نعاني من الكثير من التكرار، على الرغم من أن تصميمنا يقع في الشكل السوي الأول. بيانات أي كتاب تتكرر كاملة مع كل تصنيف للكتاب. لذلك نتقدم خطوة إلى الأمام ونتطلع إلى الشكل السوي الثاني. سرعان ما نكتشف أننا نحطم شرط الشكل السوي الثاني إلى قطع صغيرة. من يذكر قاعدة الشكل السوي الثاني؟ أنت؟ ممتاز، لقد علمت أنك ستذكرها من دون البقية، لقد كنت دائماً متابعاً جيداً. الشرط هنا ألا توجد اعتمادات جزئية. لا نريد حقولاً تعتمد فقط على جزء من المفتاح الأساسي، وليس كامل المفتاح الأساسي. تعرف أن كلمة (تعتمد) هنا تعني إمكانية تحديد حقول في سطر معين (إمكانية الوصول إلى صف بعينه). مثلاً، كل الحقول في جدول الكتب ما عدا رقم الكتاب والتصنيف يمكن معرفتها باستخدام رقم الكتاب فقط. يمكن تحديد الناشر وعدد الصفحات والطبعة بمعرفة رقم الكتاب فحسب دون معرفة تصنيف الكتاب، على الرغم من أن الاثنين معاً يشكلان المفتاح الأساسي للجدول. هذا الوضع لا يمكن السكوت عنه، ويجب فوراً تقسيم الجدول إلى جدولين: واحد يشمل كل الحقول التي تعتمد فقط على رقم الكتاب، والثاني يحوي بقية الحقول. أنت ترى أن هذا يقودنا إلى هذا التصميم:

جدول الكتب:

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر	اسم الناشر	عنوان الناشر
11111	كتاب1	1	2001	500	5	1	ناشر1	عنوان1
22222	كتاب2	1	2006	690	2	1	ناشر1	عنوان1
33333	كتاب3	2	2007	800	1	2	ناشر2	عنوان2
44444	كتاب4	3	2005	1000	3	3	ناشر3	عنوان3

جدول تصنيفات الكتب:

رقم الكتاب	التصنيف
11111	برمجة
11111	إنترنت
22222	قواعد بيانات
33333	تصميم
33333	برمجة
44444	قواعد بيانات
44444	برمجة
44444	وب

يجدر بي أن أذكر هنا أن هذا التصميم كان ليشبه تماماً حكاية الفواتير والأصناف وتفاصيل الفواتير، لكننا نجد هنا جدولين، وليس ثلاثة. ذلك لأننا لم نستخدم أرقاماً لترميز التصنيفات المختلفة للكتب. كان الوضع سيكون هكذا:

رقم الكتاب	رقم التصنيف	اسم التصنيف
11111	1	برمجة
11111	2	إنترنت
22222	3	قواعد بيانات
33333	4	تصميم
33333	1	برمجة
44444	3	قواعد بيانات
44444	1	برمجة
44444	5	وب

لاحظ كيف أصبح المفتاح الأساسي في جدول تصنيفات الكتب هو رقم الكتاب مع رقم التصنيف؛ لقد فضلنا الرقم على الاسم كمفتاح. هذا التصميم بدوره يخالف الشكل السوي الثاني، لأن اسم التصنيف لا يعتمد إلا على رقم التصنيف وليس على كامل المفتاح. هذا يقودنا إلى تقسيم الجدول كما يلي:

جدول التصنيفات:

رقم التصنيف	اسم التصنيف
1	برمجة
2	إنترنت
3	قواعد بيانات
4	تصميم
5	وب

جدول تصنيفات الكتب:

رقم الكتاب	رقم التصنيف
11111	1
11111	2
22222	3
33333	4
33333	1
44444	3
44444	1
44444	5

ربما سيبدو واضحاً أكثر عندما نناقش بإذن الله مخطط الكائنات والعلاقات ERD (Entity-Relationship Diagram) كيف أن التصميم الأول يرجع إلى معاملة التصنيف كخاصية من خصائص الكتب (خاصية متعددة القيم multi-valued property)، أما التصميم الثاني فينشأ عن معاملة التصنيف ككائن مستقل له رقم واسم. في الحالة الأخيرة تكون العلاقة بين كائن الكتب (وبالتالي جدول الكتب)، وكائن التصنيفات (وبالتالي جدول التصنيفات) علاقة متعدد إلى متعدد، ولذلك نجد جدولاً وسيطاً هو تصنيفات الكتب لتمثيل هذه العلاقة في قاعدة البيانات. كلا التصميمين قد تم الوصول إليه عبر اتباع قواعد التسوية حتى الشكل السوي الثاني.

لاحظ لو سمحت أنني أشرت في الفقرة السابقة إلى ثلاثة مفاهيم مختلفة في تصميم قواعد البيانات، أحب أن أؤكد وأوضح الفرق بينها لكي لا تختلط عليك الأمور في هذه النقطة، لكن لم يكن مناص من ربط بعضها ببعض لأنه من المهم فهم العلاقات بين المفاهيم المختلفة في نفس المجال وأين تلتقي وأين تفترق، من أجل فهم حقيقي لذلك المجال. تمت الإشارة في الفقرة السابقة إلى موضوع مخطط الكائنات والعلاقات، وإلى قواعد التسوية، وإلى مفهوم العلاقات بين الجداول. سبق مناقشة العلاقات في حلقة سابقة، ونحن الآن في خضم مفهوم التسوية، وتلحق بإذن الله مناقشة استخدام مخطط الكائنات والعلاقات فيما بعد. من الجيد أن تلاحظ الآن أننا وصلنا إلى تصميم معين انطلاقاً من تصميم أولي، ثم باستخدام قواعد التسوية. من الممكن أن نصل إلى نفس التصميم باستخدام مخطط الكائنات والعلاقات ثم تحويل المخطط إلى جداول. في كلتي الحالتين، حصلنا على التمثيل الصحيح لعلاقة متعدد إلى متعدد.

أعود الآن قبل أن أنسى إلى سؤال تجاوزته إلى حين عندما تحدثنا عن تصنيفات الكتب: لماذا استخدمنا أرقاماً للتصنيفات؟ لماذا لم نكتفي بالأسماء؟ هل في استخدام الأرقام ميزة إضافية؟ لا تخف، لن أترك هذا كتمرين، لكن لا أشك في أن لديك سبب ظاهر على الأقل، وهو أن استخدام الأرقام، خاصة في الربط بين الجداول، أسهل وأسرع في المعالجة. لكن أريد أن أوضح أن الفكرة الأساسية هنا هي معاملة التصنيفات ككائن منفصل له خصائص هي الرقم والاسم هنا، هذا يتيح فصله في جدول منفصل، واستخدام الأرقام للربط بينه وبين جدول تصنيفات الكتب. وجود أسماء التصنيفات في جدول مستقل، ونيابة الأرقام عنها في الجداول الأخرى يجنبنا مشاكل التحديث والحذف. مثلاً، عند تغيير اسم تصنيف من وب إلى الشبكة العنكبوتية على سبيل المثال، لا يتم التغيير إلا في جدول التصنيفات، وليست هناك حاجة لتحديث ربما عشرات السجلات التي تحوي كتباً تصنيفها هو (وب). لاحظ أن الأرقام لن تتغير في الجداول الأخرى. كذلك عند حذف آخر كتاب من تصنيف وب مثلاً، قد لا يعلم المطلع على الكتب وتصنيفاتها فيما بعد أن هناك تصنيفاً باسم (وب)، لأنه اطلع على الكتب وتصنيفاتها بعد حذف آخر كتاب من هذا التصنيف. لكن بوجود جدول خاص للتصنيفات، سيبقى التصنيف بغض النظر عن وجود كتب تندرج تحته.

سأفترض الآن فرضاً خطيراً، وهو أننا قد فهمنا كل ما سبق، وفي أهبة الاستعداد للتقدم نحو الشكل السوي الثالث. من كان لديه أدنى اعتراض على هذا الافتراض فليعلن ذلك الآن. حسن جداً، لم أتوقع أن أحداً لن يعترض، لا تقولوا فيما بعد أنني لم أسأل! هذه هي فائدة الكلام من وراء لوحة المفاتيح، لو كنا نتحدث وجهاً لوجه، لقفز لي الآن أكثر من عشرة قراء أشداء كلهم يصيح في نفس الوقت ويعلن بكل وضوح أنه، مع احترامي، من المستحيل أن يفهم أحد شيئاً من مثل هذا الهراء، وأنه لولا أن يقال أن المروءة قد ضاعت في الصحراء، أقصد في المنتدى، لكانوا قد أروني ما أستحقه بالضبط. لا بأس، دعونا نستمتع بالتحدث من وراء لوحة المفاتيح، ونصر على أن كل الكلام السابق مفهوم. لنستخدم الآن نفس مثال الكتب الذي بدأنا به في هذه الحلقة لتوضيح الشكل السوي الثالث...

تأمل في جدول الكتب أعلاه، وأخبرني عن أي عيب تلحظه. بالضبط، على الرغم من عدائنا السافر للتكرار، إلا أننا نرى أن بيانات الناشر من اسم وعنوان قد تتكرر بعدد مرات الكتب التي نشرها (انظر الناشر رقم 1). هذا يعرضنا لنفس نوع المشاكل التي تحدثنا عنها مراراً وتكراراً. إذا كنت تنتظر أن يزيل الشكل السوي الثالث هذا التكرار فأنت بارع حقاً. بالفعل، ينص شرط هذا الشكل على ألا تعتمد أي من الحقول غير المفتاحية على بعضها. هذا يعني أن كل الحقول التي لا تدخل ضمن حقول المفتاح الأساسي يجب أن تعتمد على المفتاح الأساسي فقط، ولا يسمح بوجود حقول تعتمد على حقول أخرى غير مفتاحية. مثلاً، حقلا اسم وعنوان الناشر يعتمدان على رقم الناشر، وهو حقل غير مفتاحي. لاحظ الفرق بين هذا الشرط وشرط الشكل السوي الثاني الذي يفرض أن تكون الحقول معتمدة على (كامل) المفتاح الأساسي. هذا الشرط يضيف ألا تكون هناك اعتمادات فرعية عابرة transitive dependencies، بين حقول أخرى غير مفتاحية. حقل عنوان الناشر على سبيل المثال، يعتمد بالطبع على كامل المفتاح، وهو رقم الكتاب، لكنه يعتمد عليه اعتماداً غير مباشر (عابر) عبر حقل رقم الناشر، بمعنى أن عنوان الناشر يعتمد على رقم الناشر، ورقم الناشر نعرفه بمعرفة رقم الكتاب. نفس الكلام يقال عن اسم الناشر. الحل؟ بالطبع، الحقول التي تعتمد على غير المفتاح الأساسي يتم نقلها إلى جدول مستقل مع الحقل الذي تعتمد عليه. يلعب هذا الحقل دور المفتاح الأساسي في الجدول الجديد، ودور المفتاح الأجنبي في الجدول الأول. ترجمة هذا الكلام هي هذان الجدولان بدلاً من جدول الكتب (آخر شكل لجدول الكتب وصلنا إليه بعد استيفاء شرط الشكل السوي الثاني):

جدول الكتب:

رقم الكتاب	العنوان	رقم الطبعة	سنة الطبعة	عدد الصفحات	عدد النسخ	رقم الناشر

جدول الناشرين:

رقم الناشر	اسم الناشر	عنون الناشر

الآن فقط نستطيع أن نقول أننا وصلنا إلى تصميم جيد لجزء قاعدة بيانات المكتبة الذي تناولناه. بتطبيق نفس الشرط على جدول الفواتير التالي من الحلقة السابقة:

رقم الفاتورة	تاريخ الفاتورة	رقم العميل	اسم العميل	هاتف العميل

نصل إلى استخراج بيانات العميل (التي تعتمد على رقم العميل) في جدول مستقل، لنحصل على الجدولين التاليين إلى جانب جدول الأصناف وتفاصيل الفواتير (لاحظ أنني أضفت، بسبب توفر المساحة، بعض الحقول التي استبعدتها سابقاً لكنك تجدها عادة في مثل هذه الجداول):

جدول الفواتير:

رقم الفاتورة	تاريخ الفاتورة	رقم العميل	نوع الفاتورة	طريقة الدفع	رقم الموظف

جدول العملاء:

رقم العميل	اسم العميل	نوع العميل	الهاتف	الجوال	العنوان	المدينة

عليك أنت أن تكمل تطبيق الشكل السوي الثالث على جدول الطلاب.

بهذا أختم الكلام عن الأشكال السوية، لأن غالب قواعد البيانات العملية لا تعرف أكثر من الشكل الثالث، فإذا وصلت إلى هنا بأمان، فتقبل تهاني. على كل حال، ما يزال أمامنا الكثير لنتناقش فيه بإذن الله، وربما أتحدث في الحلقة القادمة إن شاء الله تعالى عن إزالة التسوية de-normalization (من باب طرق الحديد وهو ساخن). بالرغم من أن أمامنا الكثير بعد، لكن الوقت قد حان لتفكر جدياً: هل أنت.....؟

رقم الحلقة (21) إزالة التسوية Denormalization

أبدأ بملحوظة تمهد تماماً لموضوع هذه الحلقة الذي لن يكون بإذن الله أكثر من مجرد ملحوظات استدراكاً على موضوع التسوية. ربما كان من الأجدر الدخول في صلب الموضوع مباشرة، وهو أن المبرمج المجرب قد يلجأ في النهاية إلى التخلص من بعض قيود التسوية لأغراض عملية. هذه الأغراض العملية تتلخص إجمالاً في أداء وتعقيد الاستعلامات التي قد نحتاجها فيما بعد لجمع البيانات من عدد من الجداول. عرفنا أن التسوية تميل إلى زيادة عدد الجداول، وهذا يؤدي إلى زيادة عدد الروابط joins في عبارات SQL (سنتكلم عنها إن شاء الله تعالى في الحلقات الأخيرة)، مما يعقد العبارات، ويستهلك وقتاً أطول في تنفيذها. التخلص من بعض قيود التسوية يعني الرضا بإعادة دمج بعض الجداول مرة أخرى، على الرغم من عدم تحقيقها لشكل سوي أو أكثر. هذه العملية تسمى إزالة التسوية Denormalization، وهي مصطلح آخر تستخدم فيه البادئة الإنجليزية de وتعني إزالة الشيء أو الضد من الشيء. هذا يشبه في عالم البرمجة مصطلح debugging بمعنى إزالة العلل والأخطاء bugs، وفي عالم الصيانة مصطلح defragmentation بمعنى إزالة الـ fragmentation، أي إزالة التجزئة من القرص الصلب. لكن الفرق أن الأخطاء المنطقية في البرمجة، وتجزئة الملفات في القرص الصلب مكروهة في الأصل، لكن التسوية في حالتنا هذه هي هدف يسعى إليه من البداية. كما تعلم، كثيراً ما نسمع عن الفرق بين النظريات المحضة والميدان العملي.

قبل أن يشط بك الخيال بعيداً، أود أن أذكر بوضوح أن التسوية هي الهدف الذي ينبغي أن تتأكد من فهمه والوصول إليه، لأنه الأصل، وهو طريق التصميم الجيد لقاعدة البيانات. إزالة التسوية ينبغي أن تفكر بها كاستثناء للقاعدة، وليس طريقة للتهرب من القيام بواجبك كمتعلم أولاً، ثم ممارس لتصميم قواعد البيانات. هذه من الأشياء التي تميز الخبراء عادة: أن تكسر القاعدة مع معرفتك التامة بها، وبتبعات كسرها، وبكيفية كسرها، وبالطبع لسبب وجيه لكسرها من الأصل. إن الطبيب الحاذق على سبيل المثال قد يخالف الجرعة النظرية، مع الإدراك الكلي لعواقب زيادة الجرعة، وكيفية التعامل مع هذه العواقب، وفي البداية مع سبب وجيه لهذه المخالفة. لا يمكن أن يكون السبب مثلاً من باب أن (الزيادة خير من النقصان)، أو لأن طعم الدواء لذيذ. لكن سبباً مثل معرفة الطبيب بأن مناعة المريض المحلي أقل من متوسط مناعة المريض الذي تم تحديد الجرعة على مقاسه، هو سبب منطقي. سأبوح لك الآن بواحد من الأسرار التي أرهقك بها بين الحين والآخر في هذه السلسلة: الفرق بين الخبير والمبتدئ، هو أن الخبير يحطم القواعد بعد العلم بها، في حين أن المبتدئ يحطم القواعد دون العلم بها. هذا الفرق كبير، لكن لأن النتيجة تكون أحياناً واحدة، اغتر الكثير من المبتدئين من ناحيتين؛ من ناحية اغتروا بأنفسهم، ومن ناحية أخرى اغتروا بالخبراء. الاغترار الأول كان بإحسان الظن الزائد بالنفس، والاغترار الثاني كان بإساءة الظن الزائدة بالآخرين (بعضهم خبراء حقيقيون).

دعوني الآن أنتقل من قسم الفلسفة إلى قسم الأمثلة العملية. لكي تستوعب جيداً معنى denormalization، ولماذا يمكن أن تستخدمه، سأضرب بإذن الله مثالين، كليهما من جدول الفواتير المألوف. عرفنا أن جدول الفواتير قد يحوي حقلاً لرقم العميل، وقد يحوي أيضاً حقلاً لرقم الموظف، وهو البائع المشرف على الفاتورة، ربما بسبب نظام عمولات على البيع أو ما شابه. لنفترض من أجل مثالنا أن عميل الفاتورة مهم جداً. ربما كان النظام لمستودع كبير يورد بالآجل للعملاء، أو من أجل الربط بنظام محاسبي أو خلاف ذلك. ما يعنينا هنا أنه بسبب ذلك، فإن كل استعلام وتقرير عن الفواتير يجب أن يظهر اسم العميل. قد (أقول قد) يختار المصمم (أو المبرمج، لأنه هو من يعاني من النتيجة) أن يضيف اسم العميل أيضاً إلى الجدول، على الرغم من أن ذلك يخالف شرط الشكل السوي الثالث (كنا قد تخلصنا من اسم العميل في جدول الفواتير في الحلقة السابقة عند تطبيق قيد الشرط السوي الثالث). السبب في هذا تجنب الربط المستمر بين جدولي الفواتير والعملاء من أجل اسم العميل في الاستعلامات والتقارير (التقارير في النهاية مبنية على استعلامات) والنماذج (من ضمن ذلك نموذج الفواتير نفسه). لي بعض الملحوظات هاهنا، لكن دعنا نؤجلها إلى ما بعد المثال الثاني.

من أجل ضرب المثال الثاني، ينبغي أن أعترف بأنني لم أستوفِ شرح شرط الشكل السوي الثاني. لقد ذكرت أن هذا الشرط ينص على أن الجدول يكون في الشكل السوي الثاني Second Normal Form - 2nd NF إذا كان في الشكل السوي الأول، وكل الأعمدة غير المفتاحية معتمدة تماماً على كامل المفتاح الأساسي. لقد ركزت في الشرح على الشطر الثاني من الشرط، وهو الاعتماد على (كامل) المفتاح الأساسي في حال كون المفتاح يتكون من أكثر من حقل. لكن الشطر الأول يقتضي أن يكون هناك اعتماد كامل للحقل غير المفتاحي على المفتاح الأساسي. هذا يعني بكل بساطة أن كل حقل يتحدد من مجرد معرفة المفتاح الأساسي. لقد ركزنا نحن في الماضي على جزئية اعتماد الحقل على كل أجزاء المفتاح الأساسي، لكن في البداية هناك جزئية أن يعتمد الحقل أصلاً اعتماداً كاملاً على المفتاح الأساسي، سواء كان هذا المفتاح مفرداً أو مركباً. مثلاً، حقل تاريخ الفاتورة يتحدد كلياً بمعرفة رقم الفاتورة، وكذلك الخصم الإجمالي على الفاتورة يحدده رقم الفاتورة من حيث أن الفاتورة الفلانية أعطي فيها الخصم الفلاني. قارن هذا بما يحدث من إضافة حقل محسوب إلى جدول الفواتير يحوي إجمالي الفاتورة. هذا الحقل في الحقيقة لا يعتمد على رقم الفاتورة، لأنك لا تستطيع معرفة إجمالي الفاتورة إلا بالرجوع إلى جدول تفاصيل الفواتير حيث الكميات والأسعار. هذا يعني أن إضافة هذا الحقل إلى جدول الفواتير يجعل الجدول خارج نطاق الشكل السوي الثاني، لأن حقل الإجمالي لا يمكن تحديده من المفتاح الأساسي حصرياً. لكن هذا يحدث كثيراً لاختصار عملية ربط جدولي الفواتير وتفاصيل الفواتير في حالة استعلامات ملخص الفواتير، وتوفير وقت إجراء العمليات الحسابية لاستخراج الإجمالي في الاستعلامات، وهو سبب لا يستهان به.

ما ينبغي أن أضيفه أيضاً أنك يجب ألا تتوقع أنك ستكسر القواعد ثم تمضي بفعلتك بسلام. كسر قواعد التسوية يعني التنازل عن التصميم المثالي لقاعدة البيانات، والتنازل عن التصميم المثالي يعني التعرض للمشاكل التي سبق أن أفضنا فيها. كل هذا يقود إلى حقيقة أن ما خسرته في تصميم قاعدة البيانات بسبب إزالة التسوية يجب أن تعوضه في البرمجة. ولهذا تزداد البرامج تعقيداً مع ازدياد البعد عن التصميم الجيد لقاعدة البيانات. مثلاً، في المثال الأول، يجب أن تتأكد من سلامة أسماء العملاء في جدول الفواتير من أجل التأكد من تكامل قاعدة البيانات. هذا يتم من خلال التطبيق، عوضاً عن برنامج إدارة قواعد البيانات. أيضاً بشكل أوضح في المثال الثاني، أمامك عبء برمجي كبير في التأكد من سلامة إجمالي الفاتورة في جدول الفواتير عند إدخال تفاصيل الفواتير في جدول تفاصيل الفواتير، وعند أي تعديل أو حذف أو إضافة لهذه التفاصيل فيما بعد، وهي ليست بالعملية البديهية. ليس هذا كل شيء، بل إنه في كثير من الأحيان يمكن تجاوز تعقيد عمليات الربط بين الجداول الكثيرة بإعداد استعلامات جاهزة مسبقة الترجمة تجمع هذه الجداول بعضها مع بعض، ثم تبنى بقية الاستعلامات والتقارير عليها. مثلاً، يمكن تجهيز استعلام يحوي كل تفاصيل الفواتير مع اسم العميل من جدول العملاء، ثم استخدام هذا الاستعلام في بقية الاستعلامات والتقارير. وبهذا أنت ترى الآن أن خيار إزالة التسوية لا يكون محبذاً في كثير من الأحيان. لكن يمكن أحياناً تجاوز القواعد الثقيلة لأسباب عملية. أنت الحكم، لكن لا تحكم إلا بعد أن تعرف جيداً ما تتكلم عنه.

أختم بمثال آخر على تجاوز القواعد في التصميم لأسباب عملية. أنا شخصياً أصبحت أستخدم كلمات (ذكر، أنثى) في حقل النوع عوضاً عن إنشاء جدول النوع واستخدام أرقام 1 و2 مثلاً لترميز النوع. السبب في ذلك أن النوع لا يتغير أبداً (فقط ذكر وأنثى بنفس الحروف)، لذلك ليست هناك مخاوف من شذوذ التعديل، ولا الحذف. أيضاً التوفير في مساحة الخزن قليل، ولا يعني شيئاً مع مساحات الخزن المتوفرة هذه الأيام. في المقابل، يتم بهذا تجنب ربط جدولين بشكل دائم. مثال يشبه هذا استخدام كلمتي (نقد وآجل) في حقل نوع الدفع في جدول الفواتير مثلاً بدون استخدام جدول خاص لترميز نوع الدفع.

كل ما مر معنا سابقاً يقود إلى استنتاج أن وجود أسماء بدلاً من أرقام في جداول أحد المبرمجين لا يعني بالضرورة أنه ليس.....! لكن الوضع قد يختلف معي ومعك، لذا يظل السؤال ملحاً: هل أنت.....؟

رقم الحلقة (22) تكامل قاعدة البيانات

عندما تشتري جهاز حاسوب جديداً، وتحصل على ضمان لمدة سنة أو اثنتين، أنت تعلم ما الذي سيقولونه لك: سوء استخدام! الجميع يعلم أن بناء جهاز أو نظام جيد لا يعني نهاية المطاف، وأن الاستخدام الرديء قد يحطم جودة النظام إلى ألف قطعة. من أجل ذلك شددت في حلقات سابقة على دور المستخدم. إن الوصول إلى تصميم جيد لقاعدة البيانات يعني أنك قد بدأت بداية صحيحة، ستكون على الأرجح سعيداً بنتائجها عند تطوير النظام وما بعده، لكنه لا يعني أبداً ضمان الاستخدام الصحيح للنظام.

دعني أتكلم بلغة عملية أكثر... أنا أعلم أنك قد وضعت رقم الفاتورة ليكون مفتاحاً أساسياً في جدول الفواتير، ولكن هل يعلم المستخدم ذلك؟ وأنا متأكد أن رقم الطالب في جدول العلامات هو مفتاح أجنبي يربط العلامة بطالب صحيح، لكن ماذا لو أدخل المستخدم رقم طالب غير صحيح؟ بالطبع أنت وضعت حقل التاريخ ليحوي تاريخاً مقبولاً، لكن ما هو التاريخ المقبول؟ ماهو عدد الإجازات المسموح به؟ ما هو مجال الأرقام المسموح به في حقل الخصم؟ هل من الممكن وجود تفصيل فاتورة بدون فاتورة؟ أدري، أنت لم تقصد ذلك، لكن منذ متى تمضي الحياة كما نريد؟

سلامة قاعدة البيانات من أمثال هذه المشاكل تسمى تكامل قاعدة البيانات، وهي ترجمة للمصطلح الإنجليزي Database Integrity الذي سيمر عليك كثيراً في عالم قواعد البيانات. يقسم الناس هذا التكامل إلى ثلاثة أنواع، من الممكن أن تجد لها كلها أمثلة في الفقرة السابقة:

1. تكامل كيان Entity Integrity
2. تكامل مجال Domain Integrity
3. تكامل مرجعي Referential Integrity

تكامل الكيان يتعلق بالمفتاح الأساسي: يجب أن يكون موجوداً، وألا يكون فارغاً، سواء كان حقلاً مفرداً أو مركباً من أكثر من حقل. أنت ترى أن وجود المفتاح الأساسي في جدول هو شرط لازم كما قال Codd، ولكنه ليس كافياً لضمان تكامل قاعدة البيانات إن كان سيترك فارغاً. من السهل أن تفرض هذا النوع من التكامل، لأن كل برامج DBMS تعمل ذلك من أجلك. فقط أخبر الأخ DBMS أن هذا الحقل (أو مجموعة الحقول) هو مفتاح أساسي، وسيحرص الأخ على عدم تركه فارغاً، لا داعي لأن تشغل دماغك بذلك (لكنك ستشغل دماغك بعد كل ذلك كما سنرى بإذن الله).

تكامل المجال يتعلق بمفهوم المجال، وهو مجموعة القيم التي يمكن أن يأخذها الحقل. هذا يشمل أمرين:

1. نوع بيانات الحقل كبداية، هل نقبل أرقاماً في هذا الحقل أم نصوصاً أم تاريخ... إلخ.
2. التشكيلات الصالحة من نوع البيانات هذا للحقل. مثلاً، ليست كل الأرقام صالحة لحقل العمر، على الرغم من أنه من نوع رقم، وليست كل التواريخ صالحة لحقل تاريخ الفاتورة (من نوع تاريخ)؛ لا يمكن تحرير فاتورة بتاريخ مستقبلي على سبيل المثال.

مراعاة تكامل المجال أصعب من تكامل الكيان، لأن برنامج DBMS لا يستطيع أن يغطي جميع الاحتياجات، على الرغم من أنه بالإمكان استخدامه في فرض قيود تتعلق بنوع البيانات المدخل ومدى القيم المسموح به إن كان معروفاً مسبقاً، ولا يعتمد على السياق. مثلاً، تستطيع تقييد قيم حقل النوع إلى ذكر وأنثى فقط كقيم صالحة، والتاريخ إلى قيمة أصغر أو تساوي تاريخ اليوم، والخصم إلى المدى (0 – 100%) إن كنت تسمح بذلك، لكن تظل هناك حالات لا تكون فيها القيود واضحة وبسيطة ومحددة بشكل مطلق. في هذه الحالات لا بد أن تضيف منطق القيود إلى التطبيق نفسه، بالإضافة إلى قيود DBMS.

التكامل المرجعي يتعلق بالمفاتيح الأجنبية. أنت تعلم أن المفتاح الأجنبي في جدول يحوي قيم مفتاح أساسي في جدول آخر، ويربط بذلك بين الجدولين. مثلاً، رقم العميل في جدول الفواتير هو مفتاح أجنبي يربط جدول العملاء بجدول الفواتير، وهو انعكاس لحقل رقم العميل (المفتاح الأساسي) في جدول العملاء. السؤال الآن: ماذا لو تم ترك هذا الحقل فارغاً في جدول الفواتير؟ الحق أن هذه ليس مشكلة دائماً، وقد يكون مقبولاً أحياناً (مثلاً، لا يهم العميل، لأن الفاتورة تم دفعها نقداً في محل تجزئة، على الرغم من أن بعض المحلات المعتبرة قد تصر على معرفة حركات عملائها من أجل الدراسات)، لكن المشكلة الحقيقية هي: ماذا لو تم إدخال رقم عميل غير موجود في جدول العملاء؟ ثم ماذا لو تم حذف عميل له فواتير، ألن تبقى فواتيره في جدول الفواتير معلقة بدون معرفة عميلها (تدعى هذه الفاتورة سجلات يتيمة لأنها بلا أب)؟ طيب، ماذا لو تم تعديل رقم العميل (حدث يجب ألا يحدث)، هل ستنشأ سجلات يتيمة أخرى في جدول الفواتير؟ إن فرض التكامل المرجعي يجنبنا أمثال هذه الأوضاع، ويضمن سلامة وصحة العلاقات بين الجداول في كل وقت. لحسن الحظ، تستطيع الاعتماد على DBMS في هذا النوع من التكامل بشرط تعريفه مسبقاً.

كما أشرت في فقرة سابقة، تسمى الشروط التي تفرضها من أجل الحفاظ على تكامل قاعدة البيانات (قيود التكامل Integrity Constraints)، لأنها تقيد الإدخال، وتمنع بعض التصرفات غير المرغوبة. هناك نوع آخر من القيود يسمى قواعد العمل Business Rules، لا تتعلق مباشرة بالمفاتيح الأساسية والأجنبية، وإن كان يمكن اعتبار قيود المجال جزءاً منها أحياناً. القيود السابقة قيود عامة تتفق في كل قواعد البيانات، لكن هناك متطلبات خاصة بكل قاعدة بيانات تفرضها حاجة العمل وسياسة المستخدم، ولذلك تسمى قواعد العمل. مثلاً، كم عدد المواد الممكن أن يحملها الطالب معه إلى المستوى الأعلى دون أن يفصل؟ ما هو الحد الأقصى لعدد التفاصيل في الفاتورة الواحدة؟ هل يمكن لموظف في إجازة أن يكلف بمهمة؟ وهكذا. من الواضح طبعاً أن قواعد العمل من اختصاص التطبيق، وتبرمج خارج قاعدة البيانات، وإن كان يمكن حسب نوع المنتج المستخدم دمج منطق قواعد العمل في قاعدة البيانات من خلال كائنات خاصة (مثلاً، قادحات الجداول Table Triggers في أوراكل). منطق قواعد العمل هو جزء التطبيق الذي يتناول فرض هذه القيود، وفي الأنظمة متعددة الطبقات (لو تذكر نقاشنا عن multi-layer architecture في أوائل الحلقات) ربما توضع في الطبقة الثانية مثلاً.

ذكرنا فيما سبق أن DBMS يمكن أن يساعدك في فرض التكامل عبر قيود التكامل، لكن ذلك لن يتم ما لم تخبره أنت بذلك (بطريقة صحيحة طبعاً). كيف يمكن أن تخبر DBMS بقيود التكامل؟ يختلف هذا من منتج إلى آخر، لكن بشكل عام، إذا كنت ستنشئ قاعدة بياناتك بلغة SQL، فإنه بالإمكان تحديد هذه القيود مع إنشاء الجداول بهذه اللغة (هل أنشأ أحدكم جداوله كاملة باستخدام SQL فقط؟ هذا مستبعد تماماً طبعاً في الأكسس، لكن في الأوراكل مثلاً...). منتجي برامج إدارة قواعد البيانات يوفرون هذه الأيام واجهات مرئية لإنشاء قاعدة البيانات وتحديد قيود التكامل، مثلاً في الأكسس، تستطيع فرض تكامل الكيان عبر تحديد حقل كمفتاح أساسي من نافة تصميم الجداول (المفتاح الأصفر الصغير في الزاوية)، كما تستطيع فرض التكامل المرجعي من نافذة العلاقات (اضغط بالزر اليمين على العلاقة بين جدولين من أجل فتح نافذة تحرير العلاقات، كما يمكنك تحديد عدد من قيود تكامل المجال في خيارات (خصائص الحقل) عند تصميم الجدول (قاعدة التحقق من الصحة Validation Rule). المهم ألا تنسى تعريف هذه القيود وأن تخطط لها جيداً. مثال على قيد مزعج وغير مدروس هو أن يتوقف المستخدم عن إدخال فاتورة لأن البرنامج يرفض تاريخ الفاتورة بيوم سابق بسبب تحديدك قيد التاريخ ليكون تاريخ اليوم فقط...

إذا وصلت إلى تصميم صحيح لقاعدة بياناتك، وفرضت قيود التكامل بشكل سليم، فإن الجزء الأهم والأكثر تأثيراً من عملك قد أنجزته. ما بقي من كتابة التطبيق ليس بالنزهة، لكنه لن يكون سباحة ضد التيار بإذن الله. سأتوقف هاهنا اليوم بإذن الله لأجعل هذه الحلقة قصيرة خفيفة على المعدة (لمن كان يتناول الشاي بالطبع). أظن أن كل هذا يجعلنا نتساءل من جديد: هل أنت.....؟

رقم الحلقة (23) مخطط الكائنات والعلاقات ERD: المفهوم

لنعد الآن إلى مخطط الكائنات والعلاقات... من المهم أن نتذكر أننا نتكلم عن مفهوم عام في تصميم قواعد البيانات العلائقية لا يرتبط بمنتج بعينه، ولذلك تستطيع أن تحكم بأن تحديده بكائنات الأكسس ليس صحيحاً، علاوة على أن المقصود بالكائنات في سياق المخطط مختلف عن المقصود بها في سياق الأكسس. ولكن، دعونا نقترب من الموضوع رويداً بنظرة الطائر...

لماذا المخططات؟ عندما يصادف الناس نظاماً صعب الفهم أو كثير التشعب، فإنهم يلجؤون عادة إلى تمثيل هذا النظام بنموذج منطقي مبسط، يمكن رسمه على الورق ودراسته، وإذا كان النظام يتكون من أكثر من جزء أو يحوي عدة جوانب، فإنه يمكن تكوين عدة نماذج models لنفس النظام، بحيث يعبر كل موديل عن أحد هذه الجوانب أو الأجزاء. هذه الموديلات تتميز بالتبسيط والتجريد والتركيز على جانب من النظام، بمعنى أنها تعالج النظام من منظور معين.

في عالم قواعد البيانات تتميز الأنظمة بصعوبة استيعابها دفعة واحدة، وباشتمالها على أكثر من جانب. مثلاً قاعدة البيانات تمثل البنية المنطقية للبيانات التي يحتاجها النظام، لكنها لا تعطي فكرة عن العمليات المختلفة التي تجرى على هذه البيانات... ذلك منظور آخر. هذا يعني أن المنظور الأول (منظور البيانات) قد يخبرك بوجود بيانات عن الفاتورة وبتفاصيل بنود الفاتورة وما الذي يعنيه كل بند، لكنه لن يخبرك عن العمليات التي سنقوم بها على الفاتورة (مثلاً، زيادة أو نقصان رصيد الأصناف والعملاء). لكن المنظور الثاني (منظور العمليات) سيكمل الصورة. تم ابتكار موديل لكل منظور، ولكل موديل طريقته ليس فقط في المبدأ والمفهوم، بل في الأشكال المستخدمة من أجل تمثيله على الورق. الذي يهمنا هنا هو منظور البيانات، لكنني أحب أن أوضح الصورة كاملة لأنني لمست أن هناك لبساً عند البعض فيما يتعلق بالموديلات والمخططات. من أجل منظور البيانات تم ابتكار موديل يسمى موديل الكائنات والعلاقات، لكنه اشتهر باسم المخطط المستخدم لتمثيله على الورق (أو الشاشة طبعاً)، وهو اسم مخطط الكائنات والعلاقات Entity-Relationship Diagram، واختصاراً ERD. هذا الموديل له فلسفته الخاصة التي نتكلم عنها بعد هنيهة بإذن الله. لاحظ أن هذا الموديل يقود إلى تصميم قاعدة البيانات، ولذلك يصح أن نقول أنه (أداة) من أجل تصميم قاعدة البيانات. بلغة هندسة البرمجيات يسمى هذا Semantic Data Model. في هندسة البرمجيات كذلك تسمى الموديلات التي تعبر عن النظام من منظور العمليات بـ Behavioural Models، أو موديلات التصرف (كيف يتصرف النظام). من أشهر أشكال هذه الموديلات مخطط تدفق البيانات Data Flow Diagram أو اختصاراً DFD. كما أسلفت، لكل موديل فلسفته وأشكاله. لكنني سأتناول بإذن الله في هذه الحلقة الأول منهما، وهو الخاص بمنظور البيانات.

سأسمح لنفسي هنا أن أقتبس من حلقة سابقة تكلمت فيها عن هذا المخطط، وأظن أن الكلام يستحق التكرار لأنه يضع الصورة كاملة، ويختصر مجموعة من المفاهيم التي مررنا عليها، وأظن أن إعادته هنا يخدم الحلقة، ويساعد على تجميع الأفكار المتناثرة:

"اسمح لي أن أقدّم بعض التوضيح لما نحن بصدده، وأرجو أن تتذكر هذا دائماً: قبل أن تخطو الخطوة ينبغي أن تدرك جيداً ما هو محلها من الإعراب، من أين تبدأ وإلى أين تنتهي. هذا الوضوح في الأفكار هو ما يميز ما نسميه بالمنهجية... لكي تبني قاعدة البيانات ينبغي أن تمر على مرحلتين، الأولى هي التحليل والثانية هي التصميم. لا تخلط بينهما. التحليل كما أسلفنا يعني جمع كل ما يمكن من فهم المتطلبات. هنا تشترك قاعدة البيانات مع التطبيق في الحاجة إلى التحليل، وفي مفهوم التحليل، وفي آليات التحليل (طرقه ووسائله لمن كان يكره المصطلحات السمجة). بعد أن تفهم النظام جيداً، تأتي مرحلة التصميم. بالنسبة لقواعد البيانات العلائقية، الهدف النهائي من التصميم هو تحويل الفهم الذي حصلت عليه في مرحلة التحليل إلى مجموعة من الجداول (السوية). ما معنى (سوية)، وكيف تصل إلى جداول (سوية)؟ هذا موضوع آخر في منتهى الأهمية، بل هو في قلب تصميم قواعد البيانات، لذا أنصحك أن تكون موجوداً عندما نتحدث عنه فيما بعد بإذن الله. في سبيل الوصول إلى هذا الهدف، هناك مجموعة من الأدوات التي قد تساعدك، من أبرزها وأكثرها انتشاراً مخطط الكائنات والعلاقات ERD (Entity Relationship Diagram).

عندما تصمم الجداول فأنت تصمم البنية المنطقية للبيانات في النظام، أنت تقول بعد التحليل: لقد فهمت المطلوب، وقد وجدت من هذا الفهم أنه لكي يعمل النظام بالشكل المرجو، فإنه يحتاج إلى كيت وكيت من البيانات، التي يمكن تقسيمها وتنظيمها بالشكل الفلاني، وهذا الشكل الفلاني هو تصميم قاعدة البيانات. هناك طريقان يتبعهما المصممون عادة. إما أن تبدأ مباشرة بتخطيط مجموعة من الجداول التي تمثل تصميم قاعدة البيانات، لكن يلزمك بعد هذا أن تقضي بقية الوقت في تطبيق قواعد التسوية على مجموعة الجداول هذه لتتأكد من أنها (سليمة). مرة أخرى، المقصود بالضبط من (سليمة) هاهنا سيكون بإذن الله واضحاً عندما نتحدث بتفصيل أكبر عن تسوية قواعد البيانات. الطريق الثاني هو أن تقضي الوقت في البداية تبني مخطط الكائنات و العلاقات ، وهذا ليس بالأمر السهل، لكنه الأكثر منهجية. إذا كانت خبرتك تمكنك من بناء مخطط سليم يعكس الواقع، فإنك تقريباً قد انتهيت؛ الباقي هو مجرد تطبيق قواعد آلية لتحويل المخطط إلى مجموعة من الجداول التي من المفترض أن تكون سوية بالفعل.

من هنا نفهم أن مخطط الكائنات والعلاقات أداة مهمة في تصميم قواعد البيانات، والتمكن منها يتطلب بعض الخبرة، والخبرة تعني الكثير من العمل. المبدأ وراء هذا المخطط واضح وسهل. أنت تميز في النظام الذي أمامك (مخزن، صيدلية، مدرسة، مستشفى، شركة...) مجموعة من الكائنات التي يمثل كل منها وحدة واحدة، قد تكون هذه الوحدة شخصاً أو شيئاً أو مفهوماً ما. مثلاً، بعض الكائنات التي قد تميزها في الأنظمة السابقة: طالب، مريض، موظف (أشخاص)، مادة أو صنف، فصل، عيادة (أشياء)، علامة، إجازة (مفهوم)، وهكذا... الصعوبة هاهنا هي في اكتشاف وتمييز الكائنات. الخطوة التالية هي اكتشاف وتحديد العلاقات بين هذه الكائنات (لا بد أن تكون هناك علاقات بينها لأنها تنتمي لنفس النظام). لاحظ أن كلاً من الكائنات والعلاقات قد يملك خصائصه الخاصة التي تصفه بدقة (الدقة هنا تعتمد على متطلبات النظام). هنا أيضاً صعوبة في اكتشاف العلاقات ومن ثم في تحديد نوعها. نعم العلاقات أنواع كما لا بد قد سمعت من قبل. بل إن هناك أنواع لأنواع العلاقات. المزيد فيما بعد بإذن الله.

على الرغم من أن مخطط الكائنات والعلاقات موضوع يدرس في علم قواعد البيانات، وله طرقه وقوانينه، إلا أنه كما أشرت من قبل يحتاج إلى بعض الخبرة للتمكن منه. الصعوبات التي قد تواجه المبتدئ تشمل، كما سبق أن أشرت، الاكتشاف الصحيح للكائنات والعلاقات والتفريق السليم بين الكائن وخاصية الكائن (هل الهاتف كائن بحد ذاته أم أنه خاصية من خصائص الموظف؟)، وكذلك الاختيار بين الكائنات والعلاقات (الفاتورة كائن أم علاقة بين كائنين؟). وبالرغم من هذه الصعوبة، إلا أن العائد كبير، وهو تصميم سليم للجداول، وتعميق لفهم النظام، وخبرة لا تقدر بثمن في تصميم قواعد البيانات.

هذا يحدد الطريق أمامنا بعض الشيء فيما يتعلق بقواعد البيانات. علينا أن نحلل النظام لنفهمه جيداً، وهذا يستلزم جمع قدر كاف من البيانات بمختلف أشكالها. ثم علينا أن نستخدم هذا الفهم في إنشاء مخطط للكائنات والعلاقات (الطريق الأكثر منهجية)، ثم يلزمنا أن نحول هذا المخطط إلى مجموعة من الجداول. هذا يلخص تحليل وتصميم قواعد البيانات. طبعاً من الممكن اختبار التصميم بتطبيق قواعد التسوية، وبعد ذلك يأتي دور إكمال التصميم بقواعد لضمان تكامل وسلامة القاعدة وبياناتها، وغيرها من المكملات التي تختلف من نظام إلى آخر."

كما ترى، لقد مررنا حقاً على أكثر المفاهيم الملخصة في تلك الحلقة (التسوية، العلاقات، التكامل)، وبقي أن نوضح مخطط الكائنات والعلاقات قليلاً مع المثال... اعذرونا على التكرار، لكن صدقوني، أن تقرأ الفكرة مرتين بعبارتين خير من مرة واحدة.

يقوم موديل الكائنات والعلاقات (كما يدل اسمه، وهذا غريب) على استيعاب الحياة العملية كمجموعة من الكائنات (كل كائن له مجموعة من الخصائص) بينها مجموعة من العلاقات. هذه المجموعات من الكائنات (وخصائصها) مع ما بينها من علاقات ستمثل كل البيانات في النظام، وأسميناها قبل ذلك بالبنية المنطقية للبيانات. من هناك، ستنطلق بكل يسر لتكون مجموعة من الجداول التي تمثل قاعدة البيانات. كل هذا يعني أنك إذا اخترت استخدام هذا الموديل ليساعدك في تصميم قاعدة البيانات (خيار جيد) فإن مهمتك تكمن في النظر بعناية إلى النظام أمامك، واستخراج كل الكائنات التي تستطيع تمييزها في النظام، مع ما بينها من علاقات (مهمة ليست بالسهلة أبداً، صدقني). ربما لاحظت في العبارة السابقة أن استخدام هذا الموديل ليس محكوماً بقواعد محددة، يتبعها الجميع فيصلون إلى نفس النتيجة. ذكرت في تلك العبارة أكثر من كلمة تدل على هذا المعنى: (النظر بعناية)، ليس كل المصممين متساوين في النظر (طبعاً لا أقصد النظر الذي نستخدم له النظارات)؛ كذلك كلمة (تستطيع تمييزها)، ليس كل المصممين متساوين في الاستطاعة. هنا تكمن صعوبة هذا الموديل: إجادته تحتاج إلى خبرة، والخبرة تحتاج إلى (احزر): التمرين. لكن الثمرة عظيمة، وهي وصولك إلى تمثيل صحيح لبنية البيانات في شكل مخطط (ERD)، وهذا يعني أنك عملياً قد وصلت إلى تصميم مجموعة من الجداول في الشكل السوي الثالث. سنتكلم بإذن الله عن كيفية تحويل المخطط إلى جداول، لكن في البداية نحتاج إلى التعرف أكثر على طبيعة هذا المخطط.

السؤال الذي يطرح نفسه في البداية: ما هي طبيعة هذه الكائنات التي علي أن أميزها؟ ما هو الكائن؟ العادة أن يعرف الكائن بأنه (شيء) في الحياة العملية يمكن تمييزه عن بقية الأشياء. يمكن تمييزه لأن له مجموعة مستقلة من الخصائص أو يدخل في تفاعلات مع بقية الأشياء. هذا الشيء قد يكون مادياً أو معنوياً. قد يكون شخصاً مثل الطالب والموظف والزبون والمورد، أو مستنداً كالفاتورة والسند المالي، أو فكرة ومفهوماً مثل الإجازة والترقية. هذا الشيء لا بد أن يملك بعض الخصائص التي تصفه وتميزه. مثلاً، كائن مثل الفاتورة له خصائص مثل الرقم والتاريخ وطريقة الدفع. كائن مثل الصنف له خصائص مثل الرمز والاسم ورقم الرف. كائن مثل العميل له خصائص مثل الرقم والاسم والجوال والرصيد. هنا تكمن صعوبة: الخلط بين الكائنات وخصائص الكائنات، بمعنى استخراج كائن هو في الحقيقة خاصية لكائن آخر، وبالعكس تحديد خاصية لكائن، في حين أنها في الحقيقة ينبغي أن تكون كائناً مستقلاً. الصعوبة هنا أن حل هذا الإشكال ليس له إلا قواعد عامة، والكلمة الأخيرة فيه للخبرة.

أمثلة على ذلك؟ هل العنوان خاصية أم كائن؟ في الحالات الاعتيادية يكون العنوان خاصية من خصائص كائن آخر مثل الطالب مثلاً، لأن كل طالب يتميز بعنوان مستقل، ولن يتكرر عنوان إلا بين أخوين مثلاً، كما أن العنوان بحد

ذاته لا يلعب أي دور ذا أهمية خاصة غير صفة أخرى من صفات الطالب. لكن كيف يكون الوضع لو كنا نتكلم عن طلاب في داخلية، حيث يشترك الكثير من الطلاب في نفس العنوان، كما أن مساكن الجامعة قد تعامل معاملة مستقلة من نوع ما؟ أيضاً قد يكون لكل موظف هاتف، لكن قد تخصص الشركة هاتفاً واحداً في غرفة محددة مع متابعة من نوع ما لتكلفة الاتصالات لمجموعة من الموظفين، فيصبح الهاتف كائناً بعد أن كان مجرد خاصية. بشكل عام، من الصعب الخلط بين الكائن والخاصية، لكن الأمر يختلف حين نصل إلى العلاقات، الخلط بين العلاقات والكائنات أكثر سهولة.

كيف أميز العلاقة؟ العلاقة بشكل عام هي نوع من التفاعل بين كائنين، وفي الغالب تتضمن حركة من نوع ما. مثلاً، لماذا لم أذكر السعر والكمية والصنف من خصائص الفاتورة؟ لأن الحقيقة أن تفاصيل الفاتورة من أمثال الصنف المشترى أو المباع، وسعر الوحدة، والكمية، والخصم على الوحدة، ووحدة القياس، كلها عبارة عن علاقة بين كائن الفاتورة وكائن الصنف. لماذا؟ لأن هذه التفاصيل لن توجد ما لم تنشأ حركة ما بين الفاتورة وبين الصنف، وهو أن يشتري عميل (أو نبيع لعميل) هذا الصنف في هذه الفاتورة، وعندئذ يتولد السعر والكمية وما إلى ذلك. بعبارة أخرى، لا سعر ولا كمية صنف بدون فاتورة، ولا سعر أو كمية لفاتورة بدون الارتباط بصنف. إذا أدركت هذه الحقيقة تجاوزت التصميم الخاطئ الذي ذكرناه في دروس التسوية، ووصلت مباشرة إلى التصميم ذي الثلاث جداول (فاتورة، صنف، تفاصيل فاتورة). لاحظ أن الكمية والسعر والخصم والوحدة هي خصائص للعلاقة، ومن هنا ندرك أن العلاقة قد يكون لها خصائص مثل الكائن.

على ذكر الخصائص، دعنا نذكر عنها بعض التفصيل الذي قد تصادفه في الكتب (وفي الحياة العملية بالطبع). تقسم الخصائص من زوايا مختلفة إلى:

1. بسيطة ومركبة.
2. مفردة ومتعددة القيم.
3. أصيلة ومشتقة.

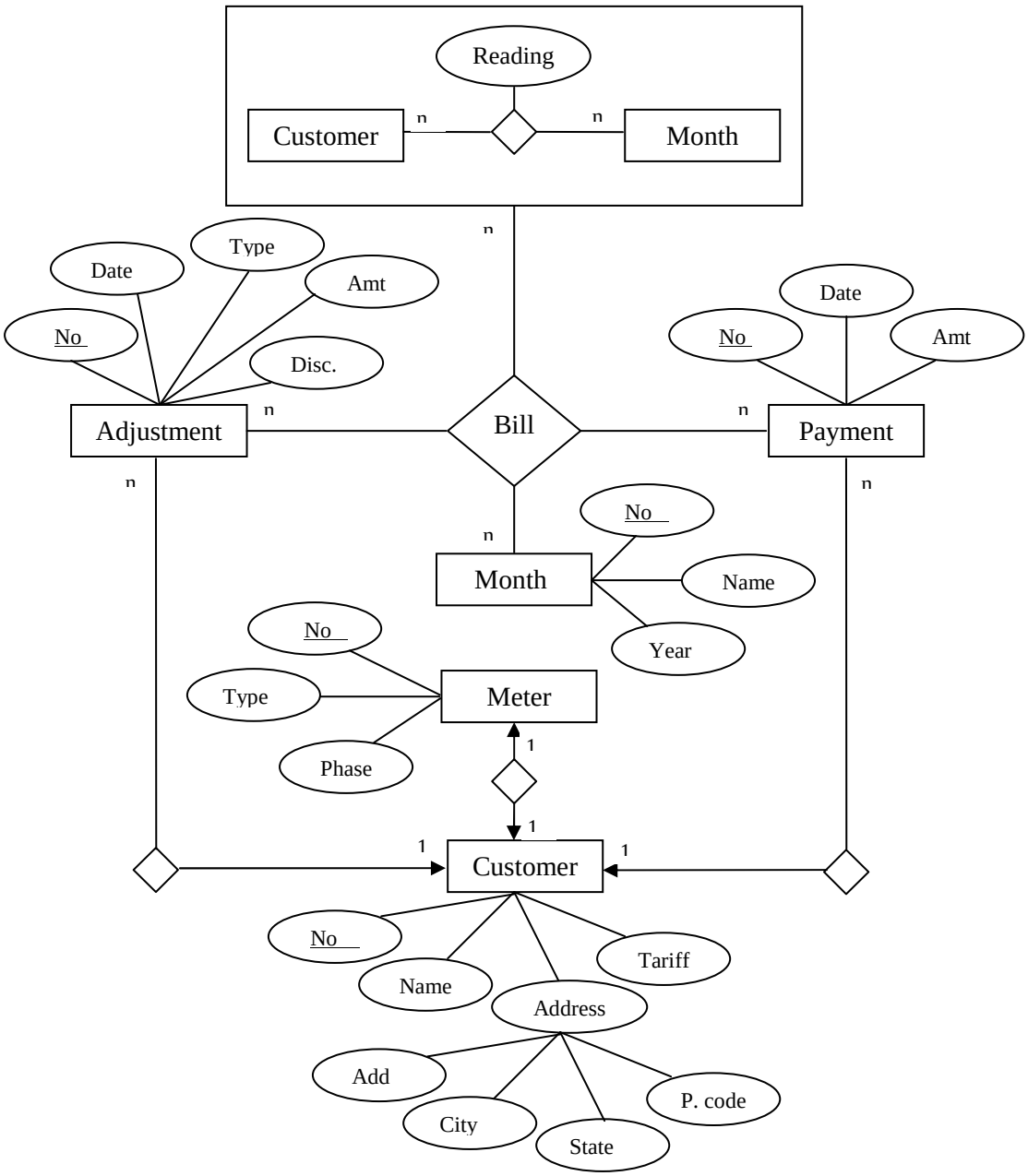
التقسيم الأول يعني أن الخاصية قد تكون من جزء واحد مثل رقم الطالب أو اسمه الكامل، وقد تكون مركبة من أكثر من جزء مثل العنوان (الذي قد يشمل تفاصيل مثل المدينة والبلد والرمز البريدي وغيرها)، والاسم إذا تم تفصيله إلى الاسم الأول واسم الأب والعائلة وخلافه.

التقسيم الثاني يعني أن الخاصية قد تكون لها قيمة واحدة مثل الاسم (لكل عميل اسم واحد فقط سواء كان بسيطاً أو مركباً) أو قد يكون للخاصية أكثر من قيمة (مثل رقم الهاتف، قد يكون للعميل أكثر من رقم هاتف).

التقسيم الثالث يعني أن الخاصية قد تقدم معلومة جديدة (الأغلب)، وقد تكون عبارة عن قيمة محسوبة من خصائص أخرى. لكل نوع من هذه الأنواع طريقة خاصة للرسم في مخطط الكائنات والعلاقات، لكن هذا فيما بعد بإذن الله.

الخلاصة إلى هنا أن مخطط الكائنات والعلاقات هو أداة لتصميم قواعد بيانات النظام، لكنه لا يتناول تمثيل وظائف النظام التي تجدها عادة في التطبيق وليس في قاعدة البيانات. وهناك مخططات أخرى تتناول النظام من حيث وظائفه وعملياته.

بقي أن نشير إلى أشكال الرسم المستخدمة في المخطط، وبعض الأمثلة، وربما بعض الزيادات التي أضيفت إلى الفكرة الأصلية للمخطط. سأترك ذلك بإذن الله للحلقة القادمة، لكن سأختم بمثال أستبق به موضوع الحلقة القادمة، وأتركك أخي مع محاولاتك لفك طلاسم هذا المثال، فإن فعلت فإنك إذاً من النابهين، وإن لم تستطع فلا تثريب عليك، بل إن هذا هو الطبيعي، لكنك ستكون عندئذ على أتم استعداد لاستيعاب الدرس القادم بإذن الله... هذا المثال هو مخطط لنظام فواتير كهرباء مبسط (لكنه عملي)، والهدف منه إعطاؤك صورة شاملة عما يعنيه مخطط الكائنات والعلاقات، وتمهيدك للحلقة القادمة:

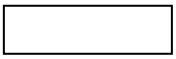
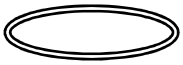


لا تأخذ الأمر شخصياً، ولكن بودي أن أعرف: هل أنت..... ؟

رقم الحلقة (24) مخطط الكائنات والعلاقات ERD: ما بعد المفهوم

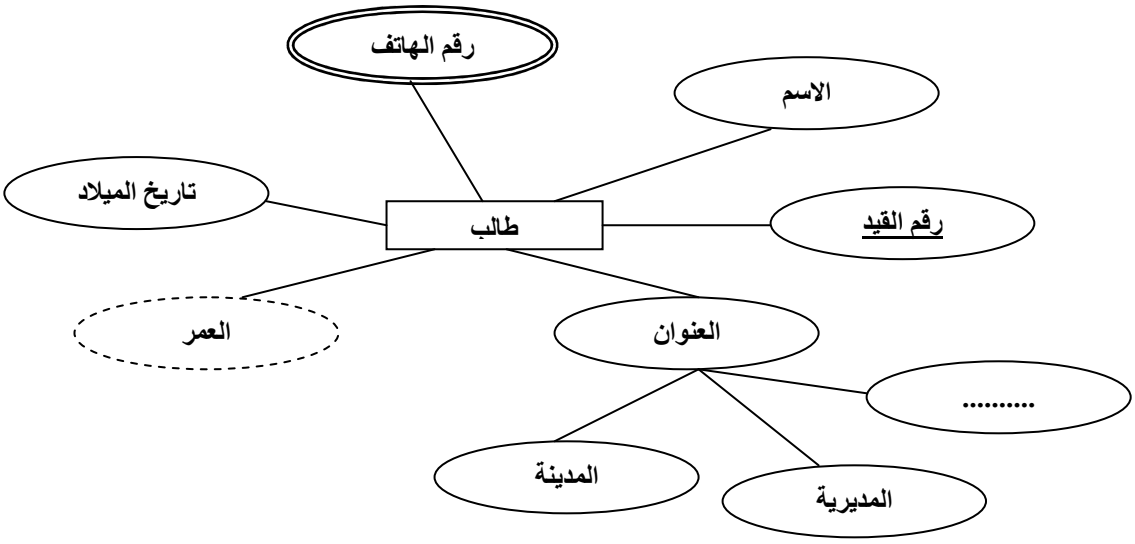
عرفنا مفهوم مخطط الكائنات والعلاقات، وحان الوقت لنتعرف أكثر على أهم أشكاله واصطلاحاته. الملخص في مفهوم المخطط أنه مخطط يمثل طريقة للنظر إلى النظام في الحياة العملية. هذه الطريقة هي النظر إلى النظام كأنه مجموعة من الكائنات التي يملك كل منها خصائص مميزة، ويتفاعل كل منها مع الآخر بعلاقات. العلاقات نفسها قد تملك خصائص تميزها بدورها. مثلاً، كائن الفاتورة يتفاعل مع كائن الصنف عبر علاقة هي ظهور هذا الصنف في تفاصيل هذه الفاتورة. هذه العلاقة تملك خصائصها المميزة، مثل سعر الوحدة، والكمية ووحدة القياس، وربما الخصم. لاحظ كيف أن الكمية المباعة مثلاً ليست من خصائص الصنف، ولا من خصائص الفاتورة، لكنها من خصائص العلاقة ناتج التفاعل بين فاتورة وصنف.

قبل أن أقدم بعض اصطلاحات المخطط، دعونا أولاً نتعرف على الأشكال الأساسية في مخطط الكائنات والعلاقات:

الشكل	رسمه	يستخدم لتمثيل
مستطيل		مجموعة كائن Entity set
بيضاوي		خاصية Attribute
معين		مجموعة علاقة Relationship set
خط		اتصال بين الأشكال اسابقة Link
بيضاوي مزدوج		خاصية متعددة القيم Multi-valued attribute
بيضاوي متقطع		خاصية مشتقة (محسوبة) Derived attribute

كل هذه الأشكال قد ذكرت تعريف ما تمثله في الحلقة السابقة. دعنا الآن نركز على نقطة صغيرة في الجدول السابق ربما تكون قد أزعجتك (إذا لاحظتها أصلاً بالطبع). لماذا أشرنا إلى المستطيل بـ(مجموعة) كائن؟ هل هناك فرق بين (كائن) و(مجموعة كائن)؟ الحق أن الموضوع لا يستحق القلق. المستطيل يرمز إلى الكائن في مخطط الكائنات والعلاقات، لكن هل قابلت يوماً نظاماً للمبيعات يحوي فاتورة واحدة فقط؟ هذا هو المقصود. يمثل المستطيل كل الكائنات من نفس النوع لأنه دوماً تكون هناك مجموعة من الكائنات من نفس النوع وليس حالة واحدة فقط. مثل كائن الطالب في نظام تسجيل جامعة أو مدرسة، يمثل مستطيل الطالب كل مجموعة الطلاب في المدرسة، ولذلك نقول عن مستطيل الطالب أنه يمثل مجموعة كائن. الأمر بسيط كما ترى. ونفس الكلام يقال عن مجموعة علاقة. هذا يشبه تماماً (في الحقيقة هو نتيجة له) ما نقوله عن جدول الطلاب مثلاً. يحوي جدول الطلاب أكثر من سجل من نفس النوع (نفس الأعمدة) كل منها يمثل طالباً، لذلك يمثل الجدول مجموعة.

إذا عرفنا أن المستطيل يمثل في الحقيقة مجموعة من الكائنات المفردة من نفس النوع (تشترك في نفس الخصائص) مثل مجموعة الفواتير مثلاً في النظام، فإن هذا يقودنا إلى ضرورة التمييز بينها، وهذا بدوره يقودنا مباشرة إلى مفهوم المفاتيح (الأساسية). لن أكرر الكلام هنا عن مفهوم المفاتيح، لأن ذلك سيجعل طعم الشاي مراً في الفم بعد كل ما قلناه عنه، لكن الذي يعنينا هنا أننا نميز الخاصية (أو مجموعة الخصائص) التي اخترناها كمفتاح بخط سفلي underline. هذا مثال لكائن الطالب مع رقم القيد كمفتاح:



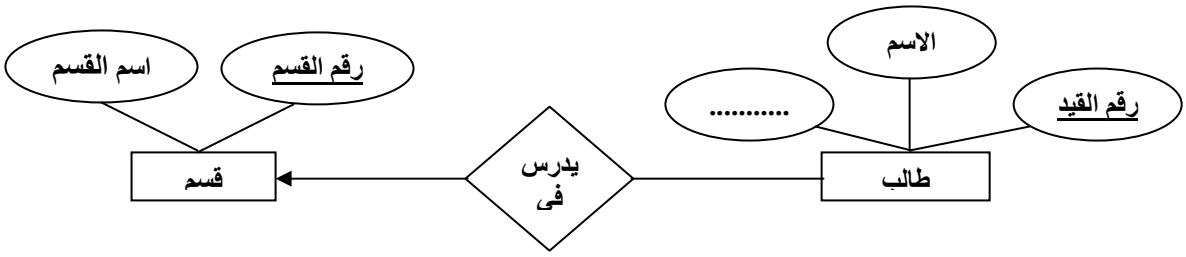
لاحظ من فضلك أن رقم الهاتف هو خاصية متعددة القيم، لأن الطالب قد يملك أكثر من رقم هاتف. لا تفكر الآن في الجداول، لأنك لو فكرت في ذلك لقلت: أليس من شروط الشكل السوي الأول أن تكون قيم الحقول ذرية atomic؟ حين عرفنا الذرية، قلنا إن ذلك يعني أن تكون قيمة الحقل مفردة، وهذا يتعارض تماماً مع مفهوم الخاصية متعددة القيم. كل هذا صحيح في سياقه الصحيح، وهو سياق الجداول. لكننا هنا في سياق مخطط الكائنات والعلاقات. عندما يأتي الوقت لتحويل هذا المخطط إلى جداول، ستعلم أن الخاصية متعددة القيم لن تصبح حقلاً في جدول.

لاحظ كذلك أن خاصية العنوان هي خاصية مركبة. هذا يعني أنها تنقسم إلى أكثر من خاصية، ولا يعني أنها تملك أكثر من قيمة، لذا لا تخلط رجاءً بين الخصائص متعددة القيم، والخصائص المركبة. هذا يعني أن للطالب عنوان واحد فقط (قيمة مفردة) ولكنها تتكون من أكثر من جزء. عندما يحين وقت تحويل المخطط إلى جدول، سنرى كيف أن هذه الخاصية قد تصبح عدداً من الحقول في نفس الجدول. ربما كان من المفيد هنا أن نشير إلى ملحوظة عملية تتعلق بالعنوان. في البلدان التي تم تخطيطها قبل بنائها (هذا يعني أن هناك بلداناً بنيت بدون تخطيط، أظن أننا نعرف هذا جيداً)، تكون أجزاء العنوان شبه ثابتة ومنظمة: رقم الشقة والبلوك (أو الفيلا)، اسم ورقم الشارع، المدينة والرمز البريدي والولاية أو المحافظة. إذا لم يكن هذا التنظيم معلوماً في العنوان فإنه من الأفضل تخصيص خاصية أو أكثر لتفاصيل العنوان (بعضهم يجعله بشكل: سطر1 وسطر2)، ثم الأجزاء الواضحة مثل المديرية أو المحافظة أو المدينة. القصد أن تحاول قدر الإمكان فصل الأجزاء الواضحة والمشتركة من العنوان في خصائص (حقول فيما بعد) منفصلة، حتى تتيح لاحقاً إمكانية تصنيف الكائنات حسب جزء محدد (مثلاً، استعلام أو تقرير عن الطلاب من مدينة ريفية).

خاصية العمر كما هو واضح خاصية مشتقة، أي يمكن حسابها من خصائص أخرى (من حقل تاريخ الميلاد هنا). نحن نتكلم هنا عن مخطط الكائنات وخصائصها، فلا غبار على الخصائص المحسوبة. لكن كما علمنا من قبل عند مناقشتنا لتصميم الجداول وأشكالها السوية، فإن موضوع الحقول المحسوبة يحتاج إلى مزيد تأمل. هل نحتاج إلى تضمين الحقول المحسوبة في الجداول؟ هذا سؤال ليست له إجابة واحدة. كقاعدة عامة، ينبغي تجنب حفظ قيم الحقول المحسوبة في الجداول، لأن ذلك يكسر قواعد التسوية (الشكل السوي الثاني أو الثالث حسب الحالة). من الواضح طبعاً أن قيم الحقول المحسوبة تمثل تكراراً، لأنه يمكن دائماً حسابها من بيانات موجودة أصلاً في قاعدة البيانات. مثلاً، بإمكانك حساب مقدار التأخير من حقل وقت الوصول. لكن الموضوع كما ذكرنا مراراً في السابق ليس قضية مساحة مهدرة، بقدر ما هو موضوع أخطاء قد تتسرب إلى قاعدة البيانات. مثلاً، إجمالي فاتورة هو حقل محسوب، وحفظه في قاعدة البيانات قد يعرضنا إلى خطر الاعتماد على إجمالي خاطئ في حالة تغيير بعض تفاصيل الفاتورة بشكل ما (حذف صنف، إضافة صنف، تغيير كمية أو سعر... إلخ)، مع إهمال تحديث الإجمالي المحفوظ مسبقاً. مثال العمر هنا أسوأ، لأن العمر يتغير باستمرار مع تقدم السنين، ويمكن دائماً طرح تاريخ الميلاد من التاريخ الحالي لمعرفة العمر. قد يقول قائل: لكن المقصود هنا عمر الطالب عند التسجيل، وليس عمره الحالي. لكن الواقع أنه دائماً ما تكون هناك حقول تدل على تاريخ التسجيل بشكل أو بآخر، فيمكن حساب العمر وقتها من تاريخ التسجيل مع تاريخ الميلاد. متى نحتاج إذاً إلى حفظ الحقول المحسوبة؟ هذا يعتمد من مبرمج إلى آخر. أقول مبرمج وليس مصمماً، على الرغم من أن هذا من قرارات تصميم الجداول، وهو من مسؤوليات المصمم، لأن المصمم عادة سيختار اتباع قواعد التصميم الصارمة ولن يضمّن أي حقول محسوبة. لكن المبرمج الذي سينشئ الاستعلامات والتقارير فيما بعد، هو الذي سيعاني من استعلامات معقدة وكبيرة لأنها تحوي الربط بين أكثر من جدول لاستخراج القيم المحسوبة، أو سيعاني من استعلامات بطيئة بسبب الوقت المستهلك في إجراء الحسابات إذا كانت الاستعلامات كبيرة؛ لذلك قد يختار المبرمج تضمين الحقول المحسوبة ابتداءً من أجل تجنب هذه الصعوبات إذا كان له حق تغيير التصميم (نحن نعرف أننا نملك غالباً هذا الحق، لأننا نلعب دور المصمم والمبرمج معاً). أقول (قد)، وأقول إن هذا يختلف من مبرمج إلى آخر، لأن هناك ثمن ليس باليسير يجب أن يدفع

مقابل ذلك. هذا الثمن هو الانتباه إلى تحديث الحقول المحسوبة باستمرار لتعكس بالفعل واقع القيم في الحقول التي تدخل في الحساب. وهذا الانتباه ليس سهلاً في الواقع. مثلاً، قد ينتبه المبرمج إلى حالة حذف صف أو إضافة صنف إلى الفاتورة، فيقوم بتعديل الإجمالي في الحدث المناسب، لكنه قد يغفل عن حالة تعديل كمية صنف موجود سابقاً في الفاتورة، أظن -إن كنت على اطلاع- أنك قد صادفت مشاكل من هذا النوع في بعض البرامج.

من المهم جداً أن تلاحظ كذلك أننا لم نضمّن أي خاصية تدل على القسم الذي ينتمي إليه الطالب في مخطط مجموعة كائن الطالب. السبب في ذلك، مرة أخرى، أننا لا نتكلم في سياق الجداول. ولذلك، لا نتكلم عن مفاتيح أجنبية تربط بين جدولين، مثل رقم القسم الذي يربط جدول الطلاب بجدول الأقسام. نحن هنا نتكلم عن كائنات منفصلة، والعلاقات بينها لا تمثل بمفاتيح أجنبية، بل بعلاقات لها أشكال خاصة. في مثالنا هذا، يمكن تمثيل كائني الطالب والقسم والعلاقة بينهما بالشكل التالي:

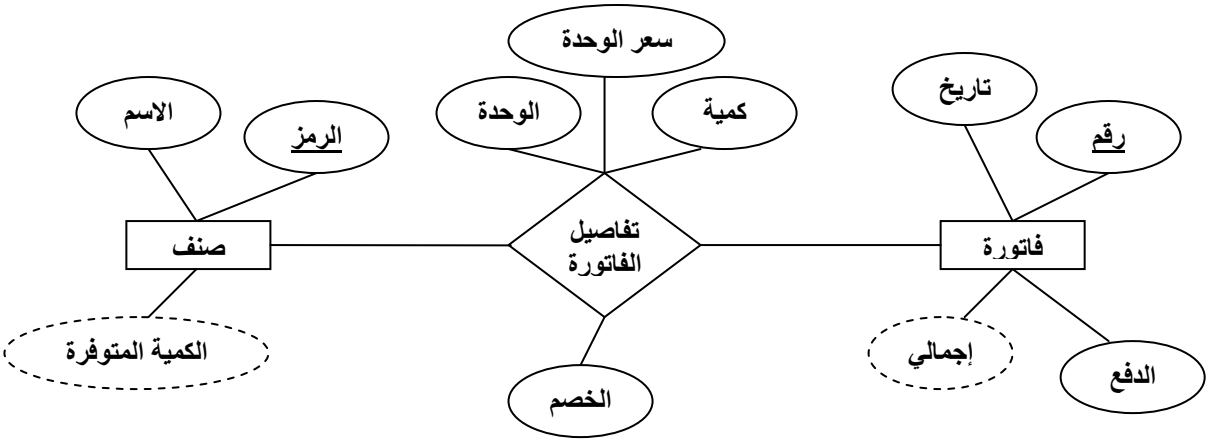


هنا ملحوظتان:

الأولى: أن للعلاقة اسم قد يعبر مباشرة عن نوع التفاعل بين مجموعتي الكائنين اللتين تربط بينهما. هنا، نوع التفاعل هو أن الطالب يدرس في هذا القسم. في حالة عدم وجود اسم مباشر واضح ومعبر، يمكن تكوين الاسم بلصق اسمي الكائنين مع شرطة في الوسط على سبيل المثال.

الثانية: أن العلاقات التي تكلمنا عنها سابقاً (واحد إلى واحد، واحد إلى متعدد، متعدد إلى متعدد)، يتم تمثيلها في المخطط بعدد من الطرق، واحد منها تمثيل جهة الواحد بسهم، وجهة المتعدد بخط دون رأس السهم. في المثال أعلاه، نفهم من المخطط أن العلاقة بين مجموعة كائن الطالب ومجموعة كائن القسم هي علاقة متعدد إلى واحد (من جهة القسم، العلاقة واحد إلى متعدد). هذا يعني أن الطالب لا يحق له الدراسة في أكثر من قسم، في حين أن القسم يحوي أكثر من طالب بالطبع. هناك طرق أخرى لتمثيل جهة المتعدد (مثل طريقة قدم الغراب، ولا تسألني لماذا اختاروا الغراب بالذات) وجهة الواحد (مثل رقم 1). مهما يكن، فإن المفهوم واحد.

على ذكر العلاقات، دعونا نر الآن العلاقة بين الفواتير والأصناف في مخطط ERD، وهو مثال مألوف لنا جميعاً على ما أظن...



بالتأكيد، الخصائص المبينة هي للمثال فحسب، وربما تجد خصائص أكثر في الأنظمة الحقيقية، لكن الفكرة كاملة. هنا يجدر أيضاً الإشارة إلى نقطتين:

الأولى: أننا نلاحظ أن للعلاقة نفسها (أسميناها تفاصيل الفاتورة) خصائص، مثل الكمية والسعر والخصم. هذه الخصائص تميز العلاقة نفسها (تفصيل الفاتورة الخاص بصنف محدد)، ولا تميز الصنف بعينه (لست تبيع هذا الصنف في كل الفواتير بنفس الكمية والسعر والخصم)، ولا الفاتورة (لا يجب أن تبيع بهذه التفاصيل في كل فاتورة).

الثانية: أنه ليس لمجموعة العلاقة مفتاح، لكن مفتاحها هو مجموع مفتاحي مجموعتي الكائنين اللتين تربط العلاقة بينهما، لذلك فإن مفتاح العلاقة (تفاصيل الفاتورة) هو مجموع رمز الصنف مع رقم الفاتورة.

لاحظ كذلك أن العلاقة هنا هي من نوع متعدد إلى متعدد، لأن الصنف الواحد قد يظهر في أكثر من فاتورة، وكذلك الفاتورة الواحدة قد تحوي أكثر من صنف.

ذكرنا إلى الآن العديد من مفاهيم واصطلاحات مخطط الكائنات والعلاقات ERD الرئيسة، لكن هناك المزيد مما لم أذكره. أسرد فيما يلي بعض المصطلحات التي قد تقرأها في المصادر، ولكنني أغفلتها لأنني لا أريد أن ألعب دور المرجع هاهنا، لكن يمكن للمهتم أن يبحث عنها:

Strong and weak entity set
Total and partial participation
Roles
Cardinality limits
Non-binary relationship sets
Extended features (specialization, generalization, aggregation)

بالمناسبة، قبل أن أنسى، ما رأيك بتمرين خفيف، وهو أن تكمل مخطط نظام الفواتير بكائنات العميل والسندات مع العلاقات الصحيحة مع المخطط المعطى في هذه الحلقة... ثم أريدك أن تفكر جدياً هذه المرة: هل أنت.....؟

رقم الحلقة (25) ERD إلى جداول

الهدف من تصميم قاعدة البيانات العلائقية هو الوصول إلى مجموعة من الجداول التي تقع على الأقل في الشكل السوي الثالث. سمعنا هذا من قبل (ومللناه). قلنا أيضاً إن تصميمك قد يمر بواحد من طريقين: إما أن تبدأ بوضع مجموعة أولية من الجداول (البعض قد يقترح عليك أن تبدأ بجدول واحد ضخم يشمل بطريقة ما كل بيانات النظام)، ثم تطبق على هذه الجداول قواعد التسوية، وإما أن تستخدم موديل الكائنات والعلاقات لتخطيط النظام على الورق عبر مخطط الكائنات والعلاقات ERD. أشرنا أيضاً إلى أن هذا الطريق الثاني فيه من الصعوبة ما يجعل الكثير من المبرمجين يتجنبونه، لكن هذه الصعوبة هي نسبية، تختلف من نظام إلى نظام، ومن مستوى خبرة إلى مستوى آخر. لكنني الآن سأكشف السبب الحقيقي الذي يجعل بعض المبرمجين يتجنب استخدام مخطط الكائنات والعلاقات، على الرغم من أن هذا السبب هو نفسه من أهم فوائد استخدام المخطط (طبعاً لا بد من وجود فوائد من استخدام المخطط، وإلا لم يكن هناك داعٍ أصلاً لابتكاره أو لاستخدامه، نحن هنا نناقش تصميم قواعد البيانات، وليس وسائل تعذيب الذات).

أشرنا في الحلقات الأولى إلى أن من أبرز الأخطاء التي نعاني منها، تجاهل التحليل والتصميم، والقفز مباشرة إلى طور التنفيذ. تصميم الجداول يحتاج أيضاً إلى تحليل، وفهم شامل للنظام. المشكلة في التحليل أنه ليس فقط يحتاج إلى وقت، ولكنه يحتاج إلى تفكير وخبرة، ولهذا كانت وظيفة (محلل أنظمة) من أهم وأعلى الوظائف قيمة في عالم تقنية المعلومات. أنا أركز على قضية التفكير كما قد يلاحظ المتابع للسلسلة، لأن جلوسك (أو قيامك وتجوالك في الغرفة كالمجنون) من أجل التفكير أثناء تحليل أو تصميم نظام، هو من أهم أسباب اكتساب الخبرة في التحليل والتصميم. هذا ما يسمى بالاحتكاك العملي، ومن دون هذا، يتحول العمل إلى نوع من (التجربة والخطأ trial and error). هذه الطريقة في العمل لا تنفع بتاتاً في تصميم الأنظمة....

إنشاء مخطط الكائنات والعلاقات يختبر فهمك الحقيقي للنظام. نقطة.

من أجل إنشاء مخطط الكائنات والعلاقات (الذي يعد من أدوات التصميم) ستكتشف أنك تحتاج إلى مراجعة التحليل عدداً من المرات، وأن هناك الكثير من النقاط التي تحتاج إلى استيضاح من صاحب النظام، وعدداً آخر من النقاط تحتاج إلى حسم حالاً لأن التصميم يعتمد عليها، وقد لا يكون هنك مجال للخطأ والتراجع عنها فيما بعد (إلا بثمن فادح). هذا يعني أن الصعوبة التي تواجهها في تصميم المخطط تستحق. الوصول إلى مخطط سليم يعني أن هذا المخطط يعكس حقيقة الواقع، وهذا بدوره يعني فهماً حقيقياً وشاملاً للواقع.

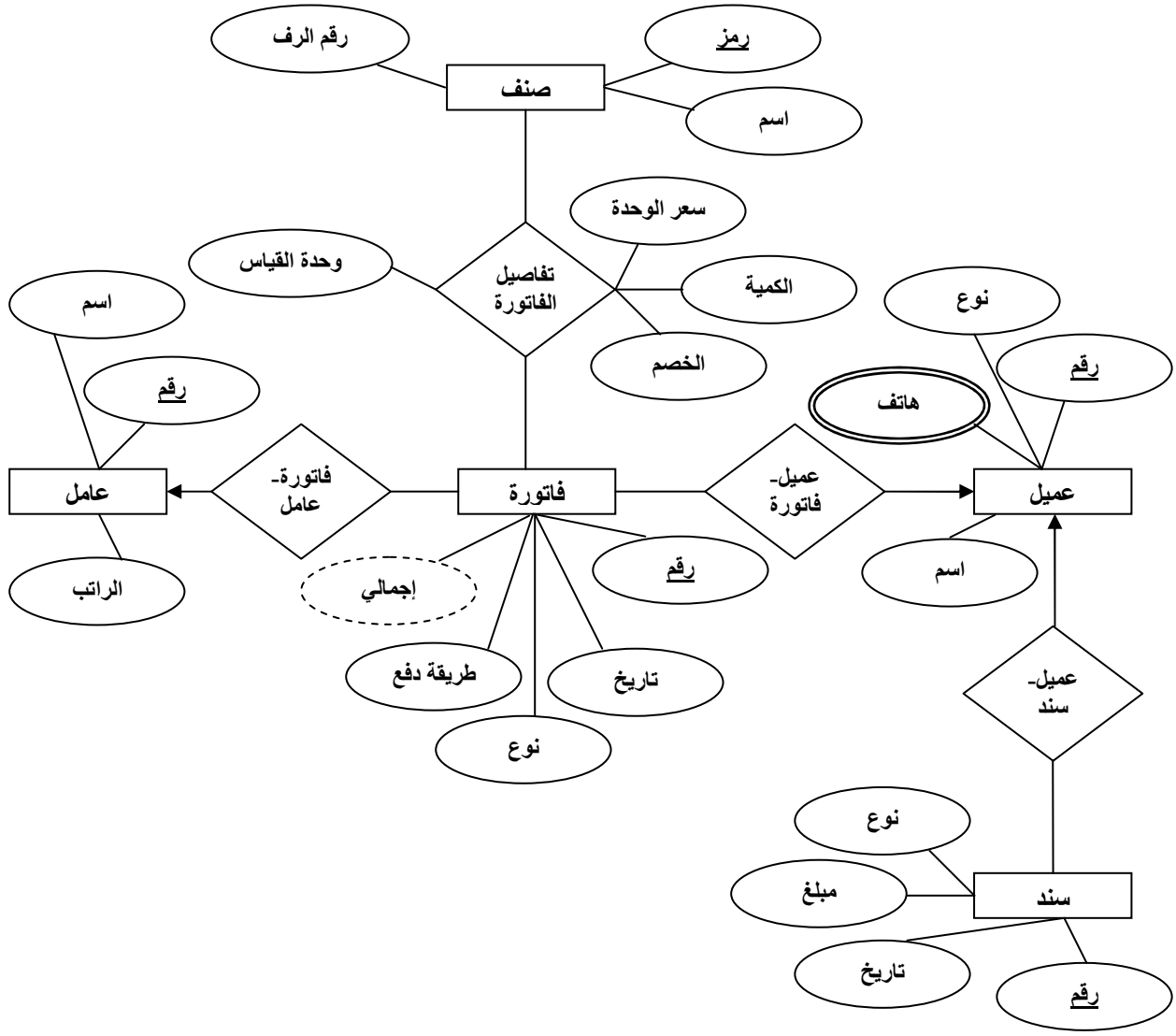
بعد الوصول إلى ERD سليم، تكتشف ثمرة جهدك: مجموعة من الجداول الواقعة في الشكل السوي الثالث. طبعاً أنت تحتاج إلى تحويل المخطط إلى هذه المجموعة من الجداول، لكن ذلك هو الجزء الأسهل، ويتم عبر قواعد شبه ثابتة، مما يعني أنك لست مضطراً إلى المزيد من التفكير العميق في هذه المرحلة (تنهيدة ارتياح). سنتناول هذا بإذن الله بعد قليل، لكن السؤال الذي أراه يتراقص في ذهنك الآن: هل أضمن الوصول إلى مخطط ER سليم حتى أضمن الوصول إلى تصميم سوي لقاعدة البيانات؟ لا عليك، في الوقت الذي تقضيه لاكتساب الخبرة في تصميم الـERDs، بإمكانك دائماً مراجعة تصميم الجداول عبر قواعد التسوية، وتنقيح الباقي، الذي تستطيع أن تتوقع أن يكون أقل بكثير مما لو بدأت من الصفر بدون المخطط.

ملحوظة: في الأنظمة الصغيرة جداً أو المألوفة جداً، يستطيع المصمم الخبير البدء مباشرة بمجموعة من الجداول السوية، لأنه غالباً قد مر بالطريق الطويل عدة مرات من قبل، ويحفظ الطريق جيداً.

تحويل المخطط إلى جداول مباشر إلى درجة كبيرة؛ من الممكن تلخيص أهم النقاط فيما يتعلق بالأشكال التي مررنا بها هنا في السلسلة بالتالي:

- الكائنات تصبح جداول.
- الخصائص المفردة تصبح حقولاً.
- كل جزء من الخصائص المركبة يتحول إلى حقل مستقل.
- الخصائص المشتقة يمكن الاستغناء عنها (راجع النقاش في الحلقة السابقة).
- الخاصية متعددة القيم تتحول إلى جدول مستقل مع إضافة حقل المفتاح الأساسي في الجدول الأصلي كمفتاح أجنبي.
- العلاقات تحتاج إلى تفصيل:
  - = العلاقات متعدد إلى متعدد تتحول إلى جدول مفتاحه هو مجموع مفتاحي الكائنين الذين تربط بينهما.
  - = العلاقات واحد إلى واحد تختفي ويتم دمج الكائنين الذين تربط بينهما في جدول واحد.
  - = إذا لم يتم دمج الجدولين تمثل العلاقة بتضمين المفتاح الأساسي لجدول في الجدول الآخر كمفتاح أجنبي.
  - = العلاقات واحد إلى متعدد غالباً ما تختفي وتمثل بتضمين مفتاح جهة الواحد في جدول جهة المتعدد كمفتاح أجنبي.

دعونا الآن نطبق هذا الكلام على المثال المألوف:



بافتراض أن المخطط السابق سليم، ويعكس حقيقة نظام مخازن ومبيعات أساسي (وهو كذلك بالفعل)، فإن مجموعة الجداول المتولدة هي كالتالي:

جدول الفواتير

رقم الفاتورة	تاريخ الفاتورة	نوع الفاتورة	طريقة الدفع	إجمالي الفاتورة	رقم العميل	رقم العامل

جدول الأصناف

رمز الصنف	اسم الصنف	رقم الرف

جدول تفاصيل الفواتير

رقم الفاتورة	رقم الصنف	الكمية	سعر الوحدة	وحدة القياس	الخصم

جدول العملاء

رقم العميل	اسم العميل	نوع العميل

جدول هواتف العملاء

رقم العميل	رقم الهاتف

جدول السندات

رقم السند	نوع السند	تاريخ السند	المبلغ	رقم العميل

جدول العمال

رقم العامل	اسم العامل	راتب العامل

لاحظ من فضلك أن مجموعة الحقول هنا مختصرة في كل جدول على خصائص أساسية أو معبرة، لكن المهم هو الفكرة، وإضافة المزيد من التفاصيل (الحقول) ليس صعباً، لكنه يختلف من متطلبات نظام إلى متطلبات نظام آخر، ويعود كما أشرنا إلى مرحلة (التفكير) إياها. اسمحوا لي أن أناقش هذا بمزيد من التفصيل من أجل توضيح الموضوع عملياً.

على سبيل المثال، في جدول الأصناف، من الممكن إضافة حقل محسوب (الكمية المتوفرة) يمثل رصيد الصنف الحالي، هذا كما علمنا يخضع لاختيار المبرمج. من المهم أيضاً التفكير في مسألة الكميات الموجودة من الأصناف في المخزن قبل بدء النظام. ربما اختار المصمم وضع حقل للكمية الافتتاحية في جدول الأصناف، وربما اختار إضافة فاتورة خاصة من أجل إدخال الكميات الافتتاحية من الأصناف (بشكل مشابه لموضوع القيد الافتتاحي في المحاسبة).

القرار السابق سيؤثر على جدول الفواتير، لأنه سيكون من الضروري وقتها توسيع أنواع الفاتورة لتكون (بيع، شراء، مردود بيع، مردود شراء، كمية افتتاحية). وعلى ذكر المردود، إذا كان من متطلبات النظام أن يكون مردود البيع مقيداً بفاتورة بيع، ومردود الشراء مقيداً بفاتورة شراء، فإنه يكون من المفيد إضافة حقل للفاتورة المصدر في حالة فواتير المردود.

هكذا ترى أن البداية هي التحليل والتفكير الكافي في متطلبات النظام، وبعدها التصميم لتلبية هذه المتطلبات. مخطط الكائنات والعلاقات هنا أداة مفيدة جداً، لكنها قد تتطلب بعض الوقت والجهد في البداية، ثم يتحسن الوضع مع اكتساب الخبرة إن شاء الله.

هذا قد يختم تقديمنا لموضوع مخطط الكائنات والعلاقات، وتبقى الكرة الآن في ملعب الإخوة القراء للإجابة عن السؤال العتيد: هل أنت.....؟

رقم الحلقة (26) تصميم جداول نظام مخازن ومبيعات: القليل من النقاش

(الجزء الأول)

إذاً، فلا مناص من التوقف، ومناقشة مثال عملي لتصميم برنامج قواعد بيانات. أنا أتوقع أن تحظى هذه الحلقة بالنصيب الأوفر من التفاعل لأننا نتحدث عن أفكار عملية مألوفة للكثيرين، على خلاف الحلقات السابقة التي كنا نتحدث فيها عن الكثير من المفاهيم، والناس لا تحب المفاهيم...

أيضاً أتوقع أن تطول هذه الحلقة قليلاً، لأنني ربما أسترسل في مناقشة بعض أفكار برامج المخازن والمبيعات، وربما من الأفضل حقاً أن تستعد من الآن بكوب من الشاي، وأن تتناول العشاء قبل أن تجلس، لأننا لا نريد أن يستدعوك من أجل العشاء ونحن نتحدث. إذا كنت مصراً، فأنا أحب الشاي سكر زيادة ☺

قبل أن نخوض في الكلام، دعونا نقف قليلاً، ونرتب أفكارنا، ونتفقد مواضع أقدامنا. أين نحن الآن بالضبط؟ تكلمنا حتى الآن عن العديد من المفاهيم في طريق تطوير أنظمة قواعد البيانات، ومن الجيد بين حين وآخر أن تتأكد من أنك لن تستيقظ فجأة لتتساءل في دهشة: أين أنا، من أنا؟ هناك الآن كم لا بأس به من المفاهيم يتراقص في السماء، اسمح لي أن أحاول التقاطها وسحبها إلى الأرض، لنرى أين تقع على أرض الواقع.

عند حديثنا عن برامج قواعد البيانات، عرفنا أن في طريقنا العديد من المحطات، منها محطتان رئيستان: قاعدة البيانات، والبرنامج (التطبيق) الذي تبنيه ويتعامل مع قاعدة البيانات. وهناك أيضاً المحطات الفرعية من أمثال تحليل قاعدة البيانات، تصميم قاعدة البيانات، تحليل وتصميم البرنامج، البناء الفعلي لقاعدة البيانات وللتطبيق، إلى آخره. ما نعتزم قضاء اللحظات القادمة فيه بإذن الله يصب أساساً في مرحلة تحليل وتصميم قاعدة البيانات، وهو أساساً ما تكلمنا عنه في هذه السلسلة حتى الآن. يحب الكثيرون أن يفكروا بتطوير برامج قواعد البيانات باعتبار التطبيق وليس قاعدة البيانات، ويسألون عن مفردات تنتمي إلى مرحلة بناء التطبيق. بما أننا ننوي الدخول في الجزء العملي، فأنا أحب أن أربط هنا المراحل السابقة بالواقع، وأن أجعل هذه المقدمة فعالة قدر الإمكان، وأظن أن أفضل وسيلة لذلك هي أن أجعلها في شكل الإجابة عن السؤال المألوف:

ماذا أحتاج لأصبح مبرمج قواعد بيانات؟

سؤال وجيه، ولكنه في كثير من الأحيان يأتي متأخراً للأسف، لكن لا بأس، الحضور متأخراً خير من عدم الحضور. سأجيب بإذن الله عن هذا السؤال في ظل ما تكلمنا عنه سابقاً حتى الآن، وفي ظل الافتراض الذي وضعناه من قبل، وهو أنك (الكل في الكل)، وافتراض خاص آخر، وهو أننا نتكلم عن أنظمة صغيرة، ولذلك سنستبعد المتطلبات من وزن (دراسة تحليل نظم) مثلاً، لأن ذلك بكل بساطة غير عملي في نطاق نقاشنا هذا.

بشكل عام، من أجل تطوير برنامج قاعدة بيانات، ستحتاج إلى تطوير قاعدة البيانات، وإلى تطوير البرنامج. لاحظ هنا، للمرة الألف، التمييز بين قاعدة البيانات وبين التطبيق أو البرنامج. من الممكن لقاعدة بيانات واحدة أن تخدم العديد من التطبيقات. قد لا تصادف ذلك في الأنظمة الصغيرة التي ستبنيها في البداية. لكن في الشركات، من الطبيعي جداً أن تخدم قاعدة بيانات واحدة كبيرة، عدة تطبيقات، مثل نظام المالية، والموارد البشرية، والمخازن، بالإضافة إلى نشاط الشركة الأساسي. كما أنه من الممكن، نظرياً على الأقل، لتطبيق واحد أن يتعامل مع أكثر من قاعدة بيانات.

من أجل تطوير قاعدة البيانات أنت تحتاج إلى تحليل متطلبات النظام من قاعدة البيانات، ثم إلى تصميم قاعدة البيانات وفقاً لتلك المتطلبات، ثم إلى بناء قاعدة البيانات فعلياً باستخدام واحد من برامج إدارة قواعد البيانات. من أجل تحليل المتطلبات أنت تحتاج إلى الإلمام بالطرق الأساسية لجمع البيانات، وإلى الوقت الكافي (والذكاء الكافي) لتطبيق هذه الطرق. أنا أتكلم عن أشياء من قبيل سؤال المستخدمين، دراسة مخرجات النظام، ومراقبة سير العمل. سبق الحديث عن هذه المواضيع، لكن لا بد هنا من التشديد على أن من أهم احتياجاتك لهذه المرحلة هو الكثير من الصبر. هذا الصبر سيتوزع على صبرك على المستخدمين، وعلى صبرك على نفسك من أجل إجبارها على قضاء وقت كافٍ في التفكير في كل ما جمعته، وفهمه جيداً، قبل القفز إلى المرحلة التالية. هل تشعر في نفسك القدرة على ذلك؟ جيد، إذاً، إلى الأمام.

من أجل تصميم قاعدة البيانات ستحتاج إلى ناتج المرحلة السابقة، وهو الفهم الجيد لمتطلبات قاعدة البيانات، وإلى الإلمام بالمفاهيم الأساسية لتصميم قواعد البيانات، والتي أخذت منا الكثير من الوقت حتى هذه اللحظة. هنا، أنا أتحدث عن أشياء من قبيل معنى الجداول والمفاتيح، والعلاقات بين الجداول، والأشكال السوية لجداول قواعد البيانات، ومتى يمكن أن تتنازل عن بعض التسوية (أقول بعض)، وتكامل قواعد البيانات. القدرة على إنشاء مخططات الكائنات والعلاقات تدخل ضمن المهارات التي قد تحتاجها في هذه المرحلة. هذه المفاهيم في هذه المرحلة بالذات لا يمكن التساهل فيها، ولا بد أن تتأكد من أن لديك الحد الأدنى من المعرفة بأصول تصميم قواعد البيانات. هل كنت قادراً على متابعة النقاش في هذه السلسلة، ولو بالحد الأدنى؟ إذا كان كذلك، فدعنا نمضي قدماً.

من أجل بناء قاعدة البيانات فعلياً، أنت ستختار (أو إن المستخدم سيختار) واحداً من أنظمة إدارة قواعد البيانات التي ستدير قاعدة البيانات. نحن نتكلم هنا عن أشياء من قبيل برنامج الأكسس والأوراكل. غالباً ستوفر هذه الأنظمة طريقة ما سهلة (بواجهة رسومية) لبناء قاعدة البيانات (مجموعة الجداول بكل حقولها وعلاقاتها، وغيرها من الكائنات التي لم نتكلم عنها بعد). لكن يمكنك دائماً أن تبني قاعدة بياناتك باستخدام اللغة الوحيدة التي يجيدها برنامج إدارة قواعد البيانات: SQL. هذا الخيار الأخير نادر الاستخدام في حالة استخدام نظام مثل الأكسس، لكنه وارد جداً في حالة استخدام أنظمة مثل الأوراكل. هذا يعني أنك ستحتاج هنا إلى إجادة استخدام واحد من أنظمة إدارة قواعد البيانات (أفترض هنا أنه الأكسس بالطبع). وأظن أن هذه من أكثر المهارات توفراً في حالة المتابعين لهذه السلسلة. مثلاً، كتاب مثل تعلم الأكسس خطوة خطوة، يزودك بالأساس الكافي للتعامل مع الأكسس من منطلق الاستخدام (وليس تصميم قواعد البيانات أو البرامج). أهمية إجادة SQL هنا قد لا تبدو واضحة، لكن خذها كلمة مني، مبرمج قواعد بيانات لا يجيد لغة قواعد البيانات، ليس مبرمج قواعد بيانات. المزيد عن ذلك فيما بعد إن شاء الله. هل تجد في نفسك المقدرة الكافية على التعامل مع الأكسس على سبيل المثال؟ ممتاز. لنر إذاً ما ينتظرنا بعد.

عند الحديث عن بناء التطبيقات، سأتجاوز مرحلتي التحليل والتصميم (مفهوم ضمنياً، أصول التحليل هنا هي نفسها أصول تحليل متطلبات قواعد البيانات، لكن التصميم قد يشمل مفاهيم وأدوات أخرى لم نتطرق إليها من قبل، على سبيل المثال، مخطط تدفق البيانات Data Flow Diagram (DFD)، لكن هذا من الممكن تجاوزه مؤقتاً في الأنظمة الصغيرة)، أحب أن أركز هنا على جزء تنفيذ التصميم، وبناء التطبيق فعلياً.

تنفيذ التصميم يعني البرمجة، والبرمجة تعني استخدام لغة برمجة محددة. لغات البرمجة التي ستستخدمها قد تكون لغات برمجة متعددة الأغراض (عامة الغرض) general-purpose، وهذا يعني أنها ليست متخصصة في الاتصال بقواعد البيانات، ولا تحوي وظائف من أصل اللغة لهذا الغرض. علاوة على كل ذلك، لغات البرمجة هذه الأيام لا تنفصل عن بيئات مدمجة للتطوير Integrated Development Environments (IDE)s، تحوي بالإضافة إلى المترجم ومحرر النصوص لكتابة اللغة، العديد من مكتبات الأكواد والأدوات المساعدة.

أرجو الآن أن تأخذ رشفة من كوب الشاي، وأن تركز معي. لا أحب تضخيم الأمور، لكن الفقرة السابقة قد احتوت على العديد من المهارات التي ربما لم تكن تأخذها في الحسبان. في البداية، وللأسف، من أكثر الأجزاء عرضة للتجاهل، الحقيقة (المرة) أنك ستحتاج لمهارة البرمجة... أعني البرمجة... هذا يعني البرمجة. لا تقلق، لم أنم على لوحة المفاتيح بعد، وأنا أقصد تماماً كل (زر) ضربته. أقصد أنك ينبغي أن تلم بمفاهيم البرمجة بغض النظر عن لغة برمجة معينة. البرمجة علم وفن بحد ذاتها، وكما أن لتصميم قواعد البيانات أصول لا تتعلق بالنظام المحدد لإدارة قواعد البيانات الذي ستختاره، فإن للبرمجة أصول لا تتعلق بلغة برمجة بعينها. هذا لا يعني أنه يتوجب عليك أن تقرأ بشكل خاص في تصميم لغات البرمجة، لكنه يعني أنك عندما تتعلم لغة برمجة، حاول أن تركز على الفصول الأولى التي تشرح مفاهيم البرمجة العامة، مثل المفهوم الأساسي للخوارزميات (algorithms)، وهذا يعني كيفية التفكير بأسلوب الحاسوب لحل المشاكل، والأساليب المختلفة التي تستخدم في بناء البرامج (برمجة هيكلية، كائنية structured, OOP)، ومعنى المتغيرات وأنواعها، وكيف يمكن أن تستخدم هياكل مختلفة للبيانات (مثل المصفوفات)، وكيف يمكن أن تتحكم في سير البرنامج (التفرع والعبارات الشرطية والتكرارات branching, conditional statements, looping). كل ذلك قبل أن تتعرف على إمكانيات والقواعد الإملائية للغة المحددة التي اخترتها (مثل VBA).

(الجزء الثاني)

هل هذه هي كل الحكاية؟ كلا بالطبع، إذا اخترت لغة مثل PL/SQL، لتطوير التطبيق، فقد لا تحتاج لأكثر من الإلمام بالوظائف التي توفرها أوراكل مع هذه اللغة، لكن في حالة لغة مثل VB، فإنك بحاجة إلى الوصول إلى قاعدة البيانات عبر تقنيات خاصة؛ في حالة منتجات مايكروسوفت، كانت تقنية DAO Data Access Objects تقوم بهذا الغرض، لكن مايكروووسوفت أصبحت توصي باستخدام تقنية ADO ActiveX Data Objects منذ عدة سنوات. هذا بخلاف أن تكون متآلفاً مع بيئة التطوير التي ستختارها، مثلاً في الأكسس، شاشة محرر VBA، وشاشة تصميم النماذج والتقارير. بالمناسبة، في حالة VBA على سبيل المثال، هناك أيضاً بالإضافة إلى الوظائف المدمجة مع اللغة Built-in Functions، ينبغي أن تتعرف على موديل كائنات الأكسس Access Objects Model، وهذا يعني مجموعة الكائنات التي يوفرها الأكسس لتتعامل مع مختلف أجزائه، ولتقوم بالعديد من الوظائف. هذا يعني أنك متآلف جيداً أيضاً مع مفهوم الكائنات والخصائص والطرق. هل ذكرت شيئاً عن لغة SQL؟ إذاً امسح كل ما سبق واكتب SQL! لا أريد أن أجعل حياتك كابوساً، لكن لا أستطيع أن أجعلها حلماً جميلاً كذلك.

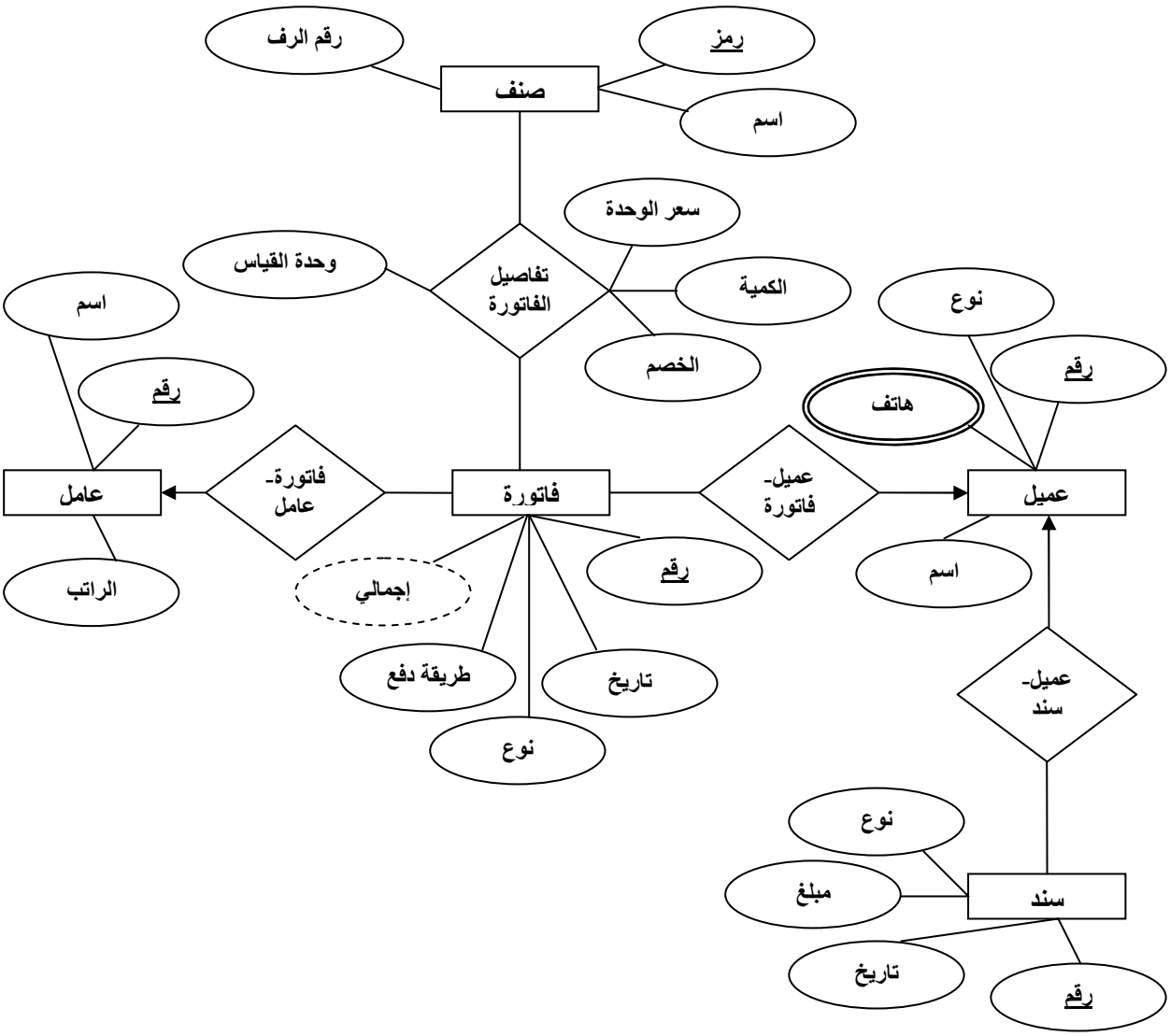
كان هذا مجرد لمحة على ما تحتاجه لتصبح حقاً مبرمج قواعد بيانات. وكما ربما تكون قد لاحظت، هناك العديد من المتطلبات، والمسألة ليست في غاية السهولة كما يتصورها البعض، ولكنها ليست صعبة مع بعض القيود. أقصد ببعض القيود أنك لا تشترط على نفسك أن تكون محترفاً في كل هذه الجزئيات معاً. من الصعب جداً أن تكون من صناديد تصميم قواعد البيانات، ومن مخضرمي الخوارزميات وهياكل البيانات، ومن فصحاء لغات من بينها SQL، ومن خبراء تقنية ADO الأفذاذ في نفس الوقت ، لكن من الممكن جداً أن تكون على دراية كافية بكل هذه المكونات لتنتج تطبيقات قواعد بيانات ممتازة. بالمناسبة، في حالة رأيتم أن من المفيد التعرف على ما تقدم من (أهوال)،

فإن في الإمكان أن نخصص بعض الحلقات لمناقشة بعضها إن أردتم. كالعادة، لن تجعلك هذه المذكرات خبيراً، لكنها قد تفتح لك بعض الأبواب، وإذا نجحت في (استفزازك) لتعلم المزيد، فإنها لمذكرات عظيمة.

نعود الآن إلى المحطة الأولى، وهي برنامج المخازن والمبيعات. وأمر هذا النظام غريب. على الرغم من أنه أكثر البرامج شيوعاً وعرضة للتطوير والنقاش، إلا أنك ما تنفك تجده مثاراً للاختلاف والاستفسار المستمر. جزء من السبب يعود إلى حقيقة اختلاف المتطلبات الدقيقة بين نظام مبيعات إلى آخر على الرغم من توحد الأساس، وجزء يعود إلى كسل بعض المبرمجين، ولجوئهم إلى (توارث) نفس برامج سابقة حتى لو لم توفر المطلوب بالشكل المرضي، هذا علاوة على لجوء البعض إلى تجاهل بعض الأجزاء المستعصية و(دسها تحت البساط). لكن من المهم هنا التركيز على جزئية مهمة: ليست كل برامج المخازن والمبيعات متطابقة في التفاصيل.

دعني أوضح النقطة الأخيرة بشيء من التفصيل. هناك برامج مخازن صرفة، لا تبيع ولا تشتري مباشرة، ولكنها تتعامل مع أوامر توريد وأوامر صرف. فكرة هذه الأوامر تشبه تماماً فكرة فواتير الشراء والبيع. لكن المخازن الصرفة قد تتضمن أيضاً أكثر من فرع (مخزن)، لذلك تجد حركات مثل أوامر نقل أصناف من مخزن إلى آخر، قد لا تجدها في برامج المبيعات المباشرة (محلات أو صيدليات تتكون من مخزن واحد) أو نقاط البيع على سبيل المثال. هناك برامج تتعامل مع أصناف قابلة للانتهاء، مثل بقالات المواد الغذائية والصيدليات، مما يبرز الحاجة لمعالجة قضية تواريخ صلاحية الأصناف. على الرغم من أن أغلب المخازن تتعامل بأكثر من وحدة قياس، إلا أن هناك مخازن لا يهمها إلا وحدة قياس واحدة، ولا تحتاج إلى دعم الوحدات الفرعية. المقصود أنه من الأفضل دائماً تضمين أكثر الأجزاء التي يغلب الاحتياج إليها، لكن من غير العملي أن تتوقع من برنامج واحد أن يستوعب جميع (التشكيلات) والاحتمالات. في نقاشنا هذا بإذن الله، سنحاول التطرق إلى أغلب الاحتياجات، ولكن قد نهمل التفاصيل التي يسهل الحصول عليها.

دعنا نختصر على أنفسنا الطريق، ونستفيد من مخطط الـERD الذي توصلنا إليه من قبل، ثم نواصل من هناك...



اتفقنا على أن الأصناف التي يتعامل بها المحل هي كائنات مستقلة مثّلناها بكائن (صنف) في المخطط، ومصيرها أن تتحول إلى جدول في قاعدة البيانات. كل كائن له خصائص (تتحول إلى حقول حسب ما تقدم من قواعد). هذه الخصائص لا تأتي من الفراغ، وإنما تعكس واقع النظام في متناول اليد، ولذلك قد تختلف يسيراً من نظام إلى آخر، حسب احتياج النظام.

لا شك أن كل الأصناف في كل المحلات لا بد أن تملك رقماً واسماً على الأقل. وحتى من هنا فقط يمكن أن نختلف. بعض المحلات يهمها رقم تسلسل الصنف كثيراً، ولها طرق خاصة في ترميز الأصناف. هذا التسلسل قد لا يكون رقماً البتة، بل مزيجاً من الأرقام والحروف التي لها معنى مصطلح عليه، مثل أن تخصص بعض الخانات لتمييز نوع أو مكان الصنف (مثلاً، أول رقمين لمجموعة الصنف، وأربعة أرقام لرمز الصنف نفسه)، بل إن هناك من المصممين من يرى أن يتم تمييز تواريخ صلاحيات الأصناف في حقل الترميز نفسه. سنتكلم عن هذا بإذن الله عند الحديث عن التعامل مع صلاحيات الأصناف. هناك في المقابل من لا يهتم بترميز الصنف، ويسمح للبرنامج أن يعطي أرقاماً متسلسلة، بشرط ألا يتكرر ترميز واحد لصنفين مختلفين. من الممكن طبعاً دمج الطريقتين، واستخدام ترقيم تلقائي لتمييز الأصناف داخلياً في البرنامج، مع إتاحة حقل ترميز نصي للمستخدم من أجل إدخال الترميز الخاص. في حالة استخدام ترميز خاص يعتمد على عدد الخانات ومعنى كل خانة، فإنه يكون من المهم للمبرمج أن يتأكد من سلامة الترميز المدخل من خلال قواعد التكامل (مثل قواعد التحقق من الصحة في أكسس validation rules) ومن خلال الكود. اختيار كل هذا يعتمد على احتياج النظام، وحتى عند إتاحة الحرية للمبرمج من صاحب النظام، قد يختار المبرمج الطريقة الأسهل (وهذا اختيار وجيه وجيد)، أو قد يختار الطريق الأصعب من أجل جعل النظام مستعداً لكل الاحتمالات. ليس المقصود في نقاشنا هنا أن نغطي كل الاحتمالات، وإنما أن نعطي مثالاً لهذه الاحتمالات، وأن نتعلم الخروج من قوقعة القوالب الجاهزة، وأن نفكر!

بالنسبة لاسم الصنف، قد تحتاج أحياناً إلى إضافة الاسم الإنجليزي، وأحياناً أخرى (مثل الصيدليات) حتى إلى الاسم العلمي، وهذا يعتمد كالعادة على حاجة المستخدم، وطريقته في النظام اليدوي (إن كان له نظام يدوي). بعض أنواع الأصناف لها أيضاً نوع ومصنّع، وتفاصيل أخرى مثل الحد الأدنى للطلب، وهو الكمية التي إن نزل إليها رصيد الصنف (أو نزل عنها)، ينبغي عندها أن يسارع صاحب المحل بطلب كمية جديدة أو شراء كمية جديدة، وهذا يساعد صاحب النظام على الانتباه إلى توفير الأصناف التي عليها طلب كثير، حتى لا يقع في فخ النفاد المفاجئ لصنف معين. لاحظ من فضلك أن هناك حاجة إلى آلية يتم بها تنبيه المستخدم إلى الوصول إلى (أو تجاوز) الحد الأدنى لصنف، وبعيداً عن الآليات المعقدة، من المناسب توفير تقرير للمستخدم يشغله دورياً حسب رغبته تظهر فيه الأصناف التي وصلت أو تجاوزت حدها الأدنى. قد تحتاج أيضاً إلى تضمين حقل لرقم الرف، هذا من أكثر الحقول فائدة لعمل المستخدم اليومي في أنظمة المبيعات في حالة التزام المستخدم بترتيب الأصناف في رفوف أو أماكن مخصصة. مثلاً، قد يحتاج بائع جديد إلى وقت طويل لحفظ أماكن الأصناف المختلفة، وقد يساعده هذا الحقل في البداية.

موضوع آخر هو مسألة الحقول المحسوبة. هل يمكن أن يحوي جدول الأصناف حقولاً محسوبة؟ بالطبع، بعضاً من أهم الحقول المحسوبة في النظام. في أول هذه الحقول يأتي رصيد الصنف (أو الكمية المتوفرة). واحدة من أصعب المهام في برامج المخازن تتبع رصيد الصنف، الذي يتغير باستمرار انعكاساً لمختلف الحركات في المخزن، وكلما زادت الحركات، زاد معدل التغير وصعوبة المحافظة على القيمة الصحيحة للكمية المتوفرة. مثلاً، إذا كانت الحركات تشمل مردودات البيع والشراء بالإضافة إلى البيع والشراء فإن اختيار تضمين الحقل المحسوب في جدول الأصناف يعني أن على المبرمج أن يبذل المزيد من الجهد في الانتباه لكل هذه التغييرات. بعض هذه التغييرات ليست واضحة في البداية. مثلاً، قد يحتاط المبرمج لحركات البيع والشراء ومردوداتهما، لكنه ينسى احتمال تغيير فاتورة محفوظة أو حذفها أو زيادة بند فيها. كثير من البرامج الموجودة تتفاضل في فكرة وآلية هذا التتبع، سواء كان عبر حقل محفوظ في الجدول أو يتم استخراجه (على الطاير) عند الاستعلامات والتقارير المختلفة. سنتكلم بإذن الله عن هذا الأمر عند الكلام عن الفواتير.

هل توجد حقول محسوبة أخرى يمكن أن يشملها جدول الأصناف؟ نعم، يمكن (هل من الممكن أن تفكر في مثال لذلك؟)... أحياناً تحتاج في بعض الأنظمة (خاصة تلك المرتبطة بأنظمة محاسبة) إلى أن تحصل على قيمة المخزون من الصنف في تاريخ معين (آخر فترة محاسبية). المشكلة أن الكميات المتوفرة في المخزن قد تم شراؤها بأسعار مختلفة، ومن الصعب (أحياناً مستحيل) تتبع كل كمية من الصنف وضربها في سعر شرائها بعد فترة طويلة من الحركة. واحد من الحلول الممكنة استخدام طريقة استخراج المتوسط المرجح لسعر الصنف، ويتم ضرب هذا المتوسط بالكمية المتوفرة في آخر الفترة. لاحظ أن هذا الحقل محسوب.

تكلمنا أيضاً من قبل عن حكاية الكمية الافتتاحية. في الغالب لن يبدأ صاحب المحل نشاطه من يوم شرائه لبرنامجك، ولكنه سيبدأ استخدام البرنامج مع وجود كمية من الأصناف في مخزنه لم يتم تسجيل حركاتها (فواتير بيعها وشرائها)، وهذا يعني أنه لا يمكن أخذها في الحسبان عند احتساب الرصيد المتوفر من الصنف. الحل؟ لا بد من وجود طريقة ما يتم عبرها توثيق هذه الكميات الموجودة أصلاً منذ البداية، وواحدة من الطرق لذلك استخدام حقل في جدول الأصناف لحفظ الكمية الافتتاحية، حتى يتم تضمينها في الاحتسابات فيما بعد. مثلاً، بعد ثلاث عمليات شراء، سيكون رصيد الصنف الفلاني هو مجموع الكميات في هذه العمليات بالإضافة إلى الكمية

الافتتاحية (طبعاً ناقصاً أي كميات في عمليات بيع مثلاً). لاحظ أن إدخال هذه الكميات الافتتاحية من مهمات المستخدم.

كل هذا جميل ولاغبار عليه، لكن دعنا الآن نتعرض لبعض النقاط التي قد تكون موضع نقاش، سنمر هنا على نقطتين مهمتين بإذن الله، ثم نغادر جدول الأصناف.

وحدات القياس، وما أدراك ما وحدات القياس! من الحقول التي تميز الصنف وحدة قياسه. قد يحدث أن يتعامل المخزن مع وحدة قياس واحدة، وعندها لا يكون لهذا الحقل أي أهمية تتعدى توثيق وحدة قياسه، ولا يكون عندئذ أي إشكال. مثال على ذلك متجر لبيع الجوالات، لا يتعامل إلا بالقطعة. لكن في كثير من الأحيان يتم التعامل بأكثر من وحدة قياس للصنف الواحد في الحركات المختلفة. في هذه الحالة، يصح أن نقول إن لهذا الصنف أكثر من وحدة قياس واحدة. المشكلة ليست في توثيق هذه الوحدات، لكن المشكلة في التعامل الصحيح مع الكميات عند الحركات المختلفة، من أجل الاحتساب الصحيح لرصيد الصنف. عادة ما يتم احتساب إجمالي سطر في فاتورة بيع مثلاً، بضرب الكمية في سعر الوحدة، لكن الكمية هنا قد تتغير مع تغير الوحدة. مثلاً، عند شراء صنف دواء، قد تكتب الكمية (7)، والسعر (350)، مع تحديد وحدة القياس بباكت. لا مشكلة مادام سعر الباكت الواحد هو بالفعل 350. فاتورة أخرى يظهر فيها نفس الصنف بكمية (10) وسعر (70)، مع وحدة القياس كشريط. لا مشكلة أيضاً مادام سعر الشريط هو 70 (ربما كان في الباكت خمسة أشرطة). المشكلة قد تظهر عند استخراج رصيد الصنف، هل هو (7 + 10 = 17)؟ طبعاً الإجابة خاطئة تماماً. كيف يمكن أن تعلم الرصيد الحقيقي؟ لا يمكنك ذلك إذا لم تكن تعلم العلاقة بين الباكت والشريط. كذلك، ينبغي أن تختار وحدة واحدة ليتم التعبير بها عن الرصيد. إذا اخترت الشريط كوحدة أساسية، وعلمت أن الباكت يحوي خمسة أشرطة، فإن الرصيد يصبح (7 * 5 + 10 = 45).

بعض الحلول التي ربما كنت صادفتها، تحديد وحدتين (أو ثلاث) فقط للصنف، مع تحديد ما تعادله الوحدة الثانية من الوحدة الأولى. البعض قد يفكر في إجبار المستخدم على اختيار وحدة واحدة (الأصغر مثلاً)، وإدخال كل الحركات بها. هذا الخيار الأخير لن يعجب المستخدم كثيراً في الواقع، لكن الكثير من المستخدمين عندنا قد تعلموا أن بعض الحركات لا يمكن تنفيذها بالحاسوب! ولهذا تعرف مدى خيبة هذا المستخدم عندما يكتشف فيما بعد أن كل العالم يوفر هذه الإمكانيات. مثل هذه الأمثلة تجعل البعض يرتاب في البرامج المحلية، بالذات المصممة من قبل أفراد، ويحسب ألف حساب قبل أن يفكر في استخدامها (إلا لو كان يعاني من محدودية الميزانية). الحل الأكثر مرونة هو تكوين جدول مستقل للوحدات الفرعية لكل صنف، بحيث يتم اختيار وحدة واحدة أساسية من قبل المستخدم لكل صنف، وتحفظ في جدول الأصناف، ثم تتم إضافة وحدات أخرى (فرعية) للصنف في جدول الوحدات الفرعية، مع ربط الجدولين بحقل رمز الصنف بالطبع. لاحظ أنه من المهم تضمين حقل لمعامل التحويل بين الوحدتين في جدول الوحدات الفرعية. مثلاً، صنف معين وحدة قياسه الأساسية هي الشريط (في جدول الأصناف). وله وحدة قياس فرعية هي الباكت (في جدول الوحدات الفرعية) مع حقل معامل التحويل يساوي 5 (كل باكت يساوي خمسة أشرطة). بهذا يمكن دائماً احتساب الرصيد بالوحدة الأساسية لكل صنف، مع إمكانية التحويل إلى هذه الوحدة في حالة وجود حركات بغير هذه الوحدة. يجب أن تلاحظ هنا مقدار العبء البرمجي الذي ألقاه هذا المتطلب (تعدد الوحدات) على كاهل المبرمج.

موضوع آخر هو سعر الصنف. كثيراً ما تجد حقل سعر الصنف في جدول الأصناف، على الرغم من حقيقة أن سعر الصنف ليس بالضرورة من ميزات الصنف الثابتة، إذ أنه من الممكن جداً أن يتغير سعر الصنف في كل فاتورة تبعاً للزبون أو لظروف أخرى. ربما كان من المناسب تسمية هذا الحقل (السعر الافتراضي)، وأن يتم استحضاره إلى الفاتورة بشكل افتراضي (ليعطي أساساً للمفاوضة على سبيل المثال)، مع إمكانية تعديله من قبل البائع. لاحظ، مرة أخرى، أن هذا يرجع تماماً إلى سياسة المستخدم؛ بعض المحلات لا يجوز فيها للبائع تعديل السعر (مثل نقاط البيع)، ولذلك لا يتغير السعر إلا بين حين وآخر، وعلى هذا يمكن اعتبار السعر من الخصائص المميزة للصنف.

هل هناك شيء آخر يتعلق بالصنف؟ بالطبع هناك الكثير مما قد ترد الحاجة إليه، لكن هذا يختلف من نظام إلى آخر، وليست هناك أية نقطة في محاولة حصره. لكن دعني أختم الحلقة بالكلام عن نقطة تتعلق بالأصناف أغفلناها عمداً إلى الآن. كثيراً ما تكون الأصناف التي يتعامل بها المحل كثيرة جداً ومتشعبة، لكنها تنقسم طبيعياً إلى مجموعات مختلفة، وقد يكون من المفيد معرفة هذه المجموعات، بل قد يتم التعامل بشكل ما مع مجموعات الأصناف أحياناً، وليس مع الأصناف المفردة. مثلاً، في سوبر ماركت، قد يتم تقسيم الأصناف إلى مواد غذائية، ومواد تنظيف، وأدوات منزلية، ... إلخ. في صيدلية قد يتم تصنيف المواد على أساس أنها حقن أو سوائل، أو حبوب، أو على أساس أنها فيتامينات أو مضادات حيوية أو للضغط أو السكري (عافانا الله وإياكم من كل شر). وقد يحتاج المستخدم إلى استخراج العديد من التقارير حسب المجموعة، أو خلاف ذلك. المهم، في هذه الحالة ينبغي إضافة جدول للمجموعات مع ربط كل صنف بمجموعة عبر حقل مفتاح أجنبي (رقم المجموعة) في جدول الأصناف. طبعاً في هذا بعض التبسيط لأنني أتجاهل حقيقة أن التصنيف قد يكون على عدة مستويات، لكن المقصود هو المثال، ولا فائدة من التشعب كثيراً.

أظنني سأتوقف هنا في هذه الحلقة، وأرجو أن تكون قد وصلت إلى إجابة مقنعة للسؤال الملح: هل أنت.....؟

رقم الحلقة (27) تصميم جداول نظام مخازن ومبيعات: استكمال النقاش

نقطة بين يدي هذه الحلقة:

1. لاحظ من فضلك أنني أكرر دوماً الكثير من المفاهيم بعبارات مختلفة، وهذا مفيد. من الصعب جداً أن تحفظ الشارع من زيارة واحدة، وكلما تعددت الزيارات من مداخل مختلفة، ازدادت ألفتك مع تفاصيل الشارع...

نحن ندندن هنا حول الطريقة السليمة لتطوير أنظمة قواعد البيانات. الطريقة السليمة تعني المنهجية في معالجة الأمور. المنهجية تعني بعضاً من الأمور، منها ما يتعلق بالإعداد الصحيح قبل العمل، ومنها ما يتعلق بأسلوب العمل نفسه. العلم قبل العمل، وهذه القاعدة صحيحة في كل شيء. لا بد من حد أدنى من الفهم لمبادئ تصميم قواعد البيانات قبل البدء بتصميم قواعد البيانات. سبق أن أفضنا في أن هذه المبادئ تشمل أكثر من مجرد القدرة على (الرسم). راجع من فضلك الحلقة السابقة. أما عند الشروع في العمل، فلابد من قضاء بعض الوقت في تصور الحل. عندما كنت تحل مسائل الرياضيات في الماضي، من الصعب أن تتوصل لشيء من خلال الشروع فوراً بخط بعض الجذور، ونثر السينات والصادات هنا وهناك، عسى أن تترتب الأمور لوحدها. لا بد لك قبل كل محاولة من التفكير في المدخل المناسب للحل، بناء على تصور في رأسك لطريقة الحل، أو على الأقل الخطوات القليلة الأولى. عند الحديث عن قاعدة البيانات، التصميم الجيد هدفه مجموعة جيدة من الجداول، ومن طرقه استخدام مخطط الكائنات والعلاقات مثلاً. عند الحديث عن التطبيق، الهدف (بلغة الأكسس من أجل التمثيل) هو مجموعة جيدة من النماذج والتقارير (ضمنياً، والاستعلامات)، ومجموعة من الوحدات البرمجية التي تترجم وظيفة التطبيق، وقد تكون مضمنة في النماذج والتقارير، أو مستقلة في وحدات نمطية modules. عند تصميم التطبيق، قد تستخدم بعض الطرق لتخطيط (طريقة الحل). من هذه الطرق بعض المخططات التي تلخص وظيفة التطبيق، مثل مخطط تدفق البيانات data flow diagram. هذه المخططات تترجم إلى مجموعة النماذج والتقارير... إلخ. عند تناول تفاصيل الوحدات البرمجية ستحتاج إلى تصور تفصيلي للحل. هذه المرة، سيأخذ هذا شكل مجموعة واضحة ودقيقة من الخطوات لتنفيذ مهمة محددة (الحل)، تسمى عادة خوارزمية الحل. الوصفة الجيدة لطبق ما، هي الأساس لإعداد هذا الطبق، لكن حسب خبرة الطباخ، قد يتفاوت نجاح تنفيذ الوصفة من شخص لآخر. بالمناسبة، يتشابه المبرمجون والطباخون في أن لهم أحياناً (وصفات سرية)، ولذلك لا يحب المبرمجون كشف أكوادهم دائماً.

الآن، لنعد إلى موضوع نظام المخازن والمبيعات. توقفنا في الحلقة السابقة عند جدول الأصناف، وكخلاصة، وصلنا إلى أن بعض التفاصيل قد تختلف من تصميم إلى آخر حسب الاحتياجات التفصيلية للنظام، وحسب اختيار المصمم / المبرمج لمعالجة بعض القضايا. لكن من الممكن أن نختار تصميماً عاماً مناسباً في الغالب:

رمز الصنف
اسم الصنف
مجموعة الصنف
وصف الصنف
المصنع
رقم الرف
الحد الأدنى
الكمية الافتتاحية
الكمية المتوفرة
الوحدة الأساسية
السعر الافتراضي
ملحوظات

لاحظ أنه قد يكون من المفيد تضمين حقول للوصف العام، أو لكتابة بعض الملحوظات في كثير من الجداول. أفترض هنا أنه ليس للصنف مورد أساسي بذاته، كما أن موضوع الوحدة الأساسية يفرض إضافة جدول للوحدات الفرعية حسب كلامنا السابق. هذا الجدول يحوي الحقول التالية:

رمز الصنف (مفتاح أجنبي لربط الوحدة الفرعية بصنف من جدول الأصناف)
الوحدة الفرعية
معامل التحويل (راجع من فضلك الحلقة السابقة من أجل التفاصيل...)

من أهم الكائنات في نظام المبيعات، كائن الفاتورة. تشكل الفواتير الحركة الأساسية لهذه الأنظمة. بناءً على مخططنا الذي سبق تصميمه، تنقسم الفواتير إلى بيانات أساسية للفاتورة نفسها (كمستند)، وإلى تفاصيل الفاتورة التي تحوي بيانات الحركة (أصناف وكميات وأسعار). بعض بيانات الفاتورة كمستند واضح جداً ومتفق عليه. لكل فاتورة رقم وتاريخ. هذا أهم ما يميز أي مستند بغض النظر عن نوع حركته. قبل الانتقال إلى البيانات الأخرى، دعنا نوضح القليل عن رقم الفاتورة. في النظام اليدوي، تطبع الفواتير على شكل دفاتر، وتحوي كل فاتورة رقماً مطبوعاً مسبقاً. إما أن يختار صاحب المحل استخدام هذه الأرقام في نظامك كأرقام الفواتير (في هذه الحالة

ستحتاج إلى أن تجعل حقل رقم الفاتورة من نوع نص وليس رقم، لأنه قد يحوي أصفاراً بادئة على اليسار)، وإما أن يقرر صاحب المحل الاعتماد على البرنامج تماماً، ويطبع الفواتير من البرنامج مع حقل رقم الفاتورة كرقم معتمد للفاتورة. طبعاً من الممكن أن يكون هناك حقلان، واحد للاستخدام الداخلي في البرنامج، وواحد لإدخال الرقم المطبوع على الفواتير الدفترية.

بخلاف الرقم والتاريخ (تاريخ اليوم افتراضياً، مع إمكان إدخال فواتير من الماضي، ولكن ليس المستقبل)، هناك طريقة الدفع. هذا الحقل مهم جداً لأنه من المهم تمييز الدفع الآجل من أجل استخراج رصيد العملاء (الديون، لنا، في فواتير البيع، أو علينا، في فواتير الشراء مثلاً). كما أنه قد يريد صاحب المحل تتبع حركات البائعين عنده، فتحتاج إلى تضمين حقل لرقم العامل الذي باشر الفاتورة كمفتاح أجنبي من جدول العمال (أو الموظفين). قد يكون من المفيد أيضاً تضمين حقل للملحوظات، وقد سبق أن تكلمنا حول الحقول المحسوبة، وما لها وما عليها في الحلقات السابقة، ونحن هنا لدينا حقل إجمالي الفاتورة الذي إن اخترنا تضمينه، فيلزمنا احتسابه من جدول آخر هو تفاصيل الفواتير.

نعم، أعلم. أنت تريد أن تسأل هنا عن موضوع المردودات (أو المرتجعات). يحدث أحياناً أن نرد بعض البضاعة المشتراة من مورد، أو يرد زبون ما بعض البضاعة المباعة. هذه المردودات (مردودات الشراء ومردودات البيع) هي حركات أيضاً تؤثر في رصيد الصنف، وفي رصيد العميل في حالة بيع وشراء الآجل. من أجل تصميم حل جيد لهذه المشكلة، نفترض بعض الافتراضات الصحيحة غالباً. أولاً، نفترض أن للمردودات فواتير. كما أن هناك فاتورة بيع، هناك أيضاً فاتورة مردود بيع، وهكذا. قد يبدو هذا الافتراض بديهياً، لكن لا بد من توضيح الافتراضات الضمنية بشكل صحيح حتى يسهل استيعاب الحل. ثانياً، شكل فاتورة المردود مشابه تماماً لفاتورة الحركة الأصلية. هذا يعني أن فاتورة مردود البيع على سبيل المثال لها نفس التفاصيل مثل فاتورة البيع من رقم وتاريخ وطريقة دفع (نعم، قد أرد بضاعة ولا أستلم فوراً ثمنها الذي سبق أن دفعته). ومع ملاحظة أن فواتير البيع والشراء متماثلة أيضاً تماماً في التفاصيل، فلذلك تمت معاملة كل أنواع الفواتير ككائن واحد وليس أربعة (جدول واحد وليس أربعة جداول). لكن بالطبع نحتاج من أجل التمييز بين النواع الأربعة المختلفة إلى حقل أسميناه (نوع) الفاتورة. هذا الحقل لا يمكن أن يحوي غير أربعة قيم (بيع، شراء، مردود بيع، مردود شراء).

هل كان ذلك واضحاً؟ جيد، إذاً دعنا الآن نكسر بعض افتراضاتنا السابقة! التحليل قد يخبرك بمتطلب دقيق لصاحب المحل، ولا بد من مراعاته في التصميم. قد يشترط صاحب النظام أن يرتبط كل مردود بفاتورة أصلية محددة من حيث الأصناف والكميات. معنى ذلك أنه في حالة رد بضاعة مباعة (مردود بيع) يجب أن يتم رد البضاعة على أساس فاتورة البيع التي تم فيها بيع هذه البضاعة بعينها. لا يمكن رد بضاعة لم يتم شراؤها في فاتورة البيع هذه نفسها، ولا رد كميات أكبر من الكميات المباعة في نفس الفاتورة. في هذه الحالة، ستحتاج إلى تضمين حقل آخر اسمه مثلاً (الفاتورة المصدر)، يربط كل فاتورة مردود بفاتورة أصلية. هنا بضع ملحوظات:

1. ينبغي عليك كمبرمج أن تتأكد من أن فاتورة المردود تلتزم بأصناف وكميات الفاتورة المصدر.
2. هذا الحقل (الفاتورة المصدر) يحوي أرقام فواتير، وهي المفتاح الأساسي لجدول الفواتير، وعلى هذا فإن هذا الحقل هو مفتاح أجنبي (غريب وسط أهله!)، يربط جدول الفواتير بنفسه، ويسمى هذا self-relationship، علاقة ذاتية.
3. لاحظ أن هذا الحقل سيبقى فارغاً في أغلب السجلات، وهي فواتير البيع والشراء الأصلية، لكن هذا لا بأس به باعتبار أننا وفرنا إنشاء أكثر من جدول، وما يستتبع ذلك من ربط في الاستعلامات المختلفة.

ملحوظة أخرى، أن حقل نوع الفاتورة قد يشمل إلى جانب الأنواع الأربعة المذكورة أعلاه نوعاً آخراً، سبق أن أشرنا إليه سابقاً. ذلك هو نوع (كمية افتتاحية)، في حالة أن إدخال الرصيد الافتتاحي سيتم عبر فاتورة واحدة في بداية استخدام النظام.

من الجيد أن تلاحظ من خلال النقاش السابق، أن الحلول الجيدة لبعض المشاكل الشائعة كثيراً ما تكون بسيطة، لكن فعالة. أحب هنا أن أستطرد في نقطة فلسفية أخرى (أرجو ألا تتضايقوا من استطراداتي، من الممكن أن تغمض عينيك عندما أشرع في الاستطراد، وعندما أنتهي سأفرقع بإصبعي☺). كثيراً ما تكون علامة تمكّن المبرمج بساطة الحل الذي يتوصل إليه. التعقيد ليس علامة الاحتراف دائماً، ولكنه أحياناً علامة عدم التمكن. تحتاج إلى قول الكثير وبأسلوب ملتو بلغة لا تتقنها من أجل توصيل معنى محدد، لكنك ستستخدم على الأرجح كلمات أقل بكثير وأسلوب أوضح باستخدام لغتك الأم (أفترض طبعاً أنك تتقن لغتك الأم☺). لذلك، قد تكون بعض حلولك في البداية طويلة ومعقدة، ثم تجد أنك تستطيع حل نفس المشكلة بكود أقل من حيث السطور أو أقل من حيث التعقيد، وغالباً الفعالية أكثر. بالمثل في تصميم قاعدة البيانات، بشكل عام، الاتجاه نحو البساطة هو علامة الجودة.

نقطتان إضافيتان عن الفواتير. أولاً، لماذا نجد أن البعض يصر على تضمين تفاصيل العميل من اسم وعنوان وهاتف في الفاتورة؟ هل هذا صحيح؟ في الغالب هذا ليس صحيحاً، لأنه تكرار لبيانات العميل الموجودة أصلاً في جدول العملاء، وقد سبق أن أفضنا في ذلك عند حديثنا عن التسوية. أقول (في الغالب) لأن هناك حالة خاصة قد يتوجب علينا فيها تضمين بعض تفاصيل العميل في الفاتورة، وبالذات ما يتعلق بالعنوان والهاتف، وربما أن البعض قد اطلع

على هذا التصميم، وظن أنه التصميم العام الصحيح. هذه الحالة هي حالة الطلبيات التي سيتم توصيلها (شحنها) إلى أماكن مختلفة كل مرة، وليس إلى عنوان ثابت للعميل. تحتاج هنا إلى إضافة عنوان التوصيل لهذه الفاتورة في الفاتورة نفسها. مثلاً، تجد في قاعدة بيانات Northwind المرفقة مع الأكسس، أن من بيانات الفاتورة (الطلبية order) بعض الحقول المتعلقة بمكان الشحن shipping.

ثانياً، قد تختار أيضاً إضافة حقل من أجل المبلغ المدفوع في فواتير الآجل، ويحوي جزءاً من إجمالي الفاتورة. ينبغي أن تراعي هذا الحقل عند احتسابك لرصيد العملاء. التصميم البديل الذي أراه مناسباً، هو استقبال كل مدفوعات العملاء بسندات مالية في جدول السندات. هذا يقودنا إلى تصميم جدول السندات الذي يحوي تفاصيل السندات المالية. هذه السندات على نوعين؛ النوع الول هو سندات قبض تثبت أننا استلمنا (قبضنا) مبلغاً من العميل كتسديد لجزء من ديونه لنا. هذه الديون أتت طبعاً من فواتير الآجل. سندات القبض يأخذها العميل كإثبات له، لكننا نوثقها في البرنامج من أجل إثبات الحركة. النوع الثاني من السندات المالية هو سندات الدفع، وتثبت أننا قد دفعنا للعميل جزءاً من ديوننا له. طبعاً هذه الديون تولدت من مشترياتنا بالآجل (أو من مردودات بيع العميل التي لم نعد له فيها ثمن البضاعة). تفاصيل السندات واضحة ومباشرة، ولا تحتاج إلى نقاش وتشمل:

رقم السند (راجع النقاش حول رقم الفاتورة من فضلك)
تاريخ السند
نوع السند (قبض/دفع)
العميل
المبلغ
البيان (وصف لتفاصيل الحركة)

نعود إلى الفواتير، وبنظرة سريعة، من الممكن أن نتفق على الحقول التي ينبغي أن يضمها جدول تفاصيل الفواتير. لا بد من رقم الفاتورة طبعاً، وحقل لرقم الصنف (هذان الحقلان هما معاً المفتاح الأساسي لجدول تفاصيل الفواتير، ويربطان هذا الجدول بجدولي الفواتير والأصناف، على التوالي). أيضاً من البديهي وجود حقل للكمية، ولسعر الوحدة، وربما للخصم، وغالباً لوحدة القياس كذلك في حالة الأصناف متعددة الوحدات. الكمية ستتبع الوحدة، أما السعر فحسب نقاشنا السابق، قد يحوي افتراضياً السعر الافتراضي من جدول الأصناف، مع إمكانية تغييره (أو عدمها، حسب المطلوب). من الممكن تطبيق خصم معين حسب رقم العميل إما في حقل السعر أو في حقل الخصم. الخصم هنا يعني الخصم على سطر واحد في تفاصيل الفاتورة، على صنف معين، وهذا يعطي مرونة تحديد الخصم لكل صنف على حدة. الطريقة الثانية لاحتساب الخصم هي بتضمين حقل للخصم في جدول الفواتير كخصم إجمالي على كل أصناف الفاتورة. طبعاً كالعادة، من الممكن دمج الطريقتين معاً. أيضاً قد تريد تضمين إجمالي السطر في هذا الجدول، لكن العناء لا يستحق، لأن احتساب السطر مباشر وسهل، ولا يحتاج إلى ربط مع جداول أخرى (السعر * الكمية * (1 – الخصم / 100، على اعتبار أن الخصم نسبة مئوية). كان هذا فيما يتعلق بالجزء الواضح، دعنا الآن نلق نظرة على الأجزاء الأقل وضوحاً.

من المهم في البداية أن تدرك أن الفواتير تمثل محور النظام، ولذلك عليك أن تتوقع الكثير من الصداع عند إعداد شاشات الفواتير. هذا لا يعني المزيد من أكواب الشاي وحبوب المسكنات بقدر ما يعني المزيد من الانتباه والوقت. لا تبخل بالوقت. سبق أن ذكرت لك في حلقة قديمة أن هناك مثلاً يقول: البخيل يشتري مرتين، لأنه يبخل بالنقود في المرة الأولى فلا يشتري إلا منتجاً رديئاً، سرعان ما يخذله ليضطر إلى الشراء من جديد. ربما لو بذل المقدار الكافي من النقود في المرة الأولى لما اضطر إلى الشراء مرة أخرى. لذلك، حاول أن تبذل الوقت الكافي من المرة الأولى، حتى تقل عدد مرات عودتك لنفس الشاشة. من أسباب الصداع في شاشة الفواتير:

1. من هنا سوف تقوم بتحديث رصيد الصنف عند كل الحركات، في حالة اخترت تضمين حقل (الكمية المتوفرة) في جدول الأصناف.
2. من هنا سوف تقوم بتحديث رصيد العملاء عند كل حركة بالآجل، في حالة اخترت تضمين حقل (رصيد العميل) في جدول العملاء.
3. ينبغي الانتباه عند تحديث رصيد الصنف إلى حقل وحدة القياس من أجل احتساب الكمية السليمة بالوحدة الأساسية. لاحظ أن ذلك قد يستلزم التحويل بين الوحدات حسب معامل التحويل في جدول الوحدات الفرعية.

الذي أغفلت ذكره حتى الآن أن التناول الصحيح –في رأيي- لمشكلة صلاحية الأصناف يتم هنا عند الفواتير، في جدول تفاصيل الفواتير بالتحديد. لن أخوض في كل الحلول الممكنة، لكنني سأعرض الحل الأمثل بإذن الله في تقديري. المشكلة في تواريخ انتهاء الصلاحية أنها ليست ثابتة للصنف، ولذلك من الصعب جعلها من خصائص الصنف. كل مرة تشتري فيها صنفاً، فإن تاريخ الصلاحية يختلف على الأرجح. ولذلك تجد كميات من نفس الصنف بصلاحيات مختلفة. لكن من الواضح جداً أن كل كمية بصلاحية معينة ترتبط بفاتورة معينة، وعلى هذا يكون أفضل مكان لرصد تاريخ الصلاحية هو جدول تفاصيل الفاتورة، مع تفاصيل الفاتورة الأخرى مثل الكمية والسعر. هذه الكمية تم شراؤها بهذه الوحدة، وبهذا السعر، وبتاريخ الصلاحية هذا. بهذا الشكل، أنت تستطيع تمييز الكميات حسب تواريخ الصلاحية، مع معرفة رصيد تاريخ صلاحية معين، لأنك تتتبع حركة تاريخ الصلاحية هذا بيعاً وشراءً. مسؤولية صاحب المحل هنا الالتزام بكتابة تواريخ الصلاحية الصحيحة للكميات التي يبيعها، وأن يقوم بترتيب

عمليات بيعه بشكل مناسب بحيث يبيع الكميات بتواريخ الصلاحية الأقدم أولاً. من الممكن أن نساعده هنا بإظهار الكميات المتبقية من الصلاحيات المختلفة حتى يعلم بوجود كميات من تاريخ صلاحية قديم، فيقوم ببيعها أولاً. هناك أساليب أخرى لمساعدته، لكنني سأقتصر على هذا لأنني أشعر أن الكلام طال، وأن الشرح ليس واضحاً تماماً. على كل حال، تستطيع الاستفسار عما أشكل بعد الحلقة بإذن الله. لاحظ من فضلك أن كل مشكلة تواريخ الصلاحية قد تم حلها بحقل واحد فقط في جدول تفاصيل الفواتير، مع بعض الكود طبعاً، وربما الاستعلامات. مثلاً، من الاستعلامات المتوقعة، إظهار الأصناف التي لها كميات تقترب مواعيد انتهاء صلاحياتها. نحن نستطيع استخراج هذا بسهولة لأن لدينا كل كمية من كل صنف مع تاريخ صلاحيتها عند شرائها. خذ مجموع الكميات التي تم شراؤها، وفصلها حسب تواريخ الصلاحية، انتبه للكميات التي تم بيعها من كل صلاحية ثم أظهر الناتج.

ما سبق يقود إلى التصميم التالي لجدول الفواتير:

رقم الفاتورة (ربما اثنان حسب النقاش أعلاه في الجزء الأول)
تاريخ الفاتورة
طريقة الدفع (نقد/آجل)
رقم العميل
رقم العامل
نوع الفاتورة
الخصم الكلي
إجمالي الفاتورة
ملحوظة

لاحظ أن لدينا حقلين من جدولين لم نذكرهما حتى الآن، وهما جدولا العملاء والعاملون... أما جدول تفاصيل الفاتورة فيمكن أن يحوي التالي:

رقم الفاتورة
رقم الصنف
الكمية
سعر الوحدة
وحدة القياس
الخصم
تاريخ الصلاحية

جدول العملاء واضح جداً، ويحوي بيانات كل عميل. نحن نحتفظ ببيانات العميل الذي نتعامل معه بالآجل، أما عملاء التفرقة فمن الصعب، وغير المجدي، الاحتفاظ ببياناتهم. الحقول هنا من الممكن أن تكون:

رقم العميل
اسم العميل
نوع العميل
بلد
مدينة
عنوان
هاتف
جوال
رصيد
ملحوظة

الحقول التي قد تحتاج إلى تعليق هي:

1. حقل النوع يحوي القيم (مورد، زبون، مورد وزبون) ليميز بين ثلاثة أنواع محتملة للعملاء. المورد نشتري منه، والزبون نبيع له، والمورد والزبون من الممكن أن نشتري منه وأن نبيع له بين حين وآخر. أصحاب المحلات ذوي الصنعة الواحدة، دائماً ما يبيع بعضهم لبعض، ويشتري بعضهم من بعض. السؤال الآن: ما الهدف من هذا الحقل؟ طبعاً هذا الحقل يوفر علينا إنشاء جدولين مع وجود سجلات مشتركة بينهما (في حالة المورد والزبون معاً). لكننا نحتاج إلى توثيق نوع العميل أصلاً لأن هناك العديد من الاستعلامات والتقارير التي تعتمد على فرز الموردين أو الزبائن. كما أن معرفة نوع العميل تمكننا برمجياً من التحكم في العملاء الذين قد يختارهم المستخدم في الفواتير حسب نوع الفاتورة (مثلاً، عند فواتير البيع يمكن توفير قائمة بالعملاء من نوع زبون أو مورد وزبون فقط، لكن ليس من نوع مورد).
2. حقل الرصيد حقل محسوب من جدولين مختلفين: جدول الفواتير (عند حركات الآجل)، وجدول السندات المالية. رصيد زبون هو مجموع الديون التي عليه ناقصاً تسديداته.

جدول العاملين لا يحتاج إلى مزيد بيان، ومن الممكن في حالة اشتمال النظام على جزء لإدارة الموظفين أن يمثل جدول الموظفين. هل هذا كل شيء؟ من الممكن، على الأقل هذا يمثل البنية الأساسية لبرامج المخازن والمبيعات. لاحظ أن نقاشنا دار حول تصميم الجداول، وما تحويه من حقول، لكن بعض المتطلبات لن يظهر حلها إلا بالكود. على كل حال، المهم في التصميم أن يوفر كل المطلوب، بحيث يجده المبرمج بشكل أو بآخر بأسهل وأسرع الطرق. من المهم ألا يضطر المبرمج إلى أن يرقع بالكود ما يمكن استكماله بأفضل الطرق في تصميم الجداول، هذا ما أظن ان تصميمنا يحققه إلى حد ما.

كمثال على الإضافات التي قد يتم تضمينها في التصميم، هو أن تضيف ميزة الجرد إلى البرنامج: جدول للجرد يحوي حقلاً لرقم الصنف، وحقلاً للكمية المجرودة يدوياً (في آخر السنة مثلاً). يقوم المستخدم طبعاً بإدخال هذه الكميات المجرودة يدوياً لكل صنف. بعدها يوفر له النظام تقريراً يقارن الكميات المجرودة يدوياً مع الكميات المحسوبة في البرنامج (أرصدة الأصناف الناتجة عن الحركات المدخلة طوال العام). هذا التقرير سيظهر أي عجز أو فائض ناتج من اختلاف الكميات الموجودة بالفعل مع الكميات التي كان يفترض أن تكون موجودة حسب الحركات التي تمت في البرنامج. هذا الاختلاف قد يكون ناجماً عن تلاعب بالمخزن أو عن خطأ في بيانات الحركات المدخلة أو عن تهاون في إدخال كل الحركات. يستفيد المستخدم كثيراً من هذا التقرير.

أيضاً قد يكون من الملائم إضافة ميزة لشطب المواد التالفة في المخزن، حتى يتم أخذها في الحسبان عند احتساب أرصدة المخزون. جدول يحوي رقم الصنف مع الكمية التالفة (ومثلاً تاريخ الشطب، وسببه) سيكون كافياً بإذن الله.

أظن أن لديك الآن قاعدة بيانات مكتملة من حيث الجداول تمثل تصميماً جيداً ومتيناً لنظام مخازن ومبيعات أساسي. تفاصيل بناء هذه القاعدة (مع ما يشمله هذا من اختيار أنواع بيانات كل حقل) ربما نقوم به فيما بعد. لكن ذلك لا يمنعك من أن تسأل نفسك هذا السؤال: هل أنت.....؟

القسم الثاني: SQL

رقم الحلقة (28) 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...

رقم الحلقة (29) 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 مساعة 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).

رقم الحلقة (30) 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 Data Definition Language (DDL)
لغة معالجة البيانات Data Manipulation Language (DML)

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

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 زيادة 1000]
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...

رقم الحلقة (31) 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	٠١٢٢٥٥٠٠٥٧	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؟

رقم الحلقة (32) 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	الشرط الثاني	(الشرط الأول) AND (الشرط الثاني)
خطأ	خطأ	خطأ
صحيح	خطأ	خطأ
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	الشرط الثاني	(الشرط الأول) OR (الشرط الثاني)
خطأ	خطأ	خطأ
صحيح	خطأ	صحيح
خطأ	صحيح	صحيح
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	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
*							

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	٠١٢٢٥٥٠٠٥٧	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؟

رقم الحلقة (33) 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؟

رقم الحلقة (34) 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!

رقم الحلقة (35) 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	٨٧٠٠
	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	٣
	١٩٨١	٢٠	٣
	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.

رقم الحلقة (36) 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...

رقم الحلقة (37) 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.

رقم الحلقة (38) 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:

t2
1
2
3
4
5

t1
0
2
3
6

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...

رقم الحلقة (39) 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.

رقم الحلقة (40) 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 يجب 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.

رقم الحلقة (41) 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 تتجمس 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 ينبغس 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 بستقبل 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 Rd
▶	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، و()Time، و()Now، 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

رقم الحلقة (42) 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). لقد طالت هذه الدورة (السريعة) أكثر مما يجب، لكنني أرجو أن يكون ذلك قد أضاف شيئاً ما إلى المصادر العربية.

هذا أيضاً يختم الجزء الأول من هذه السلسلة من المذكرات. لست أدري متى أكتب الجزء الثاني، لكنني أظن أن هذا الجزء يمثل وحدة قائمة بذاتها، وأن أحداً ليس بحاجة إلى انتظار الجزء الثاني.

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.