ملاحظات للعاملين بلغة

تأليف
مساهمون من
Stack Overflow

أكاديمية حسوب

ملاحظات للعاملين بلغة SQL

اتقن لغة SQL بأمثلة تطبيقية وملاحظات عملية

تأليف

مساهمون من

Stack Overflow

ترجمة

محمد بغات

تحرير

جميل بيلوني

تصميم الغلاف

فرج الشامي

عن الناشر

أُنتج هذا الكتاب برعاية شركة حسوب وأكاديمية حسوب.

أكاديمية حسوب

تهدف أكاديمية حسوب إلى توفير مقالات ودروس عالية الجودة حول مجالات مُختلفة وبلغة عربية فصيحة.

تقدم أكاديمية حسوب دورات شاملة بجودة عالية عن تعلم البرمجة بأحدث تقنياتها تعتمد على التطبيق العملي، مما يؤهل الطالب لدخول سوق العمل بثقة.

تتكامل الأكاديمية مع موسوعة حسوب، التي توفر توثيقًا عربيًا شاملًا مدعمًا بالأمثلة للغات البرمجة.

باب المُساهمة في الأكاديمية مفتوح لكل من يرى في نفسه القدرة على توفير مقالات أو كتب أو مسارات عالية الجودة.

Academy.hsoub.com

شركة حسوب

تهدف حسوب لتطوير الويب العربي وخدمات الإنترنت عن طريق توفير حلول عملية وسهلة الاستخدام لتحديات مختلفة تواجه المستخدمين في العالم العربي.

تشجع حسوب الشباب العربي للدخول إلى سوق العمل عن بعد بتوفيرها منصات عربية للعمل عن بعد، مستقل وخمسات إضافةً إلى موقع بعيد، وكما أنها توفر خدمات للنقاشات الهادفة في حسوب I/O، وخدمة أنا لإدارة المهام وتنظيم العمل.

يعمل في حسوب فريق شاب وشغوف من مختلف الدول العربية. ويمكن معرفة المزيد عن شركة حسوب والخدمات التي تقدمها بزيارة موقعها.

Hsoub.com

جدول المحتويات

ت

تقديم

SQL أو لغة الاستعلامات البنيوية (Structured Query Language) هي لغة برمجة متُخصِّصة تُستخدَم لمعالجة وإدارة قواعد البيانات، وتُنطق سي كويل (See-Quel). تعد اللغة القياسية لأنظمة إدارة قواعد البيانات (RDBMS)، وتُستخدم تعليمات وأوامر SQL لإجراء عمليات مباشرة على البيانات، مثل تخزينها في قاعدة بيانات، وجلبها منها والتعديل عليها بالإضافة إلى إنجاز مهام إدارية على قواعد البيانات من تأمين ونسخ احتياطي وإدارة للمستخدمين.

طوِّرت شركة IBM لغة SQL في بداية السبعينات بمساهمة بويس رايموند (Raymond Boyce) ودونالد شامبرلين (Donald Chamberlin)، وكانت تُسمَى آنذاك SEQUEL. كان الهدف من تطويرها هو إدارة ومعالجة نظام R، وهو نظام شبه علائقي لمعالجة قواعد البيانات (quasi-relational database management system).

سنة 1986، اعتمدت كل من المنظمة الأمريكية للمقاييس (ANSI) والمنظَّمة الدولية للمقاييس (ISO) لغة SQL كمقياس مرجعي، وقد تعاقبت بعد ذلك 9 إصدارات جديدة من المعيار، سنوات: 1989 و 1992 و 1996 و 1999 و 2003 و 2006 و 2008 و 2011 و 2016.

1. لماذا عليك تعلم SQL؟

إن كنت تتساءل عما إذا كانت SQL تستحق أن تتعلمها، فالنقاط التالية ستوضح لك بعض مزايا هذه اللغة:

1. **معالجة البيانات الكبيرة**: نحن نعيش في عصر الثورة الرقمية، ومن نتائج ذلك أنّ البيانات أصبحت متاحة بكميات ضخمة. ففي كل يوم تُنتج عدة تيرابايت من البيانات. بالطبع، يمكنك استخدام جداول البيانات العادية، مثل EXCEL وجداول جوجل، ولكنها مُخصَّص لمعالجة كمية صغيرة أو متوسطة من البيانات. وهنا يأتي دور SQL، لأنَّها مُصمَّمة لمعالجة كميات ضخمة من البيانات بأداء وكفاءة عالية.
2. **تطوير الويب**: SQL هي إحدى المهارات الضرورية لكل مبرمجي الواجهة الخلفية للخادم،

لأنَّها تُستخدَم لمعالجة واسترجاع البيانات المُخزَّنة في الخادم، بما فيها بيانات المستخدمين.

3. **السرعة**: لا تفعل SQL إلا شيئًا واحدًا فقط، وهو معالجة البيانات وإدارتها، وهي ممتازة فيما تفعله. فهي مُحسَّنة للدخول إلى البيانات بسرعة فائقة، ما يجعلها مثالية لتطبيقات الوقت الحقيقي.
4. **فرص العمل**: هناك طلب كبير على مبرمجي SQL في القطاع الخاص، ويُتوقَّع أن يزداد هذا الطلب في السنوات القادمة، خصوصا في العالم العربي الذي يعرف ازدهارا سريعا للاقتصاد الرقمي في السنوات الأخيرة.
5. **الشهرة**: حلت SQL في استطلاع stackoverflow لسنة 2019 في المرتبة الثالثة في قائمة أكثر تقنيات البرمجة استخداما من قبل المبرمجين على مستوى العالم بعد جافاسكربت و HTML و CSS. إذ يستخدمها أكثر من نصف المبرمجين. هذا أمر طبيعي، لأنّ SQL هي إحدى أركان تقنيات الويب، ولا يمكن تطوير أي موقع ويب أو تطبيق بدون معرفة ولو بسيطة بها؛ أضف إلى أنَّها مفتوحة المصدر، ولديها مجتمعًا كبيرًا داعمًا لها.

2. عن الكتاب واستخدامه؟

أطلقت حسوب مشروعًا لترجمة بعض أفضل الكتب التقنية في مجال البرمجة. ونظرًا لأهمية SQL الموضحة آنفًا سواءً للمبرمجين، وحتى لغير المبرمجين من العاملين في القطاعات التقنية، مثل الصناعة والمحاسبة والصيرفة وتحليل البيانات، أو للمهتمين بقواعد البيانات عمومًا، قررنا ترجمة أحد أفضل الكتب الإنجليزية المتقدمة عن SQL، وهو كتاب "SQL Notes For Professionals" المبني على توثيق موقع StackOverflow وقد ساهم في إعداده عدد كبير من المساهمين على شبكة StackOverflow الشهيرة. وإن أردت الاطلاع على قائمة المساهمين الكاملة، ارجع إلى قسم "Credits" في نهاية الكتاب الأصلي، SQL Notes For Professionals.

يغطّي هـذا الكتـاب المفـاهيم الأساسـية للغـة SQL، مثـل العمليـات الأوليـة، وإدراج البيانـات وحذفها واستخلاصها وتحديثها، وأنواع البيانـات، وتصـميم الجـداول وتنفيـذ الاسـتعلامات، إضـافة إلى مفاهيم متقدمة، مثل المعارض والدوال، وإدارة المسـتخدمين، وكيفيـة تـأمين الشـيفرة وغيرهـا من المواضـيع. كمـا أنّ الكتـاب غـني بالأمثلـة التطبيقيـة الـتي تشـرح كـل هـذه المواضـيع لترسـيخ فهمها ترسيخًا.

هذا الكتاب ليس مثـل غـيره من الكتب والشـروحات الـتي تشـرح لغـة SQL من البدايـة شـرحًا مُبسَّطًا ومتسلسلًا وإنَّما يعتمد على مبـدأ خـير الكلام مـا قـل ودل في الشـرح وتـرك الشـيفرة تشـرح نفسها بنفسها، فيحوي على كم كبير من الشيفرات بالموازنة مـع الشـرح. وُجِّه هـذا الكتـاب لمن لديـه معرفة بسيطة بلغـة SQL، لـذا يفضَّـل أن تمتلـك معرفـة بلغـة SQL لتسـتفيد أكبر اسـتفادة من هـذا الكتاب وتقرأ الشيفرات وتفهمها وتتعلَّم منها. في هذه الحالة، سيساهم هذا الكتاب في رفع مسـتواك في لغـة SQL وسـيُملِّكك مهـارات متقدمـة في اسـتعمال لغـة SQL بالإضـافة إلى بعض الخـدع والالتفافات المتقدمة أيضًا. قد تسأل نفسك، هل ينفع أن اقرأ الكتاب دون معرفة مسبقة بلغة SQL؟ سـأقول، نعم، ولكن يجب أن تتحلى بالصـبر في قـراءة الشـيفرة وتحليلهـا وفهمهـا والبحث عن أي موضوع لم تفهمه والسؤال عن شرح لأي شـيفرة غامضـة، إذ لن تجـد كلامًا وشـرحًا كبـيرًا للمواضـع التي يتحدث عنها الكتاب، كما أن تسلسل المواضيع في الكتاب لا تراعي عـدم امتلاك القـارئ معرفـة بلغة SQL، إذ رُتِّبت ترتيبًا عشوائيًا.

بذكر ترتيب عنـاوين ومواضـيع الكتـاب، حـاولت تـرتيب عنـاوين الكتـاب بأنسـب شـكل لتكـون متدرِّجة في الصعوبة وحاولت جمع المواضيع المتشابهة في فصل واحد رغم تشـرذمها وتفرقهـا في الكتاب الأصلي فلا تشبه النسخة العربية النسـخة الأجنبيـة مطلقًا، إذ حـاولت أن تكـون أفضـل منهـا وأرجو أن نكون قد حققنا ذلك. فإن كنت على معرفة بأحد المواضـيع، فلا تتخطاهـا بـل اقرأهـا، فقـد تمر معك إشارة لموضوع متقدم أو ملاحظة مهمـة لم تكن تعرفهـا (تـذكر أنَّ اسـم الكتـاب ملاحظـات

متقدمة ;-)). يمكنك أيضًا أن تقرأ الكتاب من أي قسم تريد فهو من الأساس غير مُرتَّب ترتيبًا متدرجًا ومتسلسلًا كما أشرت إلى ذلك، رغم محاولتي في ترتيبه لك أنسب ترتيب من البداية للنهاية.

أنشئ العمل الأصلي من هذا الكتاب لأغراض تعليمية ولا يتبع إلى أي شركة أو مجموعة رسمية متعلقة بلغة SQL ولا حتى شبكة Stack Overflow، كما أن جميع العلامات التجارية المذكورة في هذا الكتاب تتبع إلى الشركات المالكة لها.

3. ماذا بعد هذا الكتاب

عند الانتهاء من هذا الكتاب، يمكنك الاطلاع على العديد من المقالات العملية في أكاديمية حسوب. أثناء ذلك، يمكنك التنقل بين توثيق لغة SQL في موسوعة حسوب وفصول هذا الكتاب.

يمكن لأيٍّ شخص ملم بالبرمجة أن يساهم في المشاريع مفتوحة المصدر. البرامج مفتوحة المصدر هي برامج متاحة للاستخدام وإعادة التوزيع والتعديل دون قيود. تساعد المساهمة في المشاريع مفتوحة المصدر على تحسين البرامج، عبر ضمان تمثيلها لقاعدة عريضة من المستخدمين. عندما يساهم المستخدمون في المشاريع مفتوحة المصدر، سواء عبر كتابة الشيفرة، أو التوثيق، أو صيانة المجلدات، فإنَّهم يوفرون قيمة مضافة للمشروع، ومجتمع المطورين على العموم.

للحصول على مراجع إضافية عن SQL، أو للمشاركة في نقاشات مع الآخرين، يمكنك الاطلاع على المقالات والأسئلة والدروس عن SQL في أكاديمية حسوب.

إذا كنت مهتمًا بتعلم تطوير تطبيقات الويب أو تطبيقات الجوال، أو تعلم لغات محدَّدة مثل روبي وجافاسكربت، فاطلع على قسم الدورات في الأكاديمية، كما يمكنك تصفح موسوعة حسوب لأجل قراءة توثيقات عدد كبير من لغات البرمجة باللغة العربية.

جميل بيلوني

20 - أغسطس - 2020

1 مدخل إلى SQL

لغة الاستعلامات الهيكلية SQL (اختصارًا إلى Structured Query Language) هي لغة برمجة مُتخصِّصة في إدارة قواعد البيانات العلائقية RDBMS (اختصارًا إلى Relational database management system) كما تُستخدَم اللغات المشتقة من SQL في أنظمة إدارة مجاري البيانات العلائقية RDSMS (اختصارًا إلى Relational Data Stream Management Systems)، أو في إدارة قواعد بيانات NoSQL (التوجُّه عديم النموذج [Model-less]).

تتألف SQL من ثلاث لغات فرعية أساسية، وهي:

1. **لغة تعريف البيانات DDL**: تُستعمل لإنشاء وتعديل بنية قاعدة البيانات.
2. **لغة معالجة البيانات DML**: تستعمل لتنفيذ عمليات قراءة البيانات وإدراجها وتحديثها وحذفها.
3. **لغة التحكم في البيانات DCL**: تُستعمل للتحكم في الوصول إلى البيانات المُخزَّنة في قاعدة البيانات.

تتألف لغة DML (أي Data Manipulation Language) من أربع عمليات أساسية، وهي عمليات الإنشاء (Create) والقراءة (Read) والتحديث (Update) والحذف (Delete)، ويُطلق عليها اختصارًا CRUD، إذ تُنفَّذ هذه العمليات عبر التعليمات `INSERT` و `SELECT` و `UPDATE` و `DELETE` على التوالي. أُضيفت في الآونة الأخيرة تعليمة `MERGE`، والتي تُنفِّذ العمليات `INSERT` و `UPDATE` و `DELETE` معًا.

تُقدَّم العديد من قواعد بيانات SQL على هيئة نُظم عميل / خادم (client/server systems). ويُطلق على هذه الخوادم مصطلح "خادم SQL". أنشأت مايكروسوفت قاعدة بيانات تسمى "SQL Server". ورغم كونها إحدى لهجات SQL، إلا أنَّنا لن نتحدث عنها في هذه السلسلة، وإن كنت تريد تَعلُّمها فيمكنك الرجوع إلى توثيقها.

على صعيد آخر، يجب التفريق بين لغة SQL وأنظمة معالجة قواعد البيانات العلائقية

(Relational Database Management System) هي برامج تُستخدَم لمعالجة وإدارة قواعد البيانات العلائقية (Relational Database)، وهي قواعد تُخزِّن البيانات وفق بنية مهيكلة في جداول تتألف من صفوف وأعمدة لتسهيل الوصول إلى القيم المخزنة. لكل جدول مفتاح فريد يميز كل صف من الجدول. وتُسمى «علائقية» (relational) لأنَّ القيم المُخزّنة في الجداول متعلقة ببعضها بعضًا.

تجري أنظمة قواعد البيانات العديد من المهام، مثل:

- تأمين البيانات
- إنشاء النسخ الاحتياطية
- إدارة ومعالجة كميات ضخمة من البيانات
- تصدير البيانات أو استيرادها
- العمل على عدة جداول تزامنيا

هناك العديد من أنظمة معالجة قواعد البيانات، من أشهرها: Oracle و MySQL و Microsoft SQL Server و DB2. ورغم أنَّ أكثرها تستخدم SQL، إلا أنَّ لكل منها بعض الإضافات والصياغات الخاصة بها التي لا تُستخدم في الأنظمة الأخرى، بيْد أنّها تدعم جميعا الأوامر الأساسية للغة (`SELECT` و `UPDATE` و `DELETE` و `INSERT` و `WHERE`).

1. المُعرِّفات (identifiers)

يستعرض هذا القسم موضوع المُعرِّفات (identifiers)، وتشرح قواعد تسمية الجداول والأعمدة وباقي كائنات قاعدة البيانات. سنحاول أن تغطي الأمثلة الاختلافات بين تقديمات SQL المختلفة.

ا. المعرفات غير المقتبسة

يمكن أن تحتوي المُعرِّفات غير المُقتبسة (Unquoted identifiers) على الحروف (a - z) والأرقام (0 - 9) والشرطة السفلية (_)، وفي جميع الأحوال، ينبغي أن تبدأ بحرف.

اعتمادًا على تقديم أو لهجة SQL المُستخدَمة، و / أو إعدادات قاعدة البيانات، قد يجوز استخدام أحرف أخرى، وبعضها يمكن أن تُستخدَم حرفًا أولًا للمعرِّف.

هذه بعض الأمثلة على الأحرف الجائزة:

- MS SQL: المحارف @ و $ و # وباقي محارف اليونيكود (Unicode) الأخرى، المصدر.
- MySQL: المحرف $، المصدر.
- Oracle: المحرفان $ و # وباقي المحارف من مجموعة محارف قاعدة البيانات، المصدر.
- PostgreSQL: المحرف $ وباقي محارف اليونيكود الأخرى، المصدر.

المُعرِّفات غير المقتبسة غير حسَّاسة لحالة الأحرف عمومًا. بيْد أنَّ طريقة التعامل مع حالة الأحرف تختلف بحسب تقديم SQL، فمثلًا:

- MS SQL: تحافظ على الحالة (Case-preserving)، إذ تُحدَّد مسألة الحساسية لحالة الأحرف عبر مجموعة أحرف قاعدة البيانات (database character set)، لذا يمكن أن تكون حساسة لحالة الأحرف.
- MySQL: تحافظ على الحالة، وتعتمد الحساسية على إعدادات قاعدة البيانات ونظام الملفات الأساسي.
- Oracle: تُحوِّل المحارف إلى محارف كبيرة، ثم تُعامَل على أنَّها معرِّفات مقتبسة.
- PostgreSQL: تُحوِّل المحارف إلى الحالة الصغيرة، ثم تُعامَل مثل المعرِّفات المقتبسة.
- SQLite: تحافظ على الحالة، وتقتصر عدم حساسيتها على محارف ASCII.

2. أنواع البيانات

ا. DECIMAL و NUMERIC

يمثِّل النوعان DECIMAL و NUMERIC أعدادًا عشرية ذات دقة ثابتة، وهما متكافئان وظيفيًا ويُصاغان على النحو التالي:

```
DECIMAL ( precision [ , scale] )
NUMERIC ( precision [ , scale] )
```

أمثلة:

```
SELECT CAST(123 AS DECIMAL(5,2))       -- 123.00
SELECT CAST(12345.12 AS NUMERIC(10,5)) -- 12345.12000
```

ب. FLOAT و REAL

يمثل نوعا الأعداد FLOAT و REAL الأعداد التقريبيَّة، ويُستخدمَان لتمثيل البيانات العددية العشرية ذات الفاصلة (floating point numeric data).

```
SELECT CAST( PI() AS FLOAT) -- 3.14159265358979
SELECT CAST( PI() AS REAL)  -- 3.141593
```

ج. Integers

يمثل النوع Integers البيانات العددية الصحيحة وتتفرع منه الأنواع التالية بحسب الحجم المطلوب:

نوع البيانات	النطاق	مساحة التخزين
bigint	$(-2^{63} , 2^{63}-1)$ -9,223,372,036,854,775,808 ,) (9,223,372,036,854,775,807	8 بايتات
int	$(-2^{31} , 2^{31}-1)$ (-2,147,483,648 , 2,147,483,647)	4 بايتات
smallint	$(-2^{15} , 2^{15}-1)$ (32,767 إلى -32,768)	2 بايت
tinyint	(0 , 255)	1 بايت

د. MONEY و SMALLMONEY

يمثِّل النوعــان MONEY و SMALLMONEY البيانــات الــتي تُحــدَّد على أنَّهــا قيم نقدية أو عملات.

نوع البيانات	النطاق	مساحة التخزين
money	-922,337,203,685,477.5808 , 922,337,203,685,477.5807	8 بايتات
smallmoney	-214,748.3648 , 214,748.3647	4 بايتات

ه. BINARY و VARBINARY

يمثــل النوعــان BINARY و VARBINARY البيانــات الثنائيــة ذات الطــول الثــابت أو المتغــير، ويُصاغان على النحو التالي:

```
BINARY [ ( n_bytes ) ]
VARBINARY [ ( n_bytes | max ) ]
```

يمكن أن يكون n_bytes أي عدد محصور بين 1 إلى 8000 بايت، وتشير قيمة max إلى أنَّ الحد الأقصى لمساحة التخزين هو $2^{31}-1$.

أمثلة:

```
SELECT CAST(12345 AS BINARY(10))    -- 0x0000000000000003039
SELECT CAST(12345 AS VARBINARY(10)) -- 0x00003039
```

و. CHAR و VARCHAR

يمثِّل النوعان CHAR و VARCHAR البيانات النصيَّة ذات الطول الثابت والمتغير على التوالي، ويُصاغان على النحو التالي:

```
CHAR [ ( n_chars ) ]
VARCHAR [ ( n_chars ) ]
```

إليك الأمثلة التالية:

```
SELECT CAST('ABC' AS CHAR(10)) -- 'ABC       '
SELECT CAST('ABC' AS VARCHAR(10)) -- 'ABC'
SELECT CAST('ABCDEFGHIJKLMNOPQRSTUVWXYZ' AS CHAR(10)) --
'ABCDEFGHIJ'
```

لاحظ أنَّه في حالة استعمال CHAR، فإنَّ الحجم المطلوب سيُحجَز سواءً كان البيانات المراد تخزينا أقل من ذلك أو أكثر، فلاحظ في الحالة الأولى كيف حُشِّيت السلسلة النصية المُؤلَّفة من ثلاثة حروف بمسافات فارغة أضيفت إلى يمينها بينما اقتطعت السلسلة في الحالة الثالثة إلى لتشمل أول 10 حروف فقط. على النقيض، أخذت السلسلة النصية في الحالة الثانية الحجم المطلوب والكافي للتتَّسع فيه وهو حجم 3 حروف فقط.

ز. NCHAR و NVARCHAR

يمثـل النوعـان NCHAR و NVARCHAR نصـوص اليونيكـود ذات الطـول الثـابت أو المتغـير، ويُصاغان على النحو التالي:

```
NCHAR [ ( n_chars ) ]
NVARCHAR [ ( n_chars | MAX ) ]
```

استخدم MAX لأجل السلاسل النصية الطويلة التي يمكن أن تتجاوز 8000 حرفًا.

ح. UNIQUEIDENTIFIER

يمثـل هـذا النـوع مُعرِّفـا عموميًـا فريـدًا (Universally Unique IDentifier أو UUID) أو معرِّفًا عامًا فريدًا (globally unique identifier أو GUID) مُخزَّنا في 16 بايت.

```
DECLARE @GUID UNIQUEIDENTIFIER = NEWID();
SELECT @GUID -- 'E28B3BD9-9174-41A9-8508-899A78A33540'
DECLARE @bad_GUID_string VARCHAR(100) = 'E28B3BD9-9174-41A9-
8508-899A78A33540_foobarbaz'
SELECT
 @bad_GUID_string, -- 'E28B3BD9-9174-41A9-8508-
899A78A33540_foobarbaz'
 CONVERT(UNIQUEIDENTIFIER, @bad_GUID_string) -- 'E28B3BD9-9174-
41A9-8508-899A78A33540'
```

3. القيمة NULL

تمثِّل الكلمة المفتاحية NULL في SQL، وكذلك في لغات البرمجة الأخرى، القيمـة المعدومـة أو الغائبة أي اللا قيمة أو اللا شيء. وتُستخدَم عادة في SQL للإشارة إلى "عدم وجود قيمة".

من المهم التمييز بينها وبين القيم الفارغة، مثل السلسـلة النصـية الفارغـة '' أو الـرقم 0، إذ لا يُعدُّ أيٌّ منهما في الواقـع معـدومًا (NULL). من المهم أيضًـا تَجنُّب إحاطـة NULL بعلامـات الاقتبـاس،

على شاكلة 'NULL'، والتي يمكن استخدامها في الأعمدة التي تقبل القيم النصية، إذ لا تُمثِّل في هذه الحالة القيمة NULL، ويمكن أن تُسبِّب أخطاءً، وتفسد البيانات.

ا. ترشيح NULL في الاستعلامات

تختلف صياغة ترشيح NULL (أي عدم وجود قيمة) في كتل WHERE عن ترشيح القيم الأخرى:

```
SELECT * FROM Employees WHERE ManagerId IS NULL ;
SELECT * FROM Employees WHERE ManagerId IS NOT NULL ;
```

لاحظ أنَّه لمَّا لم تكن NULL مساوية لأيِّ شيء آخر، ولا حتى لنفسها، فستُعيِد عوامل الموازنة = NULL أو <> NULL (أو != NULL) دائمًا القيمة المنطقية الخاصة UNKNOWN، والتي ترفضها WHERE. فترشِّح WHERE كل الصفوف التي يساوي شرطها القيمة FALSE أو UNKNOWN، ولا تحتفظ إلا بالصفوف ذات الشرط الصحيح (TRUE).

ب. الأعمدة المعدومة في الجداول

عند إنشاء الجداول، يمكن جعل العمود قابلًا بأن لا يأخذ قيمة (nullable) أو يجب أن يأخذ قيمة وذلك بالشكل التالي:

```
CREATE TABLE MyTable
(
    MyCol1 INT NOT NULL, -- non-nullable
    MyCol2 INT NULL -- nullable
);
```

افتراضيًا، يمكن لجميع الأعمدة أن لا تأخذ قيمة (باستثناء تلك الموجودة في قيد المفتاح الأساسي - primary key constraint) ويلغى هذا الشرط بإضافة القيد NOT NULL صراحةً للعمود. وسينتج خطأ عن محاولة تعيين NULL لعمود لعمود لا يمكن أن يبقى فارغًا دون قيمة.

```
INSERT INTO MyTable (MyCol1, MyCol2) VALUES (1, NULL) ; -- صحيح
INSERT INTO MyTable (MyCol1, MyCol2) VALUES (NULL, 2) ;
```

لا يمكن إدراج القيمة NULL في العمود MyCol1 في الجدول MyTable لأنَّ العمود لا يقبل أن يبقى عديم القيمة، لذا ستفشل عملية الإدراج INSERT الثانية.

ج. إسناد NULL إلى حقل

إسناد القيمة NULL إلى حقل يشبه إسناد أيِّ قيمة أخرى:

```
UPDATE Employees
SET ManagerId = NULL
WHERE Id = 4
```

د. إدراج الصفوف التي تحتوي حقولًا عديمة القيمة (NULL fields)

إدراج بيانات موظف بدون رقم هاتف، وبدون مدير في جدول الموظفين Employees يمكن أن يجرى بالشكل التالي:

```
INSERT INTO Employees
 (Id, FName, LName, PhoneNumber, ManagerId, DepartmentId,
Salary, HireDate)
VALUES
 (5, 'Jane', 'Doe', NULL, NULL, 2, 800, '2016-07-22');
```

4. المفاتيح الرئيسية (Primary Keys)

تُستخدَم المفاتيح الرئيسية لتمييز صفوف جدول من قاعدة بيانات. ولا يُسمَح إلا بمفتاح رئيسي واحد في كل جدول.

تُنشِئ الشيفرة التالية جدولًا للموظفين، مع جعل الحقل Id مفتاحه الرئيسي:

```
CREATE TABLE Employees (
    Id int NOT NULL,
    PRIMARY KEY (Id),
    ...
);
```

يمكن أيضًا تركيب مفتاح من حقل واحد أو أكثر، ويسمى هذا النوع من المفاتيح المفاتيح المُركَّبة، وتُصَاغ على النحو التالي:

```
CREATE TABLE EMPLOYEE (
    e1_id INT,
    e2_id INT,
   PRIMARY KEY (e1_id, e2_id)
)
```

ا. الزيادة التلقائية لقيمة حقل

تسمح العديد من قواعد البيانات بزيادة قيمة المفتاح الرئيسي تلقائيًا عند إضافة مفتاح جديد، فيضمن هذا السلوك أن تكون كل المفاتيح فريدة ومختلفة عن بعضها.

إليك الأمثلة التوضيحية التالية:

- MySQL

```
CREATE TABLE Employees (
    Id int NOT NULL AUTO_INCREMENT,
    PRIMARY KEY (Id)
);
```

- PostgreSQL

```
CREATE TABLE Employees (
    Id SERIAL PRIMARY KEY
);
```

- SQL Server

```
CREATE TABLE Employees (
    Id int NOT NULL IDENTITY,
    PRIMARY KEY (Id)
);
```

- SQLite

```
CREATE TABLE Employees (
    Id INTEGER PRIMARY KEY
);
```

5. المفاتيح الأجنبية (Foreign Keys)

تضمن قيود المفاتيح الأجنبية أو الخارجية (Foreign Keys constraints) تكامل البيانات، إذ تفرض أن تتطابق القيم الموجودة في جدول مُعيَّن، مع القيم المقابلة في جدول آخر.

مثلا، في الجامعة، تنتمي كل دورة دراسية إلى قسم مُعيَّن. يمكننا التعبير عن هذا القيد (constraint) على النحو التالي:

```
CREATE TABLE Department (
    Dept_Code        CHAR (5) PRIMARY KEY,
    Dept_Name        VARCHAR (20) UNIQUE
);
```

يدرج المثال التالي قيمًا جديدةً في قسم علوم الحاسوب:

```
INSERT INTO Department VALUES ('CS205', 'Computer Science');
```

يحتوي الجدول التالي على معلومات عن المواضيع التي تشملها شعبة علوم الحاسوب:

```
CREATE TABLE Programming_Courses (
    Dept_Code       CHAR(5),
    Prg_Code        CHAR(9) PRIMARY KEY,
    Prg_Name        VARCHAR (50) UNIQUE,
```

```
    FOREIGN KEY (Dept_Code) References Department(Dept_Code));
```

(يجب أن يتطابق نوع بيانات المفتاح الأجنبي مع نوع البيانات الخاص بالمفتاح المشار إليه [referenced key].)

لا يسمح قيد المفتاح الأجنبي الخاص بالعمود `Dept_Code` إلا بالقيم الموجودة سلفًا في الجدول المشار إليه. هذا يعني أنه إذا حاولت إدراج القيم التالية:

```
INSERT INTO Programming_Courses Values ('CS300', 'FDB-DB001',
'Database Systems');
```

فستطرح قاعدة البيانات «خطأ انتهاك قيد المفتاح الأجنبي» (Foreign Key violation error)، لأنَّ `CS300` غير موجودة في جدول الأقسام `Department`. ولكن لن يكون هناك أيَّة مشكلة عند تجربة قيمة مفتاح موجود مثل:

```
INSERT INTO Programming_Courses VALUES ('CS205', 'FDB-DB001',
'Database Systems');
INSERT INTO Programming_Courses VALUES ('CS205', 'DB2-DB002',
'Database Systems II');
```

هذه بعض النصائح حول كيفية استخدام المفاتيح الأجنبية:

- يجب أن يشير المفتاح الخارجي إلى مفتاح فريد `UNIQUE` (أو أساسي `PRIMARY`) من الجدول الأصلي الأب (parent table).
- لن ينجم أيَّ خطأ عن إدخال القيمة المعدومة `NULL` إلى عمود المفتاح الخارجي.
- يمكن أن تشير قيود المفاتيح الأجنبية إلى الجداول الموجودة في نفس قاعدة البيانات.
- يمكن أن تشير قيود المفتاح الأجنبية إلى عمود آخر في نفس الجدول (مرجع ذاتي).

سنعمل في المثال التالي على إنشاء جدول بمفتاح أجنبي؛ لدينا جدول البيانات SuperHeros الذي يحتوي على مفتاح أساسي `ID`.

سنضيف جدولًا جديدًا بُغيَة تخزين صلاحيات كل بطل خارق:

```
CREATE TABLE HeroPowers
(
    ID int NOT NULL PRIMARY KEY,
    Name nvarchar(MAX) NOT NULL,
    HeroId int REFERENCES SuperHeros(ID)
)
```

في هذا المثال، يُعدُّ العمود HeroId مفتاحًا أجنبيًا للجدول SuperHeros.

6. التعليقات

هناك نوعان من التعليقات في SQL، التعليقات السطرية، والتعليقات متعدِّدة الأسطر.

التعليقات السطرية (Single-line comments) هي تعليقات تستمر حتى نهاية السطر، وتُسبَق بالرمز --:

```
SELECT *
FROM Employees -- هذا تعليق
WHERE FName = 'John'
```

بينما تُوضع التعليقات متعددة الأسطر (Multi-line comments) داخل /* ... */:

```
/* يعيد هذا الاستعلام
 جميع الموظفين */
SELECT *
FROM Employees
```

يجوز أيضًا إدراج مثل هذا التعليق في منتصف السطر:

```
SELECT /* جميع الأعمدة: */ *
FROM Employees
```

7. أمثلة على قواعد البيانات والجداول

إليك بعض الأمثلة التوضيحية عن قواعد البيانات.

ا. قاعدة بيانات متجر السيارات

سوف نستعرض في المثال التالي قاعدة بيانات لمتجر يبيع السيارات، إذ سنُخزِّن فيها قوائم تدمجُ الأقسام والموظفين والعملاء وسيارات العملاء. وسنستخدم المفاتيح الخارجية (foreign keys) لإنشاء علاقات بين مختلف الجداول (هذا تطبيق حي للمثال).

العلاقات بين الجداول:

- يحوي كل قسم 0 موظف أو أكثر،
- ولكل موظف مدير واحد أو أكثر،
- وقد يكون لكل عميل 0 سيارة أو أكثر

الجدول Departments:

Name	Id
HR	1
Sales	2
Tech	3

لننشئ الجدول عبر SQL:

```
CREATE TABLE Departments (
    Id INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(25) NOT NULL,
 PRIMARY KEY(Id)
```

```
);
INSERT INTO Departments
    ([Id], [Name])
VALUES
   (1, 'HR'),
   (2, 'Sales'),
   (3, 'Tech')
;
```

الجدول Employees:

HireDate	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
01-01-2002	1000	1	NULL	1234567890	Smith	James	1
23-03-2005	400	1	1	2468101214	Johnson	John	2
12-05-2009	600	2	1	1357911131	Williams	Michael	3
24-07-2016	500	1	2	1212121212	Smith	Johnathon	4

لننشئ الجدول:

```
CREATE TABLE Employees (
   Id INT NOT NULL AUTO_INCREMENT,
   FName VARCHAR(35) NOT NULL,
   LName VARCHAR(35) NOT NULL,
   PhoneNumber VARCHAR(11),
   ManagerId INT,
   DepartmentId INT NOT NULL,
```

```
Salary INT NOT NULL,
   HireDate DATETIME NOT NULL,
   PRIMARY KEY(Id),
   FOREIGN KEY (ManagerId) REFERENCES Employees(Id),
   FOREIGN KEY (DepartmentId) REFERENCES Departments(Id)
);
INSERT INTO Employees
   ([Id], [FName], [LName], [PhoneNumber], [ManagerId],
[DepartmentId], [Salary], [HireDate])
VALUES
   (1, 'James', 'Smith', 1234567890, NULL, 1, 1000, '01-01-
2002'),
   (2, 'John', 'Johnson', 2468101214, '1', 1, 400, '23-03-
2005'),
   (3, 'Michael', 'Williams', 1357911131, '1', 2, 600, '12-05-
2009'),
   (4, 'Johnathon', 'Smith', 1212121212, '2', 1, 500, '24-07-
2016')
```

الجدول Customers:

PreferredContact	PhoneNumber	Email	LName	FName	Id
PHONE	3347927472	william.jones@example.com	Jones	William	1
EMAIL	2137921892	dmiller@example.net	Miller	David	2
EMAIL	NULL	richard0123@example.com	Davis	Richard	3

لننشئ الجدول ونضيف بعض البيانات إليه:

```
CREATE TABLE Customers (
    Id INT NOT NULL AUTO_INCREMENT,
    FName VARCHAR(35) NOT NULL,
    LName VARCHAR(35) NOT NULL,
```

```
    Email varchar(100) NOT NULL,
    PhoneNumber VARCHAR(11),
    PreferredContact VARCHAR(5) NOT NULL,
 PRIMARY KEY(Id)
);
INSERT INTO Customers
     ([Id], [FName], [LName], [Email], [PhoneNumber],
[PreferredContact])
VALUES
     (1, 'William', 'Jones', 'william.jones@example.com',
'3347927472', 'PHONE'),
     (2, 'David', 'Miller', 'dmiller@example.net',
'2137921892', 'EMAIL'),
     (3, 'Richard', 'Davis', 'richard0123@example.com', NULL,
'EMAIL')
;
```

الجدول Cars:

Total Cost	Status	Model	EmployeeId	CustomerId	Id
230	READY	Ford F-150	2	1	1
200	READY	Ford F-150	2	1	2
100	WAITING	Ford Mustang	1	2	3
1254	WORKING	Toyota Prius	3	3	4

إليك تعليمات SQL لإنشاء الجدول وإضافة بيانات إليه:

```
CREATE TABLE Cars (
    Id INT NOT NULL AUTO_INCREMENT,
    CustomerId INT NOT NULL,
    EmployeeId INT NOT NULL,
    Model varchar(50) NOT NULL,
```

```
 Status varchar(25) NOT NULL,
    TotalCost INT NOT NULL,
 PRIMARY KEY(Id),
 FOREIGN KEY (CustomerId) REFERENCES Customers(Id),
 FOREIGN KEY (EmployeeId) REFERENCES Employees(Id)
);

INSERT INTO Cars
    ([Id], [CustomerId], [EmployeeId], [Model], [Status],
[TotalCost])
VALUES
     ('1', '1', '2', 'Ford F-150', 'READY', '230'),
     ('2', '1', '2', 'Ford F-150', 'READY', '200'),
     ('3', '2', '1', 'Ford Mustang', 'WAITING', '100'),
     ('4', '3', '3', 'Toyota Prius', 'WORKING', '1254')
;
```

ب. قاعدة بيانات لمكتبة

سننشئ في هذا المثال قاعدة بيانات خاصة بمكتبة، فستحتوي على جداول لتخزين المـؤلفين Authors والكتب Books وثالث يربط بينهما BooksAuthors. هنا تجد مثالًا حيًا للقاعدة.

يُعرَّف جدولَا المؤلفين والكتب بالجداول الأساسية (base tables)، لحوايتهما على تعريـف العمـود، وكـذا البيانـات الخاصـة بالكيانـات الفعليـة في النمـوذج العلائقي (relational model). ويُعرَّف الجدول BookAuthors باسم جدول العلاقة (relationship table)، لأنَّه يحـدِّد العلاقـة بين جدول الكتب Books والمؤلفين Authors.

العلاقات بين الجداول:

- يمكن أن يكون لكل مؤلف كتاب واحد أو أكثر.
- كل كتاب يمكن أن يكون له مؤلف واحد أو أكثر

الجدول Authors (عرض الجدول):

Country	Name	Id
USA	J.D. Salinger	1
USA	F. Scott. Fitzgerald	2
UK	Jane Austen	3
USA	Scott Hanselman	4
USA	Jason N. Gaylord	5
India	Pranav Rastogi	6
USA	Todd Miranda	7
USA	Christian Wenz	8

لننشئ الجدول الآن ونضيف إليه البيانات السابقة:

```
CREATE TABLE Authors (
    Id INT NOT NULL AUTO_INCREMENT,
    Name VARCHAR(70) NOT NULL,
    Country VARCHAR(100) NOT NULL,
 PRIMARY KEY(Id)
);

INSERT INTO Authors
     (Name, Country)
VALUES
     ('J.D. Salinger', 'USA'),
     ('F. Scott. Fitzgerald', 'USA'),
     ('Jane Austen', 'UK'),
     ('Scott Hanselman', 'USA'),
     ('Jason N. Gaylord', 'USA'),
     ('Pranav Rastogi', 'India'),
```

```
     ('Todd Miranda', 'USA'),
     ('Christian Wenz', 'USA')
;
```

الجدول Books (عرض الجدول):

Title	Id
The Catcher in the Rye	1
Nine Stories	2
Franny and Zooey	3
The Great Gatsby	4
Tender id the Night	5
Pride and Prejudice	6
Professional ASP.NET 4.5 in C# and VB	7

عبارات SQL لإنشاء الجدول وإضافة البيانات إليه:

```
CREATE TABLE Books (
    Id INT NOT NULL AUTO_INCREMENT,
    Title VARCHAR(50) NOT NULL,
 PRIMARY KEY(Id)
);

INSERT INTO Books
     (Id, Title)
VALUES
     (1, 'The Catcher in the Rye'),
     (2, 'Nine Stories'),
     (3, 'Franny and Zooey'),
```

```
    (4, 'The Great Gatsby'),
    (5, 'Tender id the Night'),
    (6, 'Pride and Prejudice'),
    (7, 'Professional ASP.NET 4.5 in C# and VB')
;
```

الجدول BooksAuthors (عرض الجدول):

AuthorId	BookId
1	1
1	2
1	3
2	4
2	5
3	6
4	7
5	7
6	7
7	7
8	7

تعليمات SQL لإنشاء الجدول وإضافة البيانات إليه:

```
CREATE TABLE BooksAuthors (
    AuthorId INT NOT NULL,
```

```
    BookId  INT NOT NULL,
 FOREIGN KEY (AuthorId) REFERENCES Authors(Id),
 FOREIGN KEY (BookId) REFERENCES Books(Id)
);

INSERT INTO BooksAuthors
    (BookId, AuthorId)
VALUES
     (1, 1),
     (2, 1),
     (3, 1),
     (4, 2),
     (5, 2),
     (6, 3),
     (7, 4),
     (7, 5),
     (7, 6),
     (7, 7),
     (7, 8)
;
```

الآن، إن أردت عرض جميع المؤلفين، فاكتب ما يلي (تجربة حية):

```
SELECT * FROM Authors;
```

عرض جميع عناوين الكتب (تجربة حية):

```
SELECT * FROM Books;
```

عرض جميع الكتب ومؤلفيها (تجربة حية):

```
SELECT
   ba.AuthorId,
   a.Name AuthorName,
   ba.BookId,
   b.Title BookTitle
FROM BooksAuthors ba
```

```
    INNER JOIN Authors a ON a.id = ba.authorid
    INNER JOIN Books b ON b.id = ba.bookid
;
```

ج. جدول الدول

سننشئ في هذا المثال جدولًا للبلدان، إذ يُستخدَم في العديد من المجالات، وخاصة في التطبيقات المالية التي تشمل العملات وأسعار الصرف (تجربة حية للمثال).

تطلب بعض البرمجيات الخاصة بتحليل الأسواق مثل بلومبرج ورويترز أن تعطيهم رمزًا مؤلفًا من حرفين أو ثلاث يمثل الدولة، إلى جانب رمز العملة. يحتوي الجدول التالي على عمود يحتوي رموز `ISO` المؤلَّفة من حرفين، وكذلك على عمود يحتوي رموز `ISO3` المكونة من 3 أحرف، والتي تمثِّل الدول.

الجدول **Countries** (عرض الجدول):

CurrencyCode	ContinentCode	Capital	CountryName	ISONumeric	ISO3	ISO	Id
AUD	OC	Canberra	Australia	36	AUS	AU	1
EUR	EU	Berlin	Germany	276	DEU	DE	2
INR	AS	New Delhi	India	356	IND	IN	2
LAK	AS	Vientiane	Laos	418	LAO	LA	3
USD	NA	Washington	United States	840	USA	US	4
ZWL	AF	Harare	Zimbabwe	716	ZWE	ZW	5

لننشئ جدول الدول في SQL:

```
CREATE TABLE Countries (
    Id INT NOT NULL AUTO_INCREMENT,
    ISO VARCHAR(2) NOT NULL,
    ISO3 VARCHAR(3) NOT NULL,
    ISONumeric INT NOT NULL,
    CountryName VARCHAR(64) NOT NULL,
    Capital VARCHAR(64) NOT NULL,
    ContinentCode VARCHAR(2) NOT NULL,
    CurrencyCode VARCHAR(3) NOT NULL,
 PRIMARY KEY(Id)
)
;

INSERT INTO Countries
    (ISO, ISO3, ISONumeric, CountryName, Capital,
ContinentCode, CurrencyCode)
VALUES
    ('AU', 'AUS', 36, 'Australia', 'Canberra', 'OC', 'AUD'),
    ('DE', 'DEU', 276, 'Germany', 'Berlin', 'EU', 'EUR'),
    ('IN', 'IND', 356, 'India', 'New Delhi', 'AS', 'INR'),
    ('LA', 'LAO', 418, 'Laos', 'Vientiane', 'AS', 'LAK'),
    ('US', 'USA', 840, 'United States', 'Washington', 'NA',
'USD'),
    ('ZW', 'ZWE', 716, 'Zimbabwe', 'Harare', 'AF', 'ZWL')
;
```

2

الاستعلام عن البيانات عبر SELECT

تُستخدَم العبارة SELECT في معظم استعلامات SQL وتتحكم في تحديد النتائج التي يجب أن يعيدها الاستعلام، وتُستخدَم عادةً مع العبارة FROM، والتي تحدد جزءًا (أو أجزاء) من قاعدة البيانات المُستعلَم عنها.

1. اختيار جميع الأعمدة عبر *

إليك قاعدة البيانات التالية المؤلفة من الجدولين التاليين:

جدول الموظفين Employees:

DeptId	LName	FName	Id
3	Smith	James	1
4	Johnson	John	2

جدول الأقسام Departments:

Name	Id
Sales	1
Marketing	2
Finance	3
IT	4

ا. عبارة select بسيطة

يمثل الرمز * محرف بدل، ويُستخدَم لاختيار جميع الأعمدة المتاحة في الجدول. عند استخدامه بديلًا عن الأسماء الصريحة للأعمدة، تُعَاد جميع الأعمدة في جميع الجداول التي يحدِّدها الاستعلام FROM. ينطبق هذا الأمر على جميع الجداول التي يصل إليها الاستعلام عبر عبارات JOIN.

إليك الاستعلام التالي:

```
SELECT * FROM Employees
```

سيعيد الاستعلام أعلاه جميع الحقول من جميع صفوف جدول Employees:

DeptId	LName	FName	Id
3	Smith	James	1
4	Johnson	John	2

ب. الصياغة النقطية (Dot notation)

يمكن تطبيق حرف البدل * على الجدول باستخدام الصياغة النقطية لاختيار كل الأعمدة من جدول مُحدَّد مثل:

```
SELECT
    Employees.*,
    Departments.Name
FROM
    Employees
JOIN
    Departments
 ON Departments.Id = Employees.DeptId
```

سيعيد هذا المثال مجموعة بيانات تحتوي كافة الحقول الموجودة في الجدول Employee، متبوعةً بالحقل Name من الجدول Departments:

Name	DeptId	LName	FName	Id
Finance	3	Smith	James	1
IT	4	Johnson	John	2

ج. متى يمكنك استخدام حرف البدل *؟

يُفضل عمومًا تجنُّب استخدام * في شيفرة الإنتاج، لكن لا مشكلة في استخدامها كاختصار عند تنفيذ الاستعلامات اليدوية في قاعدة البيانات عند العمل على النماذج الأولية. كما قد يفرض عليك تصميم التطبيق أحيانًا استخدام حرف البدل (في مثل هذه الظروف، يُفضل استخدام `tablealias.*` بدلًا من * حيثما أمكن ذلك).

عند استخدام `EXISTS`، كما في:

```
SELECT A.col1, A.Col2 FROM A WHERE EXISTS (SELECT * FROM B
where A.ID = B.A_ID)
```

فذلك لن يعيد أيَّ بيانات من B. وبالتالي لن يكون الدمجُ (join) ضروريًا، كما أنَّ مُحرِّك (engine) قاعدة البيانات يعلم أنَّه لن تُعاد أيَّ قيمة من B، وبالتالي لن يتأثَّر الأداء جرَّاء استخدام *. من جهة أخرى، لا بأس في استخدام `COUNT(*)`، لأنها لا تُعيد أيًا من الأعمدة فعليًا، إذ تحتاج فقط إلى قراءة ومعالجة الأعمدة المستخدمة في عملية الترشيح.

على أي حال، يوصى بتجنُّب استخدام المحرف * في شيفرة الإنتاج، إذ يمكن أن تتسبَّب في مجموعة من المشاكل، منها:

1. حمل الدخل والخرج الزائد (Excess IO)، والحمل الزائد على الشبكة، واستنزاف الذاكرة، وغيرها، وذلك بسبب أنَّ محرك قاعدة البيانات سيقرأ بيانات غير مطلوبة وينقلها إلى شيفرة الواجهة الأمامية. وقد يصبح الأمر أسوأ إن كانت هناك حقولًا كبيرة الحجم، مثل تلك المستخدمة لتخزين الملاحظات الطويلة أو الملفات المرفقة.
2. زيادة الضغط على الدخل والخرج إذا احتاجت قاعدة البيانات إلى تخزين النتائج الداخلية على القرص كجزء من عملية معالجة استعلامات أكثر تعقيدًا من العبارة البسيطة `SELECT <columns> FROM <table>`
3. معالجة زائدة (و / أو مزيد من عمليات الدخل والخرج IO) إذا كانت هناك أعمدة غير

ضرورية من نوع الأعمدة المحسوبة (computed columns) في قواعد البيانات التي تدعم هذا النوع من الأعمدة، أو في حالة الاختيار من معرض (view)، فيشمل ذلك الأعمدة من جدول / معرض معيّن، والتي كان من الممكن أن يُحسّنها "مُحسِّن الاستعلام" (query optimiser)

4. احتمال حدوث أخطاء غير متوقعة عند إضافة أعمدة إلى الجداول والمعارض لاحقًا، ممَّا قد يؤدي إلى أسماء أعمدة غير واضحة. على سبيل المثال:

```
SELECT * FROM orders JOIN people ON people.id = orders.personid
ORDER BY displayname
```

في حال إضافة عمود يُسمى `displayname` إلى جدول الطلبات - الجدول orders - قصد السماح للمستخدمين بتقديم طلباتهم تحت أسماء من اختيارهم ليسهل عليهم الرجوع إليها مستقبلًا، فسيظهر اسم العمود مرتين في المخرجات، ونتيجة لذلك قد لا تكون عبارة `ORDER BY` واضحة، وهو ما قد يتسبب في خطأ ("ambiguous column name" في إصدارات MS SQL Server الحديثة). في المثال أعلاه، قد يعرض التطبيق اسم الطلب مكان اسم الشخص بالخطأ، نتيجة أنَّ العمود الجديد سيُعاد أولًا.

2. استخدام SELECT مع كُنى الأعمدة

تُستخدَم الأسماء المستعارة أو الكُنى (aliases) لاختصار أسماء الأعمدة أو جعلها ذات معنى. ويساعد ذلك على اختزال الشيفرة وتسهيل قراءتها جزَّاء تجنُّب أسماء الجداول الطويلة وتمييز الأعمدة (على سبيل المثال، قد يكون هناك مُعرِّفان في الجدول، بيْد أنّ واحدًا منهما فقط سيستخدم في العبارة)، بالإضافة إلى تسهيل استخدام أسماء وصفية أطول في قاعدة بياناتك مع إبقاء الاستعلامات الجارية عليها مختصرة.

علاوة على ذلك، قد تكون الكنى إجبارية في بعض الأحيان (على سبيل المثـال في المعـارض) من أجل تسمية المخرجات المحسوبة (computed outputs).

جميع إصدارات SQL

يمكن إنشاء الكنى في جميع إصدارات SQL باستخدام علامات الاقتباس المزدوجة (").

```
SELECT
    FName AS "First Name",
    MName AS "Middle Name",
    LName AS "Last Name"
FROM Employees
```

إصدارات خاصة من SQL

يمكنك استخدام علامات الاقتباس المفردة ('')، وعلامات الاقتباس المزدوجـة (") والأقـواس المربّعة ([]) لإنشاء كُنيَة في Microsoft SQL Server.

```
SELECT
    FName AS "First Name",
    MName AS 'Middle Name',
    LName AS [Last Name]
FROM Employees
```

سينتج عن الشيفرة أعلاه الخرج:

Last Name	Middle Name	First Name
Smith	John	James
Johnson	James	John
Williams	Marcus	Michael

ستعيد هذه العبارة عمودين FName و LName يحملان الاسم البديل المعطى (الكنية). وقد تمذلك باستخدام العامل AS متبوعًا بالكنية، أو بكتابة الكنية مباشرةً بعد اسم العمود.

ستكون للاستعلام التالي نفس النتيجة الواردة أعلاه:

```
SELECT
    FName "First Name",
    MName "Middle Name",
    LName "Last Name"
FROM Employees
```

Last Name	Middle Name	First Name
Smith	John	James
Johnson	James	John
Williams	Marcus	Michael

كما تلاحظ، فإنَّ النسخة الصريحة (أي استخدام العامل AS) أفضل، لأنَّها أوضح وأسهل للقراءة. إذا كانت الكنية مؤلفة من كلمة واحدة، ولم تكن كلمة محجوزة، فيمكن كتابتها بدون علامات الاقتباس المفردة أو المزدوجة أو الأقواس المربعة:

```
SELECT
    FName AS FirstName,
    LName AS LastName
FROM Employees
```

LastName	FirstName
Smith	James
Johnson	John

LastName	FirstName
Williams	Michael

هناك شكل إضافي متاح في MS SQL Server، وهو:

```
<alias> = <column-or-calculation>
```

إليك المثال التالي:

```
SELECT FullName = FirstName + ' ' + LastName,
       Addr1    = FullStreetAddress,
       Addr2    = TownName
FROM CustomerDetails
```

والذي يكافئ:

```
SELECT FirstName + ' ' + LastName As FullName
       FullStreetAddress          As Addr1,
       TownName                   As Addr2
FROM CustomerDetails
```

وسيؤدي كلاهما إلى النتيجة:

Addr2	Addr1	FullName
TownVille	AnyStreet 123	James Smith
Anytown	MyRoad 668	John Johnson
Williamsburgh	High End Dr 999	Michael Williams

يرى البعض أنَّ استخدام = بدلًا من As أفضل من ناحية المقروئية، فيما يوصي آخرون بتجنُّب استخدامها لأنَّها ليست قياسية، وبالتالي لا تدعمها جميع قواعد البيانات، كما قد يتداخل استخدامها مع الاستخدامات الأخرى للمحرف =.

جميع إصدارات SQL

إن كنت بحاجة إلى استخدام الكلمات المحجوزة، فيمكنك استخدام الأقواس المربعة أو علامات الاقتباس لتهريب الكنية (escape):

```
SELECT
    FName as "SELECT",
    MName as "FROM",
    LName as "WHERE"
FROM Employees
```

إصدارات خاصة من SQL

بالمثل، يمكنك تهريب الكلمات المفتاحية في MSSQL عبر عدَّة مقاربات:

```
SELECT
    FName AS "SELECT",
    MName AS 'FROM',
    LName AS [WHERE]
FROM Employees
```

WHERE	FROM	SELECT
Smith	John	James
Johnson	James	John
Williams	Marcus	Michael

يمكن أيضًا استخدام كنى الأعمدة في أيّ من العبارات النهائية في نفس الاستعلام، من قبيل العبارة ORDER BY:

```
SELECT
    FName AS FirstName,
    LName AS LastName
```

```
FROM
    Employees
ORDER BY
    LastName DESC
```

بالمقابل، لا يجوز استخدام الشيفرة التالية لإنشاء كنية تساوي الكلمات المحجوزة (SELECT و FROM)، لأنَّ ذلك سيتسبَّب في العديد من الأخطاء في مرحلة التنفيذ.

```
SELECT
    FName AS SELECT,
    LName AS FROM
FROM
    Employees
ORDER BY
    LastName DESC
```

ا. اختيار أعمدة فردية

انظر إلى الشيفرة التالية:

```
SELECT
    PhoneNumber,
    Email,
    PreferredContact
FROM Customers
```

ستعيد الأعمدة PhoneNumber و Email و PreferredContact من جميع صفوف الجدول Customers. كما ستُعَاد الأعمدة بالتسلسل الذي تظهر به في عبارة SELECT. وها هي النتيجة:

PreferredContact	Email	PhoneNumber
PHONE	william.jones@example.com	3347927472
EMAIL	dmiller@example.net	2137921892

PreferredContact	Email	PhoneNumber
EMAIL	richard0123@example.com	NULL

إذا رُبطَت عدَّة جداول معًا، فيمكنك اختيار الأعمدة من جداول معيَّنة عن طريق وضع اسم الجدول قبل اسم العمود على النحو `[table_name].[column_name]` مثل:

```
SELECT
    Customers.PhoneNumber,
    Customers.Email,
    Customers.PreferredContact,
    Orders.Id AS OrderId
FROM
    Customers
LEFT JOIN
    Orders ON Orders.CustomerId = Customers.Id
```

تعني العبارة `Orders.Id AS OrderId` أنَّ الحقل `Id` من الجدول `Orders` سيُعاد كعمود يُسمى `OrderId`. راجع قسم الاختيار عبر الكنى أسفله لمزيد من المعلومات.

لتجنب استخدام أسماء الجداول الطويلة، يمكنك تكنية الجدول، فهذا سيخفف من صعوبات كتابة أسماء الجداول الطويلة مع كل حقل تختاره في عمليات الدمج (joins). وعند استخدام الدمج الذاتي (self join وهو دمج نسختين من الجدول نفسه)، فعليك تكنِية الجداول -التي تمثِّل الجدول نفسه- لتمييزها عن بعضها.

يمكن كتابة كنية الجدول على النحو التالي: `Customers c` أو `Customers AS c`، إذ تتصرَّف `c` هنا ككُنية لاسم الجدول `Customers`، ويمكننا آنذاك أن نختار مثلا الحقل `Email` عبر الصيغة: `c.Email`:

```
SELECT
    c.PhoneNumber,
    c.Email,
```

```
    c.PreferredContact,
    o.Id AS OrderId
FROM
    Customers c
LEFT JOIN
    Orders o ON o.CustomerId = c.Id
```

3. اختيار عدد معيَّن من السجلات

عرَّف معيار SQL 2008 العبارة `FETCH FIRST` كطريقة لاختيار عدد السجلات المُعادة.

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
FETCH FIRST 10 ROWS ONLY
```

لم يُـدعَم هـذا المعيـار إلا في الإصـدارات الأخـيرة من بعض أنظمـة إدارة قواعـد البيانـات (RDMSs). فيما توفِّر الأنظمة الأخرى صياغة غير قياسية لأداء الغرض ذاته. يـدعم الإصـدار 11.x من Progress OpenEdge أيضًا الصياغة `FETCH FIRST <n> ROWS ONLY`.

بالإضـافة إلى ذلـك، تسـمح إضـافة العبـارة `OFFSET <m> ROWS` إن اسـتعملت قبل `FETCH FIRST <n> ROWS ONLY` بتخطي عدد مُحدَّد من الصفوف قبل جلب الصفوف.

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
OFFSET 5 ROWS
FETCH FIRST 10 ROWS ONLY
```

الاستعلام التالي مدعوم في SQL Server و MS Access:

```
SELECT TOP 10 Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
```

لفعل الشيء نفسه في MySQL أو PostgreSQL، يجب استخدام الكلمة المفتاحية LIMIT:

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
ORDER BY UnitPrice DESC
LIMIT 10
```

وفي Oracle، ينبغي استخدام ROWNUM:

```
SELECT Id, ProductName, UnitPrice, Package
FROM Product
WHERE ROWNUM <= 10
ORDER BY UnitPrice DESC
```

وينتج عن هذا 10 سجلات.

```
Id    ProductName                    UnitPrice
Package
38    Côte de Blaye                  263.50                 12 -
75 cl bottles
29    Thüringer Rostbratwurst        123.79                 50 bags
x 30 sausgs.
9     Mishi Kobe Niku                 97.00                 18 -
500 g pkgs.
20    Sir Rodney's Marmalade         81.00                  30 gift
boxes
18    Carnarvon Tigers                    62.50
16 kg pkg.
59    Raclette Courdavault             55.00                  5
kg pkg.
51    Manjimup Dried Apples          53.00                  50 - 300
g pkgs.
62    Tarte au sucre                   49.30                  48
pies
43    Ipoh Coffee                      46.00                  16 -
500 g tins
28    Rössle Sauerkraut               45.60                  25 -
```

```
825 g cans
```

تنبيه حول اختلاف الأنظمة: يجدر بالذكر أنَّ الكلمة المفتاحية TOP في نظام Microsoft SQL تعمل بعد عبارة WHERE، وتعيد عددًا محددًا من النتائج في حال كانت تلك النتائج متوافرة في أيِّ مكان من الجدول، بينما تعمل ROWNUM كجزء من عبارة WHERE، لذا إذا لم تتحقَّق الشروط الأخرى في العدد المُحدَّد من الصفوف في بداية الجدول، فلن تحصل على أيِّ نتيجة، حتى لو كان من الممكن العثور على نتائج أخرى.

4. الاختيار الشرطي

يمكن استخدام العبارتين SELECT و WHERE معًا على النحو التالي:

```
SELECT column1, column2, columnN
FROM table_name
WHERE [condition]
```

يمكن أن يكون الشرط [condition] أيِّ تعبير صالح في SQL يستخدم المعاملات المنطقية التالية:

```
>
<
=
>=
<=
LIKE
NOT
IN
BETWEEN
...
```

تُعيد العبارة التالية جميع الأعمدة من الجدول "Cars" ذات الحالة "READY":

```
SELECT * FROM Cars WHERE status = 'READY'
```

5. الاختيار باستخدام CASE

تُستخدَم عبارة CASE لتطبيق عملية معيَّنة على النتائج مباشرة:

```
SELECT CASE WHEN Col1 < 50 THEN 'under' ELSE 'over' END
threshold
FROM TableName
```

يمكن أيضًا سلسَلَة CASE على النحو التالي:

```
SELECT
   CASE WHEN Col1 < 50 THEN 'under'
 WHEN Col1 > 50 AND Col1 <100 THEN 'between'
 ELSE 'over'
   END threshold
FROM TableName
```

يمكن أيضا استخدام عبارة CASE داخل أخرى:

```
SELECT
    CASE WHEN Col1 < 50 THEN 'under'
                ELSE
 CASE WHEN Col1 > 50 AND Col1 <100 THEN Col1
 ELSE 'over' END
     END threshold
FROM TableName
```

6. اختيار أعمدة ذات أسماء مطابقة لكلمات محجوزة

عندما يتطابق اسم العمود مع كلمة مفتاحية محجوزة، ينص معيار SQL على ضرورة أن تُحاط بعلامات اقتباس مزدوجة:

```
SELECT
    "ORDER",
    ID
FROM ORDERS
```

لاحظ أنَّ هذا يجعل اسم العمود حساسًا لحالة الأحرف. بعض نظم إدارة قواعد البيانات لديها طرائقها الخاصة لاقتباس الأسماء. على سبيل المثال، يستخدم SQL Server أقواس مربعة:

```
SELECT
 [Order],
    ID
FROM ORDERS
```

بينما تستخدم MySQL (و MariaDB) افتراضيا علامة اقتباس مائلة (backtick):

```
SELECT
    `Order`,
    id
FROM orders
```

7. الاختيار باستخدام كنى الجداول

إليك المثال التالي:

```
SELECT e.Fname, e.LName
FROM Employees e
```

يساعد إعطاء جدول الموظفين Employees الكنية "e" مباشرةً بعد اسمه في إزالة الغموض في حال احتوت العديد من الجداول حقولًا تحمل نفس الاسم، وكنت تحتاج إلى تحديد الجدول الذي تريد استخلاص البيانات منه.

```
SELECT e.Fname, e.LName, m.Fname AS ManagerFirstName
FROM Employees e
 JOIN Managers m ON e.ManagerId = m.Id
```

لاحظ أنَّه بمجرد تعريف الكنية، فلن يكون بمقدورك استخدام اسم الجدول الأساسي بعد الآن، لهذا ستطرح الشيفرة التالية خطأً:

```
SELECT e.Fname, Employees.LName, m.Fname AS ManagerFirstName
FROM Employees e
JOIN Managers m ON e.ManagerId = m.Id
```

تجدر الإشارة إلى أنَّ كنى الجداول - أو "متغيرات النطاق" (range variables) إن أردنا التقيد بالتسميات الرسمية - قُدِّمت في لغة SQL لحل مشكلة الأعمدة المُكرَّرة التي تنجم عن استخدام INNER JOIN. وقد صَحَّح المعيار SQL 1992 هذه الثغرة من خلال إدخال NATURAL JOIN (مطبّق حاليًا في mySQL و PostgreSQL و Oracle ولكن ليس في SQL Server بعدُ)، وقد دمجن ذلك ألا تحدث مشكلة الأسماء المُكرَّرة للأعمدة.

في المثال أعلاه، تُدمجُ الجداول في الأعمدة ذات الأسماء المختلفة (Id و ManagerId) ولكن ليس في الأعمدة التي تحمل الاسم نفسه (FName، LName)، هذا الأمر يتطلَّب إعادة تسمية الأعمدة قبل عملية الدمج:

```
SELECT Fname, LName, ManagerFirstName
FROM Employees
NATURAL JOIN
( SELECT Id AS ManagerId, Fname AS ManagerFirstName
FROM Managers ) m;
```

رغم أنَّه يجب الإعلان عن متغير الكنية / النطاق الخاص بالجدول المُعدَّل (أو ستطرح SQL خطأً)، إلا أنَّه من غير المنطقي أبدًا استخدامه فعليًا في الاستعلامات.

ا. الاختيار بناءً على عدة شروط

تُستخدم الكلمة المفتاحية AND لإضافة المزيد من الشروط إلى الاستعلام. ليكن لدينا الجدول التالي:

Gender	Age	Name
M	18	Sam
M	21	John
M	22	Bob
F	23	Mary

فبتطبيق المثال التالي عليه:

```
SELECT name FROM persons WHERE gender = 'M' AND age > 20;
```

سنحصل على النتيجة التالية:

Name
John
Bob

يمكن أيضًا استخدام المعامل المنطقي OR على النحو التالي:

```
SELECT name FROM persons WHERE gender = 'M' OR age < 20;
```

سينتج عن هذا:

Name
Sam
John
Bob

يمكن دمج هذه الكلمات المفتاحية لبناء شروط أكثر تعقيدًا:

```
SELECT name
FROM persons
WHERE (gender = 'M' AND age < 20)
    OR (gender = 'F' AND age > 20);
```

سينتج عن هذا:

Name
Sam
John
Bob

8. الاختيار دون حجز الجدول

في بعض الأحيان، عندما تُستخدَم الجداول لأجل القراءة أساسًا (أو حصرًا)، فإنَّ الفهرسة (indexing) لا تكون ضرورية، لذا لن تكون هناك حاجة إلى قفل أو حجز (LOCK) الجدول أثناء الاختيار لمنع أي عملية أخرى من تعديله أثناء القراءة.

هذه بعض الأمثلة على ذلك:

- SQL Server

```
SELECT * FROM TableName WITH (nolock)
```

- MySQL

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM TableName;
SET SESSION TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

- Oracle

```
SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM TableName;
```

- DB2

```
SELECT * FROM TableName WITH UR;
```

العبارة UR كناية عن «قراءة غير ملتزم بها» (uncommitted read). في حال استخدامها على جدول أثناء تعديل أحد حقوله، فقد تحدث نتائج غير مُتوقَّعَة.

9. الاختيار باستخدام الدوال التجميعية

ا. حساب المتوسط

تعيد الدالة التجميعية AVG() متوسط القيم المختارة:

```
SELECT AVG(Salary) FROM Employees
```

يمكن أيضًا استخدام الدوال مع عبارة where على النحو التالي:

```
SELECT AVG(Salary) FROM Employees where DepartmentId = 1
```

ويمكن أيضًا استخدام الدوال مع العبارة GROUP BY. مثلًا، إذا تم تصنيف الموظف في عدَّة أقسام، وأردنا أن نُحدِّد متوسط الراتب في كل قسم، فيمكننا استخدام الاستعلام التالي.

```
SELECT AVG(Salary) FROM Employees GROUP BY DepartmentId
```

ب. القيمة الصغرى

تعيد الدالة التجميعية MIN() الحد الأدنى من القيم المُحدَّدة.

```
SELECT MIN(Salary) FROM Employees
```

ج. القيمة الكبرى

تعيد الدالة التجميعية MAX() الحد الأقصى للقيم المُحدَّدة.

```
SELECT MAX(Salary) FROM Employees
```

د. الإحصاء

تعيد الدالة التجميعية COUNT() عدد القيم المُحدَّدة.

```
SELECT Count(*) FROM Employees
```

يمكن أيضًا دمجها مع العبارة where للحصول على عدد الصفوف التي تفي بشروط مُحدَّدة.

```
SELECT Count(*) FROM Employees where ManagerId IS NOT NULL
```

يمكن أيضًا اختيار عمود معيَّن للحصول على عدد القيم في ذلك العمود. لاحظ أنَّ القيم المعدومة NULL لا تُحتسَب.

```
Select Count(ManagerId) from Employees
```

يمكن أيضًا دمج Count مع الكلمة المفتاحية DISTINCT.

```
Select Count(DISTINCT DepartmentId) from Employees
```

ه. الجمع التراكمي

تعيد الدالة التجميعية SUM() مجموع القيم المُختارة في جميع الصفوف.

```
SELECT SUM(Salary) FROM Employees
```

10. الاختيار من بين قيم مُعيَّنة من عمود

المثال التالي:

```
SELECT * FROM Cars  WHERE status IN ( 'Waiting', 'Working' )
```

يكافئ الشيفرة التالية:

```
SELECT * FROM Cars  WHERE ( status = 'Waiting' OR status =
'Working' )
```

إذ أنَّ `value IN ( <value list> )` هي مُجرَّد اختصار لمعامل `OR` المنطقي.

11. تطبيق الدوال التجميعية على مجموعات من الصفوف

يحسب المثال التالي عدد الصفوف بناءً على قيمة مُحدَّدة من العمود:

```
SELECT category, COUNT(*) AS item_count
FROM item
GROUP BY category;
```

ويحسب المثال التالي متوسط الدخل حسب القسم:

```
SELECT department, AVG(income)
FROM employees
GROUP BY department;
```

الشيء الأساسي هنا هو اختيار الأعمدة المُحدَّدة في عبارة `GROUP BY` حصرًا، أو استخدامها مع الدوال التجميعية (Aggregate function). يمكن أيضًا استخدام العبارة `WHERE` مع `GROUP BY`، إلّا أنَّ `WHERE` ستُرشِّح السجلات قبل تجميعها (grouping):

```
SELECT department, AVG(income)
FROM employees
WHERE department <> 'ACCOUNTING'
GROUP BY department;
```

إن أردت ترشيح النتائج بعد الانتهاء من التجميع، مثلًا إن أردت ألّا تعرض إلا الأقسام التي يتجاوز معدل دخلها 1000، سيكون عليك استخدام العبارة `HAVING`.

```
SELECT department, AVG(income)
FROM employees
WHERE department <> 'ACCOUNTING'
GROUP BY department
HAVING avg(income) > 1000;
```

12. الاختيار مع ترتيب النتائج

إليك المثال التالي:

```
SELECT * FROM Employees ORDER BY LName
```

ستعيد هذه العبارة جميع الأعمدة من الجدول Employees.

PhoneNumber	LName	FName	Id
2468101214	Johnson	John	2
1234567890	Smith	James	1
1357911131	Williams	Michael	3

يمكن ترتيب النتائج تصاعديًا أو تنازليًا:

```
SELECT * FROM Employees ORDER BY LName DESC
SELECT * FROM Employees ORDER BY LName ASC
```

تُغيِّر عبارة ASC اتجَاه الترتيب، فيمكن أيضا إجراء الترتيب وفق عدَّة أعمدة على النحو التالي:

```
SELECT * FROM Employees ORDER BY LName ASC, FName ASC
```

سيُرتَّب هذا المثال النتائج حسب الحقل LName أولًا، أمَّا بالنسبة للسجلات التي لها نفس قيمة الحقل LName، فسُيرتِّبها حسب FName. سينتج عن هذا قوائم مشابهة للقوائم التي تجدها في دفتر الهاتف. إن أردت تجنُّب إعادة كتابة اسم العمود في عبارة ORDER BY، يمكنك استخدام رقم العمود بدلاً من اسمه. انتبه إلى أنَّ أرقام الأعمدة تبدأ من 1.

```
SELECT Id, FName, LName, PhoneNumber FROM Employees ORDER BY 3
```

يمكن أيضًا تدمجين عبارة CASE في عبارة ORDER BY.

```
SELECT Id, FName, LName, PhoneNumber FROM Employees ORDER BY
CASE WHEN LName='Jones' THEN 0 ELSE 1
END ASC
```

ستُرتَّب هذه الشيفرة النتائج بحيث تكون السجلات التي يساوي حقل LName خاصتها القيمة "Jones" في الأعلى.

13. استخدام null لأجل الاختيار

إليك المثال التالي:

```
SELECT Name FROM Customers WHERE PhoneNumber IS NULL
```

تختلف صياغة الاختيار باستخدام القيم المعدومة null عن الصياغة العادية، إذ أنَّها لا تستخدم معامل التساوي = وإنَّما تستخدم IS NULL أو IS NOT NULL.

14. اختيار قيم فريدة

```
SELECT DISTINCT ContinentCode
FROM Countries;
```

سيعيد هذا الاستعلام جميع القيم الفريدة والمختلفة عن بعضها من العمود ContinentCode في الجدول Countries (هذه تجربة حية للمثال):

ContinentCode
OC
EU
AS
NA
AF

15. اختيار الصفوف من عدة جداول

إليك الشيفرة التالية:

```
SELECT *
FROM
    table1,
    table2

SELECT
    table1.column1,
    table1.column2,
    table2.column1
FROM
    table1,
    table2
```

تُسمَّى هذه العملية الجداء المتقاطع (cross product) في SQL، وهي مكافئة للجداء المتقاطع في المجموعات. تعيد هذه العبارات أعمدة مُختارة من عدَّة جداول في استعلام واحد، ولا توجد علاقة مُحدَّدة بين الأعمدة المُعادة من كل جدول.

3 التجميع والترتيب

سنتحدث في هذا الفصل عن كيفيَّة استخدام العبارتين GROUP BY و ORDER BY لأجل تجميع نتائج الاستعلامات في SQL وترتيبها.

1. التجميع عبر GROUP BY

يمكن تجميع نتائج استعلام SELECT حسب عمود واحد أو أكثر باستخدام عبارة GROUP BY والتي تُجمِّع كل النتائج التي لها نفس القيمة في الأعمدة المُجمَّعة (grouped columns)، وينتج عنها جدول بالنتائج الجزئية، بدلًا من إرجاع نتيجة واحدة.

يمكن استخدام GROUP BY مع دوال التجميع (aggregation functions) لتحديد كيفيَّة تجميع الأعمدة باستخدام العبارة HAVING.

ا. مثال على استخدام GROUP BY

تشبه GROUP BY عبارة for each المُستخدمة في الكثير من لغات البرمجة، وإليك مثلًا الاستعلام التالي:

```
SELECT EmpID, SUM (MonthlySalary)
FROM Employee
GROUP BY EmpID
```

في الشيفرة أعلاه، نريد الحصول على مجموع الحقل MonthlySalary لكل قيم EmpID، مثلًا، في الجدول التالي:

```
+-----+----------------------+
|EmpID|MonthlySalary|
+-----+----------------------+
|1      |200                |
+-----+----------------------+
|2      |300                |
+-----+----------------------+
```

سنحصل على النتيجة التالية:

```
+-+---+
|1|200|
+-+---+
|2|300|
+-+---+
```

لا يبدو أنَّ `Sum` تفعل أيَّ شيء، وذلك لأنَّ مجموع عدد ما يساوي العدد نفسه.

إليك الآن الجدول التالي:

```
+-----+--------------------+
|EmpID|MonthlySalary|
+-----+--------------------+
|1    |200                 |
+-----+--------------------+
|1    |300                |
+-----+--------------------+
|2    |300               |
+-----+--------------------+
```

سنحصل بتطبيق الاستعلام نفسه على النتيجة التالية:

```
+-+---+
|1|500|
+-+---+
|2|300|
+-+---+
```

ب. ترشيح نتائج GROUP BY باستخدام عبارة HAVING

ترشِّح عبارة `HAVING` نتائج `GROUP BY`. سنستخدم في الأمثلة التالية قاعدة البيانات `Library` المُعرَّفة في الفصل الأول.

إليك مثال حول إعادة جميع المؤلفين الذين كتبوا أكثر من كتاب (مثال حي):

```
SELECT
  a.Id,
  a.Name,
 COUNT(*) BooksWritten
FROM BooksAuthors ba
 INNER JOIN Authors a ON a.id = ba.authorid
GROUP BY
  a.Id,
  a.Name
HAVING COUNT(*) > 1     --  HAVING BooksWritten > 1  يكا فئ
;
```

مثالٌ آخر عن إعادة جميع الكتب التي أُلِّفت من قبل ثلاثة مؤلفين أو أكثر (مثال حي):

```
SELECT
  b.Id,
  b.Title,
 COUNT(*) NumberOfAuthors
FROM BooksAuthors ba
 INNER JOIN Books b ON b.id = ba.bookid
GROUP BY
  b.Id,
  b.Title
HAVING COUNT(*) > 3     -- HAVING NumberOfAuthors > 3
;
```

ج. حساب عدد الصفوف لكل مدخل فريد في عمود

دعنا نفترض أنَّنا نريد عدَّ أو حساب مجاميع فرعية لقيمة معينة في عمود.

إليك الجدول التالي:

GreatHouseAllegience	Name
Stark	Arya

GreatHouseAllegience	Name
Lannister	Cercei
Lannister	Myrcella
Greyjoy	Yara
Stark	Catelyn
Stark	Sansa

في حال عدم استخدام العبارة GROUP BY، ستعيد COUNT إجمالي عدد الصفوف:

```
SELECT Count(*) Number_of_Westerosians
FROM Westerosians
```

سنحصل على الخرج الناتج:

Number_of_Westerosians
6

في حال استخدام GROUP BY، يمكن عدُّ المستخدمين لكل قيمة في عمود معيَّن كما يوضِّح المثال التالي:

```
SELECT GreatHouseAllegience House, Count(*)
Number_of_Westerosians
FROM Westerosians
GROUP BY GreatHouseAllegience
```

الخرج الناتج:

Number_of_Westerosians	House
3	Stark

Number_of_Westerosians	House
1	Greyjoy
2	Lannister

من الشائع الجمع بين العبارتين `GROUP BY` و `ORDER BY` لترتيب النتائج تصاعديًا أو تنازليًا:

```
SELECT GreatHouseAllegience House, Count(*)
Number_of_Westerosians
FROM Westerosians
GROUP BY GreatHouseAllegience
ORDER BY Number_of_Westerosians Desc
```

الخرج الناتج:

Number_of_Westerosians	House
3	Stark
2	Lannister
1	Greyjoy

د. تجميع ROLAP (استخراج البيانات)

يوفِّر معيار SQL معاملين تجميعيّين (aggregate operators) إضافيين هما:

- `with data cube`: يوفِّر كل التجميعات الممكنة لسمات وسيط العبارة (argument of the clause attributes).
- `with roll up`: يوفِّر المجموع الناتج عن اعتبار السمات مُرتَّبة من اليسار إلى اليمين مع موازنتها بكيفية إدراجها في وسيط العبارة (argument of the clause).

تُستخدَم القيمة ALL للكناية عن جميع القيم التي يمكن أن تأخـذها سـمة (attribute). تـدعم إصدارات SQL القياسية التالية هذه الميزات: 1999 - 2003 - 2006 - 2008 - 2011.

إليك الجدول التالي:

Total_amount	Brand	Food
100	Brand1	Pasta
250	Brand2	Pasta
300	Brand2	Pizza

لاستخدام عبارة cube:

```
select Food,Brand,Total_amount
from Table
group by Food,Brand,Total_amount with cube
```

الخرج الناتج:

Total_amount	Brand	Food
100	Brand1	Pasta
250	Brand2	Pasta
350	ALL	Pasta
300	Brand2	Pizza
300	ALL	Pizza
100	Brand1	ALL
550	Brand2	ALL

Total_amount	Brand	Food
650	ALL	ALL

استخدام roll up:

```
select Food,Brand,Total_amount
from Table
group by Food,Brand,Total_amount with roll up
```

الخرج الناتج:

Total_amount	Brand	Food
100	Brand1	Pasta
250	Brand2	Pasta
300	Brand2	Pizza
350	ALL	Pasta
300	ALL	Pizza
650	ALL	ALL

2. الترتيب عبر ORDER BY

ا. الترتيب حسب رقم العمود (بدلًا من اسمه)

يمكنك استخدام رقم العمود (يبدأ العمود الموجود في أقصى اليسار من الرقم "1") للإشارة إلى العمود الذي يستند الترتيب إليه. إيجابيات هذا الخيار هي أنَّه مناسب في حال كانت هناك إمكانية لتغيير أسماء الأعمدة لاحقًا، إذ سيُجنِّبُك ذلك تخريب الشيفرة. ولكن سلبياته هي أنَّ

استخدام رقم العمود بدل اسمه سيُضعِف مقروئية الاستعلام (وازن مثلًا بين `ORDER BY Reputation` و `ORDER BY 14`).

يرتب الاستعلام التالي النتيجة حسب المعلومات الموجودة في العمود رقم 3 بدلًا من الاعتماد اسم العمود `Reputation`.

```
SELECT DisplayName, JoinDate, Reputation FROM Users ORDER BY 3
```

الخرج الناتج:

Reputation	JoinDate	DisplayName
1	2008-09-15	Community
11739	2008-10-03	Jarrod Dixon
12567	2008-10-03	Geoff Dalgas
25784	2008-09-16	Joel Spolsky
37628	2008-09-16	Jeff Atwood

ب. إعادة أعلى س صفًا بناءً على قيمة العمود

في المثال التالي، يمكنك استخدام `GROUP BY` و `TOP` لتحديد الصفوف المُعادة وترتيبها. لنفترض أنَّك تريد الحصول على أفضل 5 مستخدمين من حيث السمعة في موقع مُتخصِّص في الأسئلة والأجوبة.

سنكتب استعلامًا بدون ORDER BY يعيد أعلى 5 صفوف مرتبة حسب الإعداد الافتراضي، والذي هو `Id` في هذه الحالة، أي العمود الأول في الجدول (رغم أنَّه لن يظهر في النتائج):

```
SELECT TOP 5 DisplayName, Reputation FROM Users
```

الخرج الناتج:

Reputation	DisplayName
1	Community
12567	Geoff Dalgas
11739	Jarrod Dixon
37628	Jeff Atwood
25784	Joel Spolsky

سنجرب استخدام `ORDER BY` في نفس المثال:

```
SELECT TOP 5 DisplayName, Reputation
FROM Users
ORDER BY Reputation desc
```

الخرج الناتج:

Reputation	DisplayName
865023	JonSkeet
661741	Darin Dimitrov
650237	BalusC
625870	Hans Passant
601636	Marc Gravell

انتبه إلى أنَّ بعض إصدارات SQL (مثل MySQL) تستخدم العبارة LIMIT في نهاية SELECT، بدلًا من استخدام TOP في البداية كما هو موضَّح في المثال التالي:

```
SELECT DisplayName, Reputation
FROM Users
ORDER BY Reputation DESC
LIMIT 5
```

ج. الترتيب المُخصَّص

يمكنك استخدام التعليمة ORDER BY Department لترتيب الجدول Employee حسب القسم department. أمَّا إن أردت ترتيبه ترتيبًا غير أبجدي، فيجب عليك تحويل قيم Department إلى قيم أخرى قابلة للترتيب؛ يمكن فعل ذلك باستخدام عبارة CASE:

Department	Name
IT	Hasan
HR	Yusuf
HR	Hillary
IT	Joe
HR	Merry
Accountant	Ken

في المثال التالي:

```
SELECT *
FROM Employee
ORDER BY CASE Department
 WHEN 'HR'            THEN 1
 WHEN 'Accountant'  THEN 2
 ELSE                          3
```

```
END;
```

سنحصل على الخرج:

Department	Name
HR	Yusuf
HR	Hillary
HR	Merry
Accountant	Ken
IT	Hasan
IT	Joe

د. الترتيب بالكنى

بسبب طريقة معالجة الاستعلامات المنطقية، يمكن ترتيب نتائج الاستعلامات حسب الكُنى (Order by Alias).

```
SELECT DisplayName, JoinDate as jd, Reputation as rep
FROM Users
ORDER BY jd, rep
```

كما يمكن استخدام الترتيب النسبي (relative order) للأعمدة -أي الترتيب حسب رقم العمود- في عبارة الاختيار `select`. سنعود إلى المثال أعلاه، ولكن بدلًا من استخدام الكُنيَة، سنستخدم الترتيب النسبي.

```
SELECT DisplayName, JoinDate as jd, Reputation as rep
FROM Users
ORDER BY 2, 3
```

ه. الترتيب حسب عدة أعمدة

في المثال التالي:

```
SELECT DisplayName, JoinDate, Reputation FROM Users ORDER BY
JoinDate, Reputation
```

سيكون الخرج:

Reputation	JoinDate	DisplayName
1	2008-09-15	Community
25784	2008-09-16	Jeff Atwood
37628	2008-09-16	Joel Spolsky
11739	2008-10-03	Jarrod Dixon
12567	2008-10-03	Geoff Dalgas

3. الفرق بين Group By و Distinct

تُستخدَم العبارة `GROUP BY` في SQL مع دوال التجميع (aggregation functions).

بفرض لدينا الجدول التالي:

orderDate	orderValue	storeName	userId	orderId
20-03-2016	25	Store A	43	1
22-03-2016	50	Store B	57	2
25-03-2016	30	Store A	43	3
26-03-2016	10	Store C	82	4
29-03-2016	45	Store A	21	5

يستخدم الاستعلام أدناه GROUP BY لإجراء عمليات حسابية تجميعية (aggregated calculations):

```
SELECT
    storeName,
    COUNT(*) AS total_nr_orders,
    COUNT(DISTINCT userId) AS nr_unique_customers,
    AVG(orderValue) AS average_order_value,
    MIN(orderDate) AS first_order,
    MAX(orderDate) AS lastOrder
FROM
    orders
GROUP BY
    storeName;
```

يُعَاد الخرج التالي:

	lastOrder	average_order_value first_order	nr_unique_customers	total_nr_orders	storeName
29-03-2016	20-03-2016	33.3	2	3	Store A
22-03-2016	22-03-2016	50	1	1	Store B
26-03-2016	26-03-2016	10	1	1	Store C

بالمقابل، تُستخدَم DISTINCT لسرد تجميعة فريدة (unique combination) من القيم المختلفة للأعمدة المُحدَّدة.

```
SELECT DISTINCT
    storeName,
    userId
FROM
    orders;
```

سنحصل على الخرج:

userId	storeName
43	Store A
57	Store B
82	Store C
21	Store A

4. المعاملان المنطقيان AND و OR

يستخدم المعاملان AND و OR المنطقيان في بناء الشروط المنطقية.

إليك الجدول التالي:

City	Age	Name
Paris	10	Bob
Berlin	20	Mat
Prague	24	Mary

إن طبقنا المثال التالي:

```
select Name from table where Age>10 AND City='Prague'
```

سنحصل على الخرج:

Name
Mary

وإن طبقنا المثال التالي:

```
select Name from table where Age=10 OR City='Prague'
```

سنحصل على الخرج:

Name
Bob
Mary

4

تنفيذ تعليمات شرطية عبر CASE

يستعرض هذا الفصل العبارة CASE، والتي تُستخدَم لكتابة الشيفرات الشرطية (if-then).

1. حساب عدد الصفوف في عمود يحقق شرطًا

يمكن استخدام CASE مع SUM لحساب عدد العناصر المطابقة لشرط مُحدَّد (تشبه العبارة COUNTIF في Excel). الحيلة التي سنعتمدها هي أنّنا سنعيد قيمة ثنائية (binary) للدلالة على مطابقة الشرط، حيث يشير 1 إلى أنَّ الدخل يطابق الشرط، فيما يشير 0 إلى عدم المطابقة، بعد ذلك سنجمع الوحدات التي حصلنا عليها للحصول على عدد المطابقات.

في الجدول ItemSales التالي، سنحاول عدَّ العناصر الثمينة (EXPENSIVE):

PriceRating	Price	ItemId	Id
EXPENSIVE	34.5	100	1
CHEAP	2.3	145	2
EXPENSIVE	34.5	100	3
EXPENSIVE	34.5	100	4
AFFORDABLE	10	145	5

سنستخدم الاستعلام التالي:

```
SELECT
 COUNT(Id) AS ItemsCount,
 SUM ( CASE
 WHEN PriceRating = 'Expensive' THEN 1
 ELSE 0
         END
       ) AS ExpensiveItemsCount
FROM ItemSales
```

وسنحصل على الخرج التالي:

ExpensiveItemsCount	ItemsCount
3	5

هذا استعلام آخر بديل:

```
SELECT
 COUNT(Id) as ItemsCount,
 SUM (
 CASE PriceRating
 WHEN 'Expensive' THEN 1
 ELSE 0
 END
 ) AS ExpensiveItemsCount
FROM ItemSales
```

2. البحث الشرطي

يمكن استخدام CASE مع العبارة SELECT لترشيح النتائج حسب شرط معيَّن، بحيث لا تُعاد إلا النتائج التي تعيد القيمة المنطقية TRUE (هذا يختلف عن استخدام case العادي، والذي يتحقق من التكافؤ مع الدخل وحسب).

```
SELECT Id, ItemId, Price,
     CASE WHEN Price < 10 THEN 'CHEAP'
 WHEN Price < 20 THEN 'AFFORDABLE'
 ELSE 'EXPENSIVE'
     END AS PriceRating
FROM ItemSales
```

سنحصل على الخرج التالي:

PriceRating	Price	ItemId	Id
EXPENSIVE	34.5	100	1
CHEAP	2.3	145	2
EXPENSIVE	34.5	100	3
EXPENSIVE	34.5	100	4
AFFORDABLE	10	145	5

ا. صياغة CASE المختزلة

يقيِّم الشكل المختزل للعبارة CASE تعبيرًا ما (عادةً ما يكون عمودًا)، ويوازنه بعدة قيم. هذا الشكل مختصر قليلًا أكثر من الشكل العادي، ويُعفِيك من تكرار التعبير المقيَّم.

يمكن استخدام صياغة ELSE في الصياغة على النحو التالي:

```
SELECT Id, ItemId, Price,
 CASE Price WHEN 5  THEN 'CHEAP'
 WHEN 15 THEN 'AFFORDABLE'
 ELSE 'EXPENSIVE'
 END as PriceRating
FROM ItemSales
```

من المهم أن تدرك أنَّه عند استخدام الشكل المختصر، فسيُقيَّم التعبير بالكامل في كل عبارة WHEN. لذلك، فإنَّ الشيفرة التالية:

```
SELECT
 CASE ABS(CHECKSUM(NEWID())) % 4
 WHEN 0 THEN 'Dr'
 WHEN 1 THEN 'Master'
 WHEN 2 THEN 'Mr'
 WHEN 3 THEN 'Mrs'
```

```
 END
```

قد تعيد القيمة المعدومة NULL. لأنَّه في كل عبارة WHEN، تُستدعَى NEWID() مع نتيجة جديدة، وهذا يكافئ:

```
SELECT
 CASE
 WHEN ABS(CHECKSUM(NEWID())) % 4 = 0 THEN 'Dr'
 WHEN ABS(CHECKSUM(NEWID())) % 4 = 1 THEN 'Master'
 WHEN ABS(CHECKSUM(NEWID())) % 4 = 2 THEN 'Mr'
 WHEN ABS(CHECKSUM(NEWID())) % 4 = 3 THEN 'Mrs'
 END
```

لذلك يمكن أن تُفوَّت جميع عبارات WHEN، لنحصل على القيمة NULL.

ب. استخدام CASE في عبارة ORDER BY

سنستخدم في الشيفرة أدناه الأرقام 1، 2، 3 ...إلخ لتصنيف الطلب إلى أنواع:

```
SELECT * FROM DEPT
ORDER BY
CASE DEPARTMENT
 WHEN 'MARKETING' THEN  1
 WHEN 'SALES' THEN 2
 WHEN 'RESEARCH' THEN 3
 WHEN 'INNOVATION' THEN 4
 ELSE        5
 END,
          CITY
```

الخرج الناتج:

EMPLOYEES_NUMBER	DEPARTMENT	CITY	REGION	ID
9	MARKETING	Boston	New England	12

EMPLOYEES_NUMBER	DEPARTMENT	CITY	REGION	ID
12	MARKETING	San Francisco	West	15
8	SALES	Chicago	Midwest	9
12	SALES	New York	Mid-Atlantic	14
11	RESEARCH	Los Angeles	West	5
13	RESEARCH	Philadelphia	Mid-Atlantic	10
11	INNOVATION	Chicago	Midwest	4
9	HUMAN RESOURCES	Detroit	Midwest	2

ج. استخدام CASE في UPDATE

يزيد المثال التالي الأسعار في قاعدة البيانات:

```
UPDATE ItemPrice
SET Price = Price *
   CASE ItemId
 WHEN 1 THEN 1.05
 WHEN 2 THEN 1.10
 WHEN 3 THEN 1.15
 ELSE 1.00
    END
```

د. استخدام CASE مع القيم المعدومة NULL

في المثال التالي، يُمثِّل الرقم "0" القيم المعروفة، والموضوعة في البداية، فيما يمثل "1" القيم NULL، و هي موضوعة في آخر الترتيب:

```
SELECT ID
 ,REGION
 ,CITY
 ,DEPARTMENT
 ,EMPLOYEES_NUMBER
   FROM DEPT
   ORDER BY
   CASE WHEN REGION IS NULL THEN 1
   ELSE 0
   END,
   REGION
```

سنحصل على الخرج التالي:

EMPLOYEES_NUMBER	DEPARTMENT	CITY	REGION	ID
13	RESEARCH	Philadelphia	Mid-Atlantic	10
12	SALES	New York	Mid-Atlantic	14
8	SALES	Chicago	Midwest	9
9	MARKETING	Boston	New England	12
11	RESEARCH	Los Angeles	West	5
12	MARKETING	San Francisco	NULL	15
11	INNOVATION	Chicago	NULL	4
9	HUMAN RESOURCES	Detroit	NULL	2

ه. استخدام CASE في عبارة ORDER BY

لنفترض أنك بحاجة إلى ترتيب السجلات حسب القيمة الدنيا في عمودين. قد تستخدم بعض

قواعـد البيانـات الـدالتين غـير التجميعيـتين MIN() أو LEAST() (مثلا: ORDER BY ... MIN(Date1, Date2))، ولكن في SQL القياسية، يجب استخدام التعبير CASE.

يبحث التعبير CASE في الاستعلام أدنـاه في العمـودين Date1 و Date2 ، ويبحث عن العمـود الذي له أدنى قيمة، ثم يرتِّب السجلات وفقًا لتلك القيمة.

إليك الجدول التالي:

Date2	Date1	Id
2017-01-31	2017-01-01	1
2017-01-03	2017-01-31	2
2017-01-02	2017-01-31	3
2017-01-31	2017-01-06	4
2017-01-05	2017-01-31	5
2017-01-31	2017-01-04	6

سنطبِّق عليه الاستعلام التالي:

```
SELECT Id, Date1, Date2
FROM YourTable
ORDER BY CASE
       WHEN COALESCE(Date1, '1753-01-01') < COALESCE(Date2,
'1753-01-01') THEN Date1
       ELSE Date2
   END
```

الخرج الناتج:

Date2	Date1	Id
2017-01-31	2017-01-01	1
2017-01-02	2017-01-31	3
2017-01-03	2017-01-31	2
2017-01-31	2017-01-04	6
2017-01-05	2017-01-31	5
2017-01-31	2017-01-06	4

كما ترى، الصف ذو المُعرِّف `Id = 1` جاء أولًا، وذلك لأنَّ العمود `Date1` يحوي أدنى سجل في الجدول، وهو `2017-01-01`، بينما جاء الصف ذو المُعرِّف `Id = 3` ثانيًا، لأنَّ العمود `Date2` يحوي القيمة `2017-01-02`، وهي ثاني أقل قيمة في الجدول، وهكذا دواليك.

لقد رتَّبنا السجلات من `2017-01-01` إلى `2017-01-06` تصاعديًا، بغض النظر عن العمود `Date1` أو `Date2` الذي جاءت منه تلك القيم.

5

البحث والتنقيب والترشيح

يتطـــرَّق هـــذا الفصـــل إلى بعض معـــاملات SQL المُتخصِّصـــة في البحث والتنقيب وترشيح النتائج.

1. المعامل LIKE

ا. مطابقة الأنماط المفتوحة (open-ended pattern)

يطابق حرف البدل % الموضوع في بداية أو نهاية السلسلة النصية (أو كليهما) 0 حرف أو أكثر قبل بداية أو بعد نهاية النمط المراد مطابقته. يسمح استخدام '%' في الوسط بوجود 0 حرف أو أكثر بين جزأي النمط المُراد مُطابقَته.

سنستخدم جدول الموظفين Employees التالي:

Hire_date	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
23-03-2005	400	1	1	2468101214	Johnson	John	1
11-01-2010	400	1	1	2479100211	Amudsen	Sophie	2
06-08-2015	600	1	2	2462544026	Smith	Ronny	3
23-03-2005	400	1	1	2454124602	Sanchez	Jon	4
01-01-2000	800	1	2	2468021911	Knag	Hilde	5

تطابق العبارة التالية جميع السجلات التي يحتوي حقل FName خاصتها على السلسلة النصية 'on':

```
SELECT * FROM Employees WHERE FName LIKE '%on%';
```

سنحصل على الخرج التالي:

Hire_date	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
06-08-2015	600	1	2	2462544026	Smith	Ronny	3
23-03-2005	400	1	1	2454124602	Sanchez	Jon	4

يطابق التعبير التالي جميع السجلات التي يبدأ الحقل PhoneNumber خاصتها بالسلسلة النصية "246" في جدول الموظفين.

```
SELECT * FROM Employees WHERE PhoneNumber LIKE '246%';
```

سنحصل على الخرج التالي:

Hire_date	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
23-03-2005	400	1	1	2468101214	Johnson	John	1
06-08-2015	600	1	2	2462544026	Smith	Ronny	3
01-01-2000	800	1	2	2468021911	Knag	Hilde	5

تطابق العبارة التالية جميع السجلات التي تنتهي قيمة الحقل PhoneNumber خاصتها بالسلسلة النصية "11" في جدول الموظفين.

```
SELECT * FROM Employees WHERE PhoneNumber LIKE '%11'
```

Hire_date	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
11-01-2010	400	1	1	2479100211	Amudsen	Sophie	2
01-01-2000	800	1	2	2468021911	Knag	Hilde	5

يطابق التعبير التالي جميع السجلات التي يساوي الحرف الثالث من حقل Fname خاصتها 'n' في جدول الموظفين.

```
SELECT * FROM Employees WHERE FName LIKE '__n%';
```

(استخدمنا شرطتين سفليتين قبل 'n' لتخطي أول حرفين)

Hire_date	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
06-08-2015	600	1	2	2462544026	Smith	Ronny	3
23-03-2005	400	1	1	2454124602	Sanchez	Jon	4

ب. مطابقة محرف واحد

يمكن استخدام أحرف البدل وعلامة النسبة المئوية (%) والشرطة السفلية (_) لتوسيع أنماط الاختيار في SQL، إذ يمكن استخدام المحرف _ (الشرطة السفلية) كحرف بدل يمثل حرفًا واحدًا فقط.

يبحث النمط التالي عن جميع الموظفين الذين يتألف الحقل Fname خاصتهم من 3 حروف، ويبدأ بالحرف "j" وينتهي بـ "n".

```
SELECT * FROM Employees WHERE FName LIKE 'j_n'
```

يمكن استخدام المحرف _ أكثر من مرة بحيث يتصرَّف كبطاقة بدل (wild card) لمطابقة أنماط مُعيَّنة. على سبيل المثال، يُطابِق النمط أعلاه السلاسل النصية التالية: jon و jan و jen بيْد أنَّه لا يُطابق الأسماء التالية: jn و john و jordan و justin ومثيلاتها، لأنَّ الشرطة السفلية التي استخدمناها في الاستعلام تتخطى حرفًا واحدًا فقط، لذلك لن تُقبَل إلا الحقول المؤلفة من 3 أحرف.

يُطابق النمط التالي السلاسل النصية التالية: LaSt و LoSt و HaLt:

```
SELECT * FROM Employees WHERE FName LIKE '_A_T'
```

ج. العبارة ESCAPE في الاستعلام LIKE

يمكن إجراء بحث نصي في العبارة LIKE على النحو التالي:

```
SELECT *
FROM T_Whatever
WHERE SomeField LIKE CONCAT('%', @in_SearchText, '%')
```

ستحدث مشكلة إذا أدخل شخص ما نصًا مثل "50%" أو "a_b" (بصرف النظر عن حقيقة

أنـه يُفضِّـل البحث عن النص الكامـل بـدل اسـتخدام LIKE). يمكن حـل هـذه المشـكلة باسـتخدام عبارة LIKE:

```
SELECT *
FROM T_Whatever
WHERE SomeField LIKE CONCAT('%', @in_SearchText, '%') ESCAPE
'\'
```

هذا يعني أنَّ \ ستُعامل على أنَّها محرف تهريب (ESCAPE)؛ أي أنَّه يمكنـك الآن إضـافة \ إلى كل حرف في السلسلة النصية التي تبحث عنها، وستكون النتائج صحيحة، حتى لو أدخل المستخدم محارف خاصة مثل % أو _.

إليك المثال التالي:

```
string stringToSearch = "abc_def 50%";
string newString = "";
foreach(char c in stringToSearch)
     newString += @"\" + c;

sqlCmd.Parameters.Add("@in_SearchText", newString);
// بدلا من
// sqlCmd.Parameters.Add("@in_SearchText", stringToSearch);
```

انتبه إلى أنَّ الخوارزمية أعلاه للتوضيح وحسب، ولن تعمل في حال احتوت السلسـلة النصـية على وحـدة كتابيـة (grapheme) مؤلفـة من عـدَّة أحـرف (مثـل رمـوز utf-8). مثلًا، في السلسلة النصية:

```
string stringToSearch = "Les Mise\u0301rables";
```

ستحتاج إلى فعل ذلك لكل وحـدة كتابيـة (grapheme)، وليس لكـل حـرف (character). عليـك ألا تسـتخدم الخوارزميـة أعلاه إذا كنت تتعامـل مـع لغـات آسـيوية أو شـرق آسـيوية أو جنوب آسيوية.

د. البحث عن مجموعة من المحارف

تطابق العبارة التالية جميع السجلات التي يبدأ حقل FName خاصتها بحرف محصور (أبجديًا) بين A و F في جدول الموظفين:

```
SELECT * FROM Employees WHERE FName LIKE '[A-F]%'
```

ه. مطابقة نطاق أو مجموعة

يمكن مطابقة حرف واحد داخل نطاق مُحدِّد (على سبيل المثال: [a-f]) أو مجموعة (على سبيل المثال: [abcdef]).

يطابق نمط النطاق التالي السلسلة النصية gary، ولكن ليس mary:

```
SELECT * FROM Employees WHERE FName LIKE '[a-g]ary'
```

يُطابِق نمط المجموعة التالي mary ولكن ليس gary:

```
SELECT * FROM Employees WHERE Fname LIKE '[lmnop]ary'
```

يمكن أيضًا عكس أو نفي النطاق أو المجموعة بوضع العلامة ^ قبل النطاق أو المجموعة، فلن يتطابق نمط النطاق التالي مع gary، ولكنَّه سيتطابق مع mary:

```
SELECT * FROM Employees WHERE FName LIKE '[^a-g]ary'
```

لن يتطابق نمط المجموعة التالي مع mary ولكن سيتطابق مع gary:

```
SELECT * FROM Employees WHERE Fname LIKE '[^lmnop]ary'
```

و. أحرف البدل

يمكن استخدام أحرف البدل (Wildcard characters) مع المعامل LIKE. تُستخدَم أحرف البدل في SQL للبحث عن البيانات داخل جدول مُعيَّن. وهناك أربعة منها، وهي كالتالي:

- %: بديل عن صفر حرف أو أكثر

```
--   "Lo" اختيار جميع العملاء الذين يقطنون مدينة تبدأ بـ
 SELECT * FROM Customers
 WHERE City LIKE 'Lo%';

-- "es" اختيار جميع العملاء الذين يقطنون مدينة تحتوي
 SELECT * FROM Customers
 WHERE City LIKE '%es%';
```

- _: بديل عن حرف واحد:

```
-- اختيار جميع العملاء الذين يقطنون
-- erlin مدينة تبدأ بحرف معين، متبوعا بـ
SELECT * FROM Customers
WHERE City LIKE '_erlin';
```

- [charlist]: مجموعات ونطاقات مؤلَّفة من الحروف المُراد مُطابقتها:

```
-- اختيار جميع العملاء الذين يقطنون
-- "a" أو "d" أو "l" مدينة تبدأ بـ
SELECT * FROM Customers
WHERE City LIKE '[adl]%';
-- اختيار جميع العملاء الذين يقطنون
-- "a" أو "d" أو "l" مدينة تبدأ بـ
SELECT * FROM Customers
WHERE City LIKE '[a-c]%';
```

- [^ charlist]: تطابق الحروف غير الموجودة داخل القوسين المربعين:

```
-- اختيار جميع العملاء الذين يقطنون
-- "a" أو "d" أو "l" مدينة لا تبدأ بـ
SELECT * FROM Customers
WHERE City LIKE '[^apl]%';
or
SELECT * FROM Customers
WHERE City NOT LIKE '[apl]%' and city like '_%';
```

2. التحقق من الانتماء عبر IN

يمكن استخدام العبارة IN للتحقق من انتماء قيمة إلى مجموعة معينة.

تعيد الشيفرة التالية السجلات التي ينتمي مُعرِّفها `id` إلى مجموعة معينة من القيم (وهي 1,8,3):

```
select *
from products
where id in (1,8,3)LName
PhoneNumber
ManagerId
DepartmentId
Salary
Hire_date
Amudsen
2479100211
1
1
400
11-01-2010
Knag
2468021911
2
1
800
01-01-2000
```

يكافئ الاستعلام أعلاه:

```
select *
from products
where id = 1
 or id = 8
 or id = 3
```

يمكن استخدام IN مع استعلام فرعي على النحو التالي:

```
SELECT *
FROM customers
WHERE id IN (
 SELECT DISTINCT customer_id
 FROM orders
);
```

ستعيد الشيفرة أعلاه جميع العملاء الذين لديهم طلبات في النظام.

3. ترشيح النتائج باستخدام WHERE و HAVING

ا. ترشيح النتائج عبر BETWEEN

تستخدم الأمثلة التالية قاعدتي البيانات `Sales` و `Customers`. تذكر أنَّ المعامل `BETWEEN` شمولي (inclusive) أي يشمل المجال كلَّه.

استخدام BETWEEN مع الأعداد

يعيد الاستعلام التالي جميع سجلات `ItemSales` التي ينحصر حقل الكمية `quantity` خاصتها بين 10 و 17.

```
SELECT * From ItemSales
WHERE Quantity BETWEEN 10 AND 17
```

سنحصل على النتائج التالية:

Quantity	ItemId	SaleDate	Id
10	100	2013-07-01	1
15	100	2013-07-23	4
10	145	2013-07-24	5

استخدام BETWEEN مع قيم التاريخ

يعيد الاستعلام التالي كافة سجلات `ItemSales` التي ينحصر حقل `SaleDate` خاصتها بين التاريخين 11 يوليو 2013 و 24 مايو 2013.

```
SELECT * From ItemSales
WHERE SaleDate BETWEEN '2013-07-11' AND '2013-05-24'
```

هذا هو الخرج المتوقع:

Price	Quantity	ItemId	SaleDate	Id
34.5	20	100	2013-07-11	3
34.5	15	100	2013-07-23	4
34.5	10	145	2013-07-24	5

عند موازنة قيم الوقت (datetime) بدلًا من قيم التاريخ (dates)، قد تحتاج إلى تحويل قيم الوقت إلى قيم التاريخ، أو إضافة أو طرح 24 ساعة للحصول على النتيجة المتوقعة.

استخدام BETWEEN مع القيم النصية

يعيد الاستعلام التالي كافَّة العملاء الذين تنحصر أسماؤهم (أبجديًا) بين الحرفين 'D' و 'L'. في هذه الحالة، سيُعاد العميلان ذوا الرقمين 1 و 3. أمَّا العميل ذو الرقم 2 ، الذي يبدأ اسمه بالحرف "M"، فلن يُعَاد (المثال الحي):

```
SELECT Id, FName, LName FROM Customers
WHERE LName BETWEEN 'D' AND 'L';
```

الخرج:

LName	FName	Id
Jones	William	1
Davis	Richard	3

ب. استخدم HAVING مع الدوال التجميعية

على خلاف العبارة `WHERE`، يمكن استخدام `HAVING` مع الدوال التجميعية (aggregate functions) وهي دوال تأخذ القيم الموجودة في في عدَّة صفوف كمُدخلات (بناء على شروط مُحدَّدة) وتعيد قيمة مُعيَّنة. هذه بعض الدوال التجميعية: `COUNT()` و `SUM()` و `MIN()` و `MAX()`.

يستخدم هذا المثال الجدول `Car` من الفصل الأول.

```
SELECT CustomerId, COUNT(Id) AS [Number of Cars]
FROM Cars
GROUP BY CustomerId
HAVING COUNT(Id) > 1
```

يعيد الاستعلام أعلاه `CustomerId` وعدد السيارات `Number of Cars` لأيٍّ عميل لديه أكثر من سيارة واحدة. في هذا المثال، العميل الوحيد الذي لديه أكثر من سيارة واحدة هو العميل ذو الرقم 1.

هذا هو الخرج:

Number of Cars	CustomerId
2	1

ج. استخدام WHERE مع القيم NULL / NOT NULL

يعيد المثال التالي جميع سجلات الموظفين (Employee) التي تتساوى قيمة العمود `ManagerId` خاصتهم مع القيمة المعدومة `NULL`.

```
SELECT *
FROM Employees
WHERE ManagerId IS NULL
```

النتيجة:

```
Id     FName     LName     PhoneNumber     ManagerId
DepartmentId
1      James      Smith       1234567890         NULL            1
```

يعيد المثال التالي جميع سجلات الموظفين التي لا تساوي قيمة العمود `ManagerId` خاصتهم القيمة `NULL`.

```
SELECT *
FROM Employees
WHERE ManagerId IS NOT NULL
```

ستكون النتيجة كما يلي:

```
Id     FName         LName      PhoneNumber     ManagerId
DepartmentId
2      John            Johnson     2468101214       1
1
3      Michael        Williams    1357911131        1
2
4      Johnathon    Smith        1212121212        2
1
```

ملاحظة: لن يعيد الاستعلام أعلاه أيَّة نتائج في حال غيَّرت صياغة العبارة `WHERE` إلى: `WHERE ManagerId = NULL` أو `WHERE ManagerId <> NULL`.

د. معامل التساوي =

تعيد الشيفرة التالية كل صفوف الجدول Employees:

```
SELECT * FROM Employees
```

الخرج:

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId
Salary  Hire_date CreatedDate   ModifiedDate
1    James     Smith        1234567890      NULL          1
1000     01-01-2002  01-01-2002  01-01-2002
2    John       Johnson     2468101214      1                1
400       23-03-2005   23-03-2005  01-01-2002
3    Michael   Williams    1357911131      1                2
600      12-05-2009   12-05-2009  NULL
4    Johnathon Smith      1212121212      2                1
500       24-07-2016   24-07-2016  01-01-2002
```

يتيح لك استخدام WHERE في نهاية العبارة SELECT ترشيح الصفوف المُعَادة وفق شرط مُعيَّن. إن أردت مثلًا اشتراط التطابق التام مع قيمة مُعيَّنة، فاستخدم علامة التساوي =:

```
SELECT * FROM Employees WHERE DepartmentId = 1
```

لن يعيد الاستعلام أعلاه إلا الصفوف التي يساوي الحقل DepartmentId خاصتها القيمة 1:

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId
Salary  Hire_date       CreatedDate   ModifiedDate
1    James       Smith      1234567890       NULL          1
1000      01-01-2002    01-01-2002    01-01-2002
2    John          Johnson  2468101214       1
1                    400        23-03-2005    23-03-2005
01-01-2002
4    Johnathon Smith       1212121212       2                1
500        24-07-2016    24-07-2016     01-01-2002
```

لنفترض أنَّ متجرًا للألعاب لديه فئة من الألعاب يقل سعرها عن 10 دولارات، فيعيد الاستعلام التالي هذه الفئة من الألعاب:

```
SELECT *
FROM Items
WHERE Price < 10
```

ه. المعاملان المنطقيان AND و OR

يمكنك الجمع بين عدَّة معاملات معًا لإنشاء شروط WHERE أكثر تعقيدًا.

تستخدم الأمثلة التالية الجدول Employees التالي:

```
Id    FName      LName     PhoneNumber    ManagerId    DepartmentId
Salary     Hire_date        CreatedDate    ModifiedDate
1     James        Smith       1234567890       NULL            1
1000         01-01-2002     01-01-2002     01-01-2002
2     John          Johnson    2468101214       1
1                        400          23-03-2005     23-03-2005
01-01-2002
3     Michael      Williams   1357911131       1
2                        600          12-05-2009     12-05-2009
NULL
4     Johnathon   Smith      1212121212       2
1                        500          24-07-2016     24-07-2016
01-01-2002
```

إليك الاستعلام التالي:

```
SELECT * FROM Employees WHERE DepartmentId = 1 AND ManagerId =
1
```

سينتج الخرج التالي:

```
Id    FName      LName     PhoneNumber    ManagerId    DepartmentId
Salary   Hire_date        CreatedDate    ModifiedDate
```

```
2    John        Johnson   2468101214      1
1                    400         23-03-2005    23-03-2005
01-01-2002
```

وهذا استعلام آخر يستخدم المعامل المنطقي OR:

```
SELECT * FROM Employees WHERE DepartmentId = 2 OR ManagerId = 2
```

سينتج الخرج التالي:

```
Id   FName     LName    PhoneNumber   ManagerId   DepartmentId
Salary  Hire_date       CreatedDate   ModifiedDate
3    Michael     Williams   1357911131      1
2                    600         12-05-2009    12-05-2009
NULL
4    Johnathon Smith      1212121212      2                 1
500         24-07-2016     24-07-2016   01-01-2002
```

و. المعامل IN

يستخدم المثال التالي الجدول "Car":

```
SELECT *
FROM Cars
WHERE TotalCost IN (100, 200, 300)
```

سيعيد هذا الاستعلام السيارة رقم 2، والتي تبلغ تكلفتها 200، والسيَّارة ذات الرقم 3، والتي تساوي تكلفتها 100. لاحظ أنَّ الاستعلام أعلاه يكافئ استخدام OR عدَّة مرَّات كما هو موضَّح في المثال التالي:

```
SELECT *
FROM Cars
WHERE TotalCost = 100 OR TotalCost = 200 OR TotalCost = 300
```

ز. استعمال LIKE في البحث

يَستخدِم المثال التالي الجدول Car:

```
SELECT *
FROM Employees
WHERE FName LIKE 'John'
```

لن يعيد هذا الاستعلام إلا الموظف رقم 1، والذي يتطابق اسمه الأول مع السلسلة النصية "John".

```
SELECT *
FROM Employees
WHERE FName like 'John%'
```

يمكنك البحث عن سلسلة نصية فرعية عبر إضافة الرمز %:

- `John%` - تعيد أيَّ موظف يبدأ اسمه بالكلمة "John"، متبوعًا بأي عدد من الأحرف
- `%John` - تعيد أيَّ موظف ينتهي اسمه بالكلمة "John"، يعقبه أي عدد من الأحرف
- `%John%` - تعيد أيَّ موظف يحوي اسمه السلسلة النصية "John"

في المثال أعلاه، سيعيد الاستعلام الموظفَ رقم 2، والذي يحمل الاسم "John"، وكذلك الموظف رقم 4، والذي يحمل الاسم "Johnathon".

ح. العبارة Where EXISTS

في المثال التالي، تختار العبارة `WHERE EXISTS` سجلَّات `TableName` التي تطابق سجلاتِ في الجدول `TableName1`.

```
SELECT * FROM TableName t WHERE EXISTS (
 SELECT 1 FROM TableName1 t1 where t.Id = t1.Id)
```

ط. استخدام HAVING للتحقق من عدة شروط

إليك جدول الطلبات التالي:

Price	Quantity	ProductId	CustomerId
100	5	2	1
200	2	3	1
500	1	4	1
50	4	1	2
700	6	5	3

للحصـول على العملاء الـذين طلبـوا المنتَجين ذوي المُعـرِّف 2 و 3، يمكن اسـتخدام العبـارة HAVING:

```
select customerId
from orders
where productID in (2,3)
group by customerId
having count(distinct productID) = 2
```

الخرج الناتج:

customerId
1

لن يختـار الاسـتعلام إلا السـجلات ذات معرِّفـات المنتجـات المُحـدَّدة، والـتي تُحقِّق شـرط HAVING، أي وجود مُعرِّفين اثنين للمنتجات (productIds)، وليس معرِّفًا واحدًا فقط.

هذه صياغة أخرى:

```
select customerId
from orders
group by customerId
having sum(case when productID = 2 then 1 else 0 end) > 0
    and sum(case when productID = 3 then 1 else 0 end) > 0
```

لن يختار هذا الاستعلام إلا المجموعات التي لها سجل واحد على الأقل يساوي مُعـرِّف منتجـه (productID) القيمة 2، وسجل واحد على الأقل يساوي معرِّف منتجه 3.

4. ضبط عدد نتائج الاستعلام

يمكن وضع حدٍّ لعدد النتائج المُعادَة (Pagination) في استعلام معين، لكنَّ الصـياغة تختلـف بحسب النظام المُستخدَم:

- في إصدار SQL القياسي ISO / ANSI:

```
SELECT * FROM TableName FETCH FIRST 20 ROWS ONLY;
```

- MySQL و PostgreSQL و SQLite:

```
SELECT * FROM TableName LIMIT 20;
```

- Oracle:

```
SELECT Id,
   Col1
FROM (SELECT Id,
           Col1,
 row_number() over (order by Id) RowNumber
 FROM TableName)
WHERE RowNumber <= 20
```

- SQL Server:

```
SELECT TOP 20 *
FROM dbo.[Sale]
```

قد ترغب أحيانًا في تخطّي عدد من نتائج الاستعلام لأخذ النتائج التي تليها، ويمكنك ذلك عبر الصياغة التالية:

- ISO / ANSI SQL:

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS FETCH NEXT 20 ROWS ONLY;
```

- MySQL:

```
SELECT * FROM TableName LIMIT 20, 20; -- offset, limit
```

- Oracle و SQL Server:

```
SELECT Id,
   Col1
 FROM (SELECT Id,
           Col1,
 row_number() over (order by Id) RowNumber
 FROM TableName)
WHERE RowNumber BETWEEN 21 AND 40
```

- PostgreSQL و SQLite:

```
SELECT * FROM TableName LIMIT 20 OFFSET 20;
```

يمكنك كذلك تخطي بعض الصفوف من نتائج الاستعلام على النحو التالي:

- ISO / ANSI SQL:

```
SELECT Id, Col1
FROM TableName
ORDER BY Id
OFFSET 20 ROWS
```

- MySQL:

```
SELECT * FROM TableName LIMIT 20, 42424242424242;
```

تخطي 20 صفًا، وبالنسبة لعدد الصفوف المراد جلبها، استخدم عددًا كبيرًا يتجاوز عدد الصفوف في الجدول.

- Oracle:

```
SELECT Id,
   Col1
FROM (SELECT Id,
           Col1,
 row_number() over (order by Id) RowNumber
 FROM TableName)
WHERE RowNumber > 20
```

- PostgreSQL:

```
SELECT * FROM TableName OFFSET 20;
```

- SQLite:

```
SELECT * FROM TableName LIMIT -1 OFFSET 20;
```

5. تخطي مجموعة نتائج من استعلام

يمكنك استثناء مجموعة من البيانات عبر استخدام الكلمة المفتاحية EXCEPT، وإليك المثال التالي:

```
-- ينبغي أن تكون مجموعات البيانات متماثلة
SELECT 'Data1' as 'Column' UNION ALL
SELECT 'Data2' as 'Column' UNION ALL
SELECT 'Data3' as 'Column' UNION ALL
SELECT 'Data4' as 'Column' UNION ALL
SELECT 'Data5' as 'Column'
EXCEPT
SELECT 'Data3' as 'Column'
-- ==>  Data1 و Data2 و Data4 و Data5
```

6. استعمال EXPLAIN و DESCRIBE مع الاستعلامات

عند وضع `Explain` قُبالة استعلام `select`، سيعرض مُحرِّك قاعدة البيانات بعض البيانات التي توضِّح كيفيَّة تنفيذ الاستعلام. يمكنك استخدام هذه البيانات لفهم الشيفرة ومن ثَمَّ تحسينها، مثلًا، لو لاحظت أنَّ الاستعلام لا يستخدم فهرسًا، فيمكنك تحسين استعلامك عن طريق إضافة فهرس.

إليك الاستعلام التالي:

```
explain select * from user join data on user.test =
data.fk_user;
```

سنحصل على الخرج التالي:

```
id  select_type  table   type     possible_keys  key
key_len ref          rows  Extra
1   SIMPLE       user    index   test                  test
5           (null)       1      Using where; Using index
1   SIMPLE       data    ref       fk_user             fk_user
5          user.tes t 1     (null)
```

يُحدِّد العمود `type` ما إذا كان الاستعلام يستخدم عملية الفهرسة أم لا. وفي العمود `possible_keys`، سترى ما إذا كان بالإمكان تنفيذ الاستعلام بواسطة فهارس أخرى إن لم يتوافر

فهرس. يعرض key الفهرس المستخدم، فيما يعرض العمود key_len حجم عنصر من الفهرس (بالبايتات bytes)، وكلما انخفضت هذه القيمة، زاد عدد عناصر الفهرس التي يمكن تخزينها في مساحة معينة من الذاكرة، وهو ما يسرِّع معالجتها. يعرض العمود rows العدد المتوقع للصفوف التي يحتاج الاستعلام إلى جردها، وكلما كان هذا العدد أصغر، كان الأداء أفضل.

في الجهة المقابلة، DESCRIBE و EXPLAIN متماثلتان، بيد أنَّ DESCRIBE تعيد معلومات تعريفية لأعمدة الجدول tablename:

```
DESCRIBE tablename;
```

النتيجة:

```
COLUMN_NAME      COLUMN_TYPE      IS_NULLABLE      COLUMN_KEY
COLUMN_DEFAULT    EXTRA
id                                int(11)
NO                          PRI                              0
auto_increment
test                              varchar(255)               YES
(null)
```

عُرِضَت أسماء الأعمدة متبوعة بنوعها، إضافة إلى توضيح ما إذا كان من الممكن استخدام null في العمود، وما إذا كان العمود يستخدم فهرسًا. تُعرض أيضًا القيمة الافتراضية للعمود، وما إذا كان الجدول ينطوي على أي سلوك خاص، مثل auto_increment.

7. العبارة EXISTS

إليك جدول العملاء التالي:

LastName	FirstName	Id
Ozturk	Ozgur	1
Medi	Youssef	2

LastName	FirstName	Id
Tai	Henry	3

وهذا جدول آخر للطلبيات:

Amount	CustomerId	Id
123.50	2	1
14.80	3	2

تعيد هذه الشيفرة جميع العملاء الذين قدموا طلبية واحدة على الأقل:

```
SELECT * FROM Customer WHERE EXISTS (
 SELECT * FROM Order WHERE Order.CustomerId=Customer.Id
)
```

النتيجة المتوقعة:

LastName	FirstName	Id
Medi	Youssef	2
Tai	Henry	3

يعيد الاستعلام التالي جميع العملاء الذين لم يقدِّموا أيَّ طلبية:

```
SELECT * FROM Customer WHERE NOT EXISTS (
 SELECT * FROM Order WHERE Order.CustomerId = Customer.Id
)
```

النتيجة المتوقَّعة:

LastName	FirstName	Id
Ozturk	Ozgur	1

مبدأ عمل `EXISTS` و `IN` و `JOIN` متشابه إلى درجة ما، بيْد أنَّ هناك اختلافات في كيفية عمـل كل منها:

- تُستخدَم `EXISTS` للتحقق من وجود قيمة في جدول آخر.
- تُستخدَم `IN` للحصول على قائمة ثابتة.
- تُستخدَم `JOIN` لاسترجاع البيانات من جداول أخرى.

6

إنشاء الجداول وتحديثها وحذفها

يستعرض هذا الفصل كيفية تحديث قواعد البيانات، وكيفية إنشاء قواعد بيانات وجداول ودوال جديدة.

1. إنشاء جدول جديد

تُستخدَم العبارة `CREATE TABLE` لإنشاء جدول جديد في قاعدة البيانات. يتألَّف تعريف الجدول من قائمة من الأعمدة وأنواعها، إضافة إلى فرض أيَّة قيود عليها.

صياغة إنشاء جدول جديد:

```
CREATE TABLE tableName columns;
```

- `tableName`: اسم الجدول
- `columns`: قائمة مفصولة بفاصلة أجنبية تصف الأعمدة الموجودة في الجدول

تنشئ الشيفرة التالية جدولًا بسيطًا باسم (`Employees`)، يتألَّف من مُعرِّف، واسم الموظف الأول، واسم العائلة، إضافة إلى رقم الهاتف:

```
CREATE TABLE Employees(
    Id int identity(1,1) primary key not null,
    FName varchar(20) not null,
    LName varchar(20) not null,
    PhoneNumber varchar(10) not null
);
```

هذا المثال خاص بالإصدار Transact-SQL.

تنشئ العبارة `CREATE TABLE` جدولًا جديدًا وتضيفه إلى قاعدة البيانات، وتُتبَع هذه العبارة باسم الجدول (Employees) ثم تعقبه قائمة بأسماء الأعمدة وخصائصها، مثل المُعرِّف ID.

```
Id int identity(1,1) not null
```

إليك شرح الشيفرة:

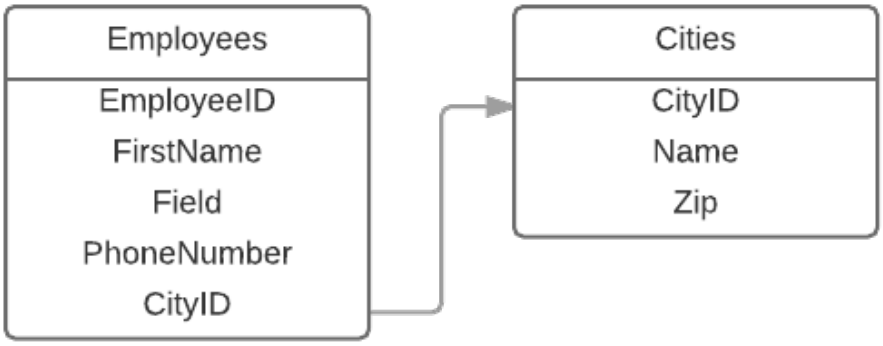
- `Id`: اسم العمود
- `int`: نوع البيانات
- `identity(1,1)`: ينصّ على أنَّ قيم العمود ستُنشأ تلقائيًا، بداية من 1، ثم تزداد بمقدار 1 في كل صف جديد
- `primary key`: تنص على أنَّ قيم هذا العمود فريدة وغير مكرَّرة
- `not null`: تنص على أنَّ قيم العمود لا يمكن أن تكون معدومة

ا. إنشاء جدول باستخدام مفتاح خارجي FOREIGN KEY

ينشئ المثال أدناه جدولًا للموظفين `Employees` يحتوي مرجعًا إلى جدول آخر، وهو جدول المدن `Cities`.

```
CREATE TABLE Cities(
    CityID INT IDENTITY(1,1) NOT NULL,
    Name VARCHAR(20) NOT NULL,
    Zip VARCHAR(10) NOT NULL
);
CREATE TABLE Employees(
    EmployeeID INT IDENTITY (1,1) NOT NULL,
    FirstName VARCHAR(20) NOT NULL,
    LastName VARCHAR(20) NOT NULL,
    PhoneNumber VARCHAR(10) NOT NULL,
    CityID INT FOREIGN KEY REFERENCES Cities(CityID)
);
```

يستعرض الرسم التالي مخططًا توضيحيًا للعلاقة بين قاعدتي البيانات.



لاحظ أنَّه في السطر الأخير من الشيفرة، يشير العمود `CityID` من الجدول `Employees` إلى العمود `CityID` في الجدول `Cities`:

```
CityID INT FOREIGN KEY REFERENCES Cities(CityID)
```

شرح الشيفرة:

- `CityID`: اسم العمود
- `int`: نوع العمود
- `FOREIGN KEY`: إنشاء المفتاح الخارجي (اختياري)
- `REFERENCES Cities(CityID)`: إنشاء مرجع إلى العمود CityID من جدول المدن

تنبيه: لا يجوز إنشاء مرجع إلى جدول غير موجود في قاعدة البيانات. عليك أن تنشئ الجدول `Cities` أولًا، ثم تنشئ الجدول `Employees` عقب ذلك. إذا فعلت ذلك بترتيب معكوس، فسيُطرَح خطأ.

ب. إنشاء جدول عبر Select

يمكنك إنشاء نسخة مكرَّرة من جدول معيَّن على النحو التالي:

```
CREATE TABLE ClonedEmployees AS SELECT * FROM Employees;
```

يمكنك استخدام أيٍّ من الميزات الأخرى لعبارة SELECT لتعديل البيانات قبل نقلها إلى الجدول الجديد.

تُنشَأ أعمدة الجدول الجديد تلقائيًا انطلاقًا من الصفوف المُختارة.

```
CREATE TABLE ModifiedEmployees AS
SELECT Id, CONCAT(FName," ",LName) AS FullName FROM Employees
WHERE Id > 10;
```

ج. استنساخ جدول

يمكنك إنشاء نسخة طبق الأصل من جدول مُعيَّن عبر الصياغة التالية:

```
CREATE TABLE newtable LIKE oldtable;
INSERT newtable SELECT * FROM oldtable;
```

د. إنشاء جدول مؤقت

يمكن إنشاء جدول مؤقت خاص بالجلسة (session) الحالية لكنَّ الصياغة تختلف بحسب إصدار SQL:

- PostgreSQL و SQLite

```
CREATE TEMP TABLE MyTable(...);
```

- SQL Server

```
CREATE TABLE #TempPhysical(...);
```

يمكن كذلك إنشاء جدول مؤقت مرئي للجميع، وليس حصرًا على الجلسة الحالية على النحو التالي:

```
CREATE TABLE ##TempPhysicalVisibleToEveryone(...);
```

ويمكن أيضًا إنشاء جدول في الذاكرة:

```
DECLARE @TempMemory TABLE(...);
```

2. إنشاء قاعدة بيانات جديدة

يمكن إنشاء قاعدة بيانات باستخدام أمر SQL التالي:

```
CREATE DATABASE myDatabase;
```

سيُنشِئ الأمر أعلاه قاعدة بيانات فارغة باسم `myDatabase`، والتي يمكن أن تضيف جداول إليها باستخدام العبارة `CREATE TABLE`.

3. إنشاء دالة جديدة

تتيح SQL إنشاء دالة جديدة. فينشئ المثال التالي دالة باسم `FirstWord`، والتي تقبل معاملًا من النوع `varchar`، وتُعيد قيمة من النوع نفسه.

```
CREATE FUNCTION FirstWord (@input varchar(1000))
RETURNS varchar(1000)
AS
BEGIN
     DECLARE @output varchar(1000)
     SET @output = SUBSTRING(@input, 0, CASE CHARINDEX(' ',
@input)
          WHEN 0 THEN LEN(@input) + 1
          ELSE CHARINDEX(' ', @input)
     END)
     RETURN @output
END
```

شرح الشيفرة:

- `function_name`: اسم الدالة
- `list_of_paramenters`: معاملات الدالة

- `return_data_type`: نوع القيمة المُعادة
- `function_body`: متن الدالة
- `scalar_expression`: القيمة العددية المُعادة من الدالة

4. تعديل معمارية جدول

تُستخدَم العبارة `ALTER` في SQL لتَعدِيل قيد (constraint) أو عمود من جدول.

ا. إضافة عمود

يضيف المثال التالي عمودين إلى جدول الموظفين، الأول باسم `StartingDate`، ولا يمكن أن يكون معدومًا (not NULLABLE)، وقيمته الافتراضية تساوي التاريخ الحالي، أمَّا العمود الثاني، فيُسمَّى `DateOfBirth`، ويمكن أن يكون معدومًا.

```
ALTER TABLE Employees
ADD StartingDate date NOT NULL DEFAULT GetDate(),
    DateOfBirth date NULL
```

ب. حذف عمود

تحذف الشيفرَة أدناه العمود `salary` من جدول الموظفين:

```
ALTER TABLE Employees
DROP COLUMN salary;
```

ج. إضافة مفتاح رئيسي Primary Key

تضيف الشيفرة أدناه مفتاحًا رئيسيًا إلى جدول "الموظفين" في الحقل `ID`:

```
ALTER TABLE EMPLOYEES ADD pk_EmployeeID PRIMARY KEY (ID)
```

يؤدِّي دمج عدَّة أسماء أعمدة بين قوسين بعد العبارة PRIMARY KEY إلى إنشاء مفتاح رئيسي مُركَّب (Composite Primary Key):

```
ALTER TABLE EMPLOYEES ADD pk_EmployeeID PRIMARY KEY (ID, FName)
```

د. تغيير عمود

يُعدِّل الاستعلام التالي نوع بيانات العمود StartingDate، ويغيِّره من النوع date إلى datetime، كما يعيِّن قيمته الافتراضية عند قيمة التاريخ الحالي.

```
ALTER TABLE Employees
ALTER COLUMN StartingDate DATETIME NOT NULL DEFAULT (GETDATE())
```

ه. حذف القيود

تحذف الشيفرة التالية قيدًا يُسمّى DefaultSalary من جدول الموظفين:

```
ALTER TABLE Employees
DROP CONSTRAINT DefaultSalary
```

تنبيه: عليك حذف قيود العمود قبل حذف العمود.

---

5. إضافة بيانات لجدول

ا. إدراج البيانات من جدول آخر

يدرج المثال التالي جميع الموظفين (Employees) في جدول العملاء (Customers) ونظرًا لأنَّ الجدولين يحتويان على حقول مختلفة، وقد لا تريد نقل جميع الحقول، فستحتاج إلى تحديد الحقول التي ستُدرج القيم فيها، والحقول التي يجب اختيارها.

لست مضطرًا للحفاظ على أسماء الحقول المترابطة (correlating field)، بيد أنَّه ينبغي عدم تغيير نوع البيانات.

يفترض المثال التالي أنَّ هويَّة حقل المُعرِّف (Id) مُحدَّدة، وأنَّه متزايد القيمة تلقائيًا (auto increment).

```
INSERT INTO Customers (FName, LName, PhoneNumber)
SELECT FName, LName, PhoneNumber FROM Employees
```

إن كان للجدولين نفس أسماء الحقول، وكنت تريد نقل جميع السجلات من جدول إلى آخر، فيمكنك استخدام الشيفرة التالية:

```
INSERT INTO Table1
SELECT * FROM Table2
```

ب. إدراج صف جديد

تدرج الشيفرة التالية صفًّا جديدًا في الجدول `Customers`. لاحظ أنَّنا لم نُحدِّد قيمة العمود `Id`، ذلك أنَّ قيمته ستضاف تلقائيًا أمَّا الأعمدة الأخرى فينبغي أن تُعيَّن قيمها.

```
INSERT INTO Customers
VALUES ('Zack', 'Smith', 'zack@example.com', '7049989942',
'EMAIL');
```

ج. إدراج أعمدة مُعيَّنة

تدرج الشيفرة التالية صفًّا جديدًا في الجدول `Customers`، لاحظ أنَّها لن تُدرِج البيانات إلا في الأعمدة المُحدَّدة:

```
INSERT INTO Customers (FName, LName, Email, PreferredContact)
VALUES ('Zack', 'Smith', 'zack@example.com', 'EMAIL');
```

لاحظ أنَّه لم يتم تقديم أيِّ قيمة للعمود PhoneNumber ولاحظ أيضًا أنَّه ينبغي دمج الأعمدة الموسومة بـ not null.

د. إدراج عدَّة صفوف دفعة واحدة

يمكن إدراج عدَّة صفوف باستخدام أمر إدراج واحد على النحو التالي:

```
INSERT INTO tbl_name (field1, field2, field3)
VALUES (1,2,3), (4,5,6), (7,8,9);
```

قد تحتاج أحيانًا إلى إدراج كمِّيات كبيرة من البيانات دُفعة واحدة، فعادة ما توفِّر أنظمة إدارة قواعد البيانات بعض التوصيات والميزات للمساعدة على ذلك، كما في حالة: MySQL و MSSQL.

6. تحديث بيانات جدول

تُستخدَم الكلمة المفتاحية UPDATE لتحديث بيانات الجداول.

ا. تحديث جدول من بيانات جدول آخر

تملأ الأمثلة الموضحة أدناه الحقل PhoneNumber لأي موظف يكون أيضًا عميلًا Customer، وليس له حاليًا رقم هاتف في جدول الموظفين Employees.

(تستخدم الأمثلة التالية جدولي الموظفين Employees والعملاء Customers.)

- SQL القياسية - يُحدِّث المثال التالي قاعدة البيانات باستخدام استعلام فرعي مربوط (correlated subquery):

```
UPDATE
    Employees
SET PhoneNumber =
   (SELECT
         c.PhoneNumber
   FROM
```

```
        Customers c
   WHERE
        c.FName = Employees.FName
        AND c.LName = Employees.LName)
WHERE Employees.PhoneNumber IS NULL
```

- SQL:2003 - تحديث باستخدام `MERGE`:

```
MERGE INTO
    Employees e
USING
    Customers c
ON
    e.FName = c.Fname
   AND e.LName = c.LName
   AND e.PhoneNumber IS NULL
WHEN MATCHED THEN
  UPDATE
       SET PhoneNumber = c.PhoneNumber
```

- SQL Server - تحديث باستخدام `INNER JOIN`:

```
UPDATE
    Employees
SET
    PhoneNumber = c.PhoneNumber
FROM
    Employees e
INNER JOIN Customers c
 ON e.FName = c.FName
 AND e.LName = c.LName
WHERE
    PhoneNumber IS NULL
```

ب. تعديل القيم الحالية

يستخدم هذا المثال "الجدول Car" من الفصل الأول.

تتيح لك SQL إمكانيَّة استخدام القيم القديمة في عمليات التحـديث. في المثـال التـالي، تُزَاد قيمة `TotalCost` بمقدار 100 في الصفين ذوي المعرِّفين 3 و 4:

```
UPDATE Cars
SET TotalCost = TotalCost + 100
WHERE Id = 3 or Id = 4
```

- زِيدَت قيمة `TotalCost` الخاصَّة بالسيارة 3 من 100 إلى 200.
- زيدت قيمة `TotalCost` الخاصَّة بالسيارة 4 من 1254 إلى 1354.

يمكن اشتقاق القيمة الجديدة للعمود من قيمته السابقة، أو من قيمة أيِّ عمود آخر في الجدول أو من الجدول المُدمَج (joined table) نفسه.

ج. تحديث صفوف معيَّنة

تُعيِّن الشيفرة أدناه قيمة حالة الصف ذي المُعرِّف 4 إلى `READY`.

```
UPDATE
    Cars
SET
    Status = 'READY'
WHERE
    Id = 4
```

تُقيِّم عبارة `WHERE` شرطًا منطقيًا في كل صفٍّ. ، وإذا استوفى الصـف ذلـك الشـرط، فسـتُحدَّث قيمته؛ خلاف ذلك، يظل الصف دون تغيير.

د. تحديث جميع الصفوف

يُعيِّن المثال التالي العمـود "status" الخـاص بجميـع صـفوف الجـدول "Cars" إلى القيمـة "READY"، إذ يشمل التحديث جميع الصفوف بسبب غياب العبارة `WHERE` (التي ترشِّح الصفوف).

```
UPDATE Cars
SET Status = 'READY'
```

ه. التقاط السجلات المُحدَّثة

قد ترغب في بعض الأحيان في التقاط السجلات التي حُدِّثت للتو، ويمكنك ذلك عبر الصياغة التالية:

```
CREATE TABLE #TempUpdated(ID INT)
Update TableName SET Col1 = 42
    OUTPUT inserted.ID INTO #TempUpdated
    WHERE Id > 50
```

7. التحديث عبر الدمج

تسمح العبارة MERGE (تُسمَّى أيضًا UPSERT) بإدراج صفوف جديدة أو تحديثها، والهدف من هذه العملية هو إجراء مجموعة كاملة من العمليات تلقائيًا (لضمان بقاء البيانات متسقة)، ومنع تحميل الاتصال حملًا زائدًا (communication overhead) لعبارات SQL في نظام عميل / خادم (client/server system).

ا. إجراء عملية الدمج لجعل المصدر يطابق الهدف

يجري المثال التالي عملية دمج (MERGE) لجعل الجدول المصدري يطابق الجدول الهدف:

```
MERGE INTO targetTable t
    USING sourceTable s
        ON t.PKID = s.PKID
     WHEN MATCHED AND NOT EXISTS (
            SELECT s.ColumnA, s.ColumnB, s.ColumnC
            INTERSECT
            SELECT t.ColumnA, t.ColumnB, s.ColumnC
            )
```

```
    THEN UPDATE SET
           t.ColumnA = s.ColumnA
        ,t.ColumnB = s.ColumnB
        ,t.ColumnC = s.ColumnC
    WHEN NOT MATCHED BY TARGET
        THEN INSERT (PKID, ColumnA, ColumnB, ColumnC)
        VALUES (s.PKID, s.ColumnA, s.ColumnB, s.ColumnC)
    WHEN NOT MATCHED BY SOURCE
        THEN DELETE
    ;
```

ملاحظة: تمنع العبارة `AND NOT EXISTS` تحديث السجلات التي لم تتغير. ويتيح استخدام البنية `INTER SECT` موازنة الأعمدة المعدومة (nullable) دون مشاكل.

ب. عدُّ أسماء المستخدمين عبر MySQL

لنفترض أنَّنا نريد أن نحسب عدد المستخدمين الذين لهم نفس الاسم. سننشئ أولًا جدولًا باسم `users` على النحو التالي:

```
create table users(
    id int primary key auto_increment,
    name varchar(8),
    count int,
    unique key name(name)
);
```

لنفترض الآن أنَّنا اكتشفنا وجود مستخدم آخر جديد يُدعَى `Joe`، ونودُّ أن نأخذه في الحسبان. أولًا، سنتحقَّق ممَّا إذا كان هناك صف يحمل ذلك الاسم، فإنَّ كان الأمر كذلك، حدِّثناه وزدنا قيمته بواحد أو ننشئ صفًّا جديدًا.

تَستخدِم MySQL الصياغة التالية: insert ... on duplicate key update ...:

```
insert into users(name, count)
    values ('Joe', 1)
    on duplicate key update count=count+1;
```

ج. عدُّ أسماء المستخدمين عبر PostgreSQL

سنعيد المثال التالي ولكن في بيئة PostgreSQL. سننشئ أولًا جدولًا باسم users على النحو التالي:

```
create table users(
    id serial,
    name varchar(8) unique,
    count int
);
```

كما في المثال السابق، سنفترض أنَّنا اكتشفنا للتو مستخدمًا جديدًا يُدعَى Joe، ونود أن نأخذه في الحسبان:

تستخدم PostgreSQL الصياغة التالية: insert ... on conflict ... do update ...:

```
insert into users(name, count)
    values('Joe', 1)
    on conflict (name) do update set count = users.count + 1;
```

8. حذف الجداول أو قواعد البيانات

يتحدَّث هذا الفصل عن كيفية حذف الجداول (DROP) وقواعد البيانات (DELETE)، واقتطاع الجداول (TRUNCATE TABLE)، وكيفية استخدام الحذف المتفشي أو المنتشر (Cascading Delete) في SQL.

ا. الحذف باستخدام DELETE

تُستخدَم عبارة DELETE لحذف السجلات من جدول معيَّن.

حذف جميع الصفوف

ستُحذف جميع الصفوف من الجدول عند استخدام DELETE بدون عبارة WHERE:

```
DELETE FROM Employees
```

على العموم، أداء العبارة TRUNCATE (انظر الفقرة أدناه) أفضل من أداء DELETE، لأنَّها تتجاهل المنبهات (triggers) والفهارس وتحذف البيانات مباشرة.

استخدام DELETE مع WHERE

ستحذف الشيفرة التالية جميع الصفوف التي تفي بشرط WHERE، أي الصفوف التي تمثل الموظفين الذين يحملون الاسم John:

```
DELETE FROM Employees
WHERE FName = 'John'
```

من الممكن حذف البيانات (DELETE) من جدول إذا كانت مطابقة (أو غير مطابقة) لبيانات جدول آخر.

لنفترض أنَّنا نريد حذف البيانات من الجدول المصدري بمجرَّد تحميلها إلى الجدول الهدف.

```
DELETE FROM Source
WHERE EXISTS ( SELECT 1 --القيمة المحدّدة في SELECT ليست مهمّة
 FROM Target
 Where Source.ID = Target.ID )
```

تسمح معظم أنظمة معالجة قواعد البيانات (RDBMS) الشهيرة (مثل MySQL و Oracle و PostgresSQL و Teradata)ـ بدمج الجداول خلال عملية الحذف DELETE، ممَّا يتيح إجراء موازنات مُعقَّدة في عبارات قصيرة.

لنفترض الآن أنَّنا نريد تجميع (Aggregate) جدول من الجدول الهدف على أساس التاريخ

date وليس المُعرِّف ID. لنفترض أيضًا أنَّنا نريد ألا تُحذف البيانات من المصدر إلا بعد أن يُملَأ حقل التاريخ Date الخاص بالجدول المُجمَّع (aggregate).

في أنظمة MySQL و Oracle و Teradata، يمكن القيام بذلك باستخدام:

```
DELETE FROM Source
WHERE Source.ID = TargetSchema.Target.ID
     AND TargetSchema.Target.Date =
AggregateSchema.Aggregate.Date
```

أمَّا في PostgreSQL، فاستخدم الصياغة التالية:

```
DELETE FROM Source
USING  TargetSchema.Target, AggregateSchema.Aggregate
WHERE Source.ID = TargetSchema.Target.ID
     AND TargetSchema.Target.DataDate =
AggregateSchema.Aggregate.AggDate
```

ينتج عن هذا أساسًا عمليات دمج داخلي (INNER JOIN) بين الجدول المصدري والجدول الهدف والجدول المُجمَّع (Aggregate) يُنفَّذ الحذف على الجدول المصدري في حال وجود نفس المُعرِّفات في الهدف، وكذلك في حال تساوي التاريخين `date` في الجدول الهدف وكذلك في الجدول المُجمَّع.

يمكن كتابة الاستعلام نفسه (في MySQL و Oracle و Teradata) على النحو التالي:

```
DELETE Source
FROM Source, TargetSchema.Target, AggregateSchema.Aggregate
WHERE Source.ID = TargetSchema.Target.ID
 AND TargetSchema.Target.DataDate =
AggregateSchema.Aggregate.AggDate
```

يمكن في بعض أنظمة إدارة قواعد البيانات (مثل Oracle و MySQL) استخدام عمليات الدمج (joins) صراحة في عبارات `Delete`، بيد أنَّها غير مدعومة في جميع المنصات (كما هو الحال في Teradata).

يمكن إجراء عمليات الموازنة للتحقق من سيناريوهات عدم التطابق بدلًا من سيناريوهات التطابق مع جميع أنماط الصياغات (لاحظ NOT EXISTS أدناه):

```
DELETE FROM Source
WHERE NOT EXISTS ( SELECT 1 -- القيم المحدّدة في SELECT لا تهمّ
 FROM Target
 Where Source.ID = Target.ID )
```

ب. الاقتطاع عبر TRUNCATE

تحذف عبارة TRUNCATE كافة البيانات من الجدول، فهي تكافئ إجراء عملية الحذف DELETE بدون ترشيح، ولكن قد تكون لها بعض القيود أو التحسينات اعتمادًا على برنامج قواعد البيانات المُستخدَم.

يزيل المثال التالي جميع الصفوف من جدول الموظفين Employee:

```
TRUNCATE TABLE Employee;
```

يُفضَّل عمومًا استخدام TRUNCATE على DELETE، لأنَّها تتجاهل جميع الفهارس والمنبهات (triggers)، وتزيل العناصر مباشرة. حذف الجداول (DELETE) هي عملية تعمل على الصفوف، بمعنى أنَّها تحذف كل صف على حدة. أمَّا اقتطاع الجداول (TRUNCATE)، فهي عملية تعمل على صفحة كاملة من البيانات (page operation)، إذ يُعَاد تخصيص (reallocate) صفحة البيانات بأكملها. إذا كان لديك جدول يحتوي مليون صف، فسيكون اقتطاع الجدول أسرع بكثير من استخدام عبارة حذف DELETE الجدول. بالمقابل، يمكننا تحديد الصفوف المراد حذفها باستخدام DELETE، ولكن لا يمكننا تحديد الصفوف المراد اقتطاعها باستخدام TRUNCATE، إذ لا يمكننا سوى اقتطاع جميع السجلات مرَّة واحدة.

يؤدِّي حذف جميع الصفوف (عبر DELETE) ثم إدراج سجلات جديدة إلى زيادة قيمة المفتاح الرئيسي المتزايد تلقائيًا (Auto incremented Primary key) انطلاقًا من القيمة المُدرجَة سابقًا،

أمَّا في عبارة TRUNCATE، فسيُعَاد تعيين قيمة المفتاح الرئيسي التلقائي، وسيبدأ من القيمة 1. ولاحظ أنَّه عند اقتطاع جدول ما، يجب ألا تكون هناك مفاتيح خارجية (foreign keys)، وإلا فستحصل على خطأ.

ج. محو الجداول عبر DROP

تحذف عبارة DROP TABLE جدولًا مع بياناته من قاعدة البيانات بشكل دائم.

الأمثلة التالية تتحقَّق من وجود الجدول قبل محوه:

- MySQL ≥ 3.19

```
DROP TABLE IF EXISTS MyTable;
```

- PostgreSQL ≥ 8.x

```
DROP TABLE IF EXISTS MyTable;
```

- SQL Server ≥ 2005

```
If Exists(Select * From Information_Schema.Tables
      Where Table_Schema = 'dbo'
         And Table_Name = 'MyTable')
    Drop Table dbo.MyTable
```

- SQLite ≥ 3.0

```
DROP TABLE IF EXISTS MyTable;
```

د. محو قاعدة بيانات

يمكن محو قاعدة البيانات باستخدام عبارة DROP DATABASE.

تمحو الشيفرة التالي قاعدة بيانات الموظفين:

```
DROP DATABASE [dbo].[Employees]
```

تنبيه: تحذف DROP DATABASE قاعدة البيانات نهائيًا، لذا عليك أن تحرص دائمًا على تخزين نسخة احتياطية من قاعدة البيانات إن خشيت ضياع البيانات.

ه. الحذف المتفشي

لنفترض أنَّ لديك تطبيقًا يدير فندقًا يدمج عددًا من الغرف. لنفترض أنَّ لديك عملاء كثر، وقد قَرَّرَت إنشاء قاعدة بيانات لتخزين المعلومات الخاصة بعملائك.

ستحوي قاعدة البيانات جدولًا واحدًا للعملاء، وآخر للغرف؛ فكل عميل يمكن أن يستأجر N غرفة. هذا يعني أنَّ جدول الغرف سيحوي مفتاحًا خارجيًا (foreign key) يشير إلى جدول العملاء.

```
ALTER TABLE dbo.T_Room WITH CHECK ADD CONSTRAINT
FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
GO
```

عند خروج أحد العملاء، سيتعيَّن عليك حذف بياناته من البرنامج لكن إن كتبت:

```
DELETE FROM T_Client WHERE CLI_ID = x
```

فسترتَكِب انتهاكًا بحق «المفتاح الخارجي» (foreign key violation)، ذلك أنَّه لا يجوز لك حذف عميل لديه غرفة.

عليك حذف غرف العميل قبل أن تحذف العميل ولكن لنفترض توقعك إضافة العديد من المفاتيح الخارجية (foreign key dependencies) في قاعدة البيانات مستقبلًا نتيجةً لنمو التطبيق، فقد يخلق هذا مشكلة كبيرةً لأنَّه في كل مرَّة تعدِّل قاعدة البيانات، سيكون عليك تعديل شيفرة تطبيقك في كل المواضِع المرتبطة بها. وقد يكون عليك أيضًا تعديل شيفرات تطبيقات أخرى (مثل الواجهات البرمجية للأنظمة الأخرى).

هناك حل أفضل يكفيك كل هذا العناء، فيكفي أن تضيف العبارة ON DELETE CASCADE إلى مفتاحك الخارجي:

```
ALTER TABLE dbo.T_Room -- WITH CHECK -- SQL-Server can specify
WITH CHECK/WITH NOCHECK
ADD CONSTRAINT FK_T_Room_T_Client FOREIGN KEY(RM_CLI_ID)
REFERENCES dbo.T_Client (CLI_ID)
ON DELETE CASCADE
```

الآن يمكنك أن تكتب:

```
DELETE FROM T_Client WHERE CLI_ID = x
```

وستُحذَف الغرف تلقائيًا عند حذف العميل. لقد حللنا المشكلة دون الحاجة إلى إجراء تغييرات في شيفرة التطبيق.

انتبه إلى أنَّ هذه المقاربة لن تنجح في Microsoft SQL-Server إذا كان الجدول يشير إلى نفسه؛ لذا إن حاولت إجراء حذف متسلسل على بنية مُتشعِّبة عودية (recursive tree structure)، على النحو التالي:

```
IF NOT EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND
parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation] WITH CHECK ADD CONSTRAINT
[FK_T_FMS_Navigation_T_FMS_Navigation] FOREIGN KEY([NA_NA_UID])
REFERENCES [dbo].[T_FMS_Navigation] ([NA_UID])
ON DELETE CASCADE
GO
IF EXISTS (SELECT * FROM sys.foreign_keys WHERE object_id =
OBJECT_ID(N'[dbo].[FK_T_FMS_Navigation_T_FMS_Navigation]') AND
parent_object_id =
OBJECT_ID(N'[dbo].[T_FMS_Navigation]'))
ALTER TABLE [dbo].[T_FMS_Navigation] CHECK CONSTRAINT
```

```
[FK_T_FMS_Navigation_T_FMS_Navigation]
GO
```

فلن ينجح الأمـر، لأنَّ Microsoft-SQL-server لن تسـمح لـك بتعـيين مفتـاح خـارجي باستخدام `ON DELETE CASCADE` على بنية مُتشعِّبة عودية. أحد أسباب ذلك هو أنَّ الشعبة قد تكون دورية، وهذا قد يؤدي إلى عملية سرمدية غير منتهية.

نظام PostgreSQL -من ناحية أخرى- يمكنه القيام بذلك شريطة ألَّا تكون الشعبة دورية (non-cyclic). إذ أنَّه في حال كانت الشعبة دورية، فسيُطرَح خطأ وقت التشغيل؛ الحل في مثل هذه الحالة هو إنشاء دالة حذف مُخصَّصة.

تنبيه: لا يمكنك حذف جدول العملاء وإعادة إدراج القيم مرَّة أخرى، وإن حاولت ذلك، فستُحذَف جميع المدخلات في الجدول `T_Room`.

---

7 الدمج بين الجداول

تـدمج العبـارة JOIN البيانـات من جـدولين، وتعيـد مجموعـة مختلطـة من الأعمـدة من كلا الجـدولين، وذلـك حسـب نـوع الـدمج أو الـدمج المُسـتخدم، ومعـاييره (كيفيـة ربـط الصـفوف من كلا الجدولين).

يمكن دمج جـدولٍ مـع نفسـه، أو مـع أي جـدول آخـر، وإذا كـانت هنـاك حاجـة للوصـول إلى معلومات من أكثر من جدولين، فيمكن استخدام الدمج عدَّة مرَّات في عبارة FROM.

ا. الدمج الذاتي (Self Join)

يمكن دمج جدول إلى نفسـه، بحيث تتطـابق الصـفوف مـع بعضـها بعضًـا وفـق شـروط معينـة، ويجب استخدام الكنى (aliases) -في مثل هذه الحالة- للتمييز بين العناصر المُكرَّرة من الجدول.

في المثال التالي، لكـلِّ موظَّف في جـدول المـوظَّفين Employees، يُعـاد سـجل يحـوي الاسـم الأول للموظَّف، والاسـم الأول لمـديره. ولمَّا كـان المـدراء هم أيضًـا مـوظفين، فسـندمج الجـدول مع نفسه:

```
SELECT
    e.FName AS "Employee",
    m.FName AS "Manager"
FROM
    Employees e
JOIN
    Employees m
   ON e.ManagerId = m.Id
```

سيعيد هذا الاستعلام البيانات التالية:

Manager	Employee
James	John
James	Michael
John	Johnathon

لنشرح الاستعلام؛ يحتوي الجدول الأصلي على هذه السجلات:

Id	FName	LName	PhoneNumber	ManagerId	DepartmentId	Salary	HireDate
1	James	Smith	1234567890	NULL	1	1000	01-01-2002
2	John	Johnson	2468101214	1	1	400	23-03-2005
3	Michael	Williams	1357911131	1	2	600	12-05-2009
4	Johnathon	Smith	1212121212	2	1	500	24-07-2016

الخطوة الأولى في تنفيذ الاستعلام هي إجراء جداء ديكارتي لجميع السجلات في الجداول المستخدمة في عبارة `FROM`. في حالتنا هذه، استخدمنا جدول الموظفين مرَّتين، لذا سيبدو الجدول الوسيط كما يلي (أزلنا الحقول غير المستخدمة في المثال):

e.Id	e.FName	e.ManagerId	m.Id	m.FName	m.ManagerId
1	James	NULL	1	James	NULL
1	James	NULL	2	John	1
1	James	NULL	3	Michael	1
1	James	NULL	4	Johnathon	2
2	John	1	1	James	NULL

m.ManagerId	m.FName	m.Id	e.ManagerId	e.FName	e.Id
1	John	2	1	John	2
1	Michael	3	1	John	2
2	Johnathon	4	1	John	2
NULL	James	1	1	Michael	3
1	John	2	1	Michael	3
1	Michael	3	1	Michael	3
2	Johnathon	4	1	Michael	3
NULL	James	1	2	Johnathon	4
1	John	2	2	Johnathon	4
1	Michael	3	2	Johnathon	4
2	Johnathon	4	2	Johnathon	4

الخطوة التالية هي ترشيح السجلات، والإبقاء على السجلات التي تفي بشرط **الدمج** وحسب، أي سجلات الجدول `e` التي يساوي الحقل `ManagerId` خاصتها الحقلَ `Id` في الجدول `m`:

m.ManagerId	m.FName	m.Id	e.ManagerId	e.FName	e.Id
NULL	James	1	1	John	2
NULL	James	1	1	Michael	3
1	John	2	2	Johnathon	4

بعد ذلك، تُقيَّم كل التعبيرات المستخدمة في عبارة SELECT لإعادة الجدول التالي:

m.FName	e.FName
James	John
James	Michael
John	Johnathon

أخيرًا، يُستبدَل اسما العمودين e.FName و m.FName بكُنيتيهما:

Manager	Employee
James	John
James	Michael

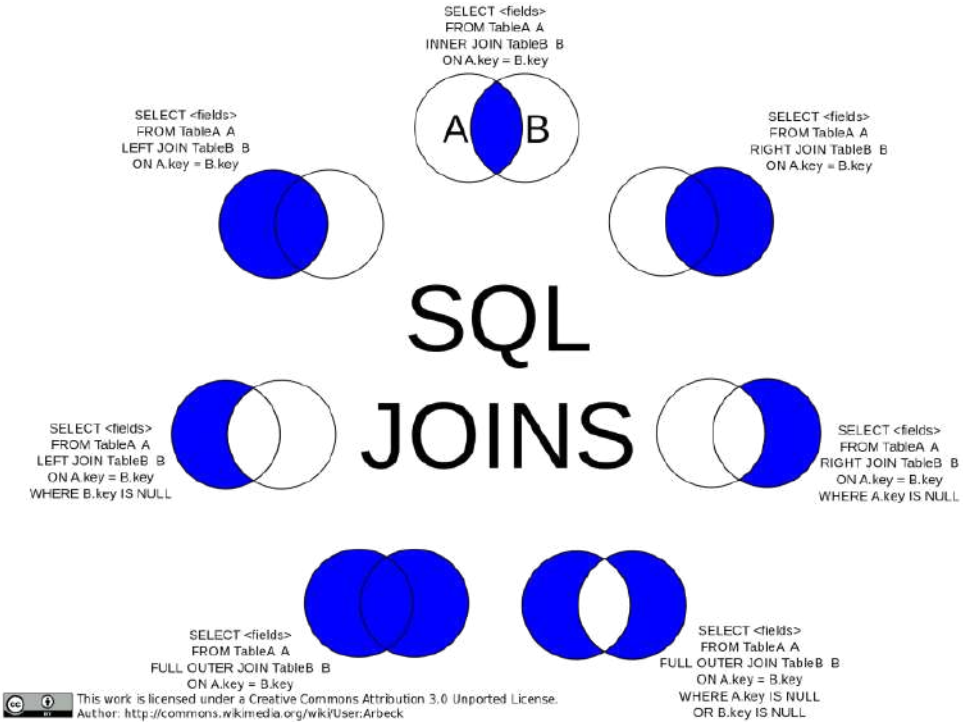
ب. الاختلاف بين الدمج الداخلي والخارجي

هناك عدَّة أنواع من الدمج في SQL، وتختلف تلك الأنواع عن بعضها من حيث ما إذا كانت الصفوف التي (لا) تُحقِّق الشرط ستُدمج أم لا، وهذه أهمها:

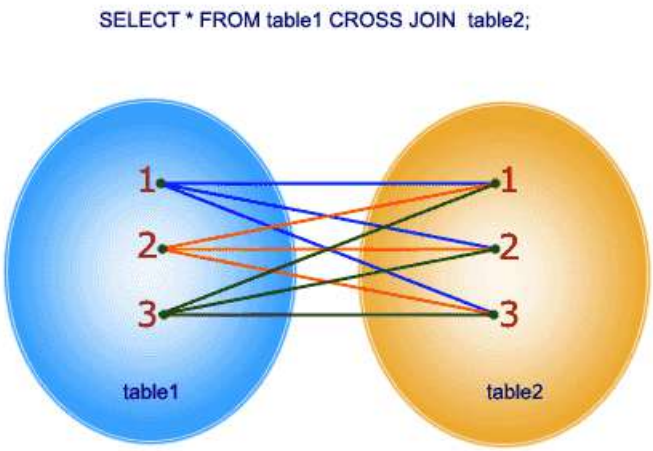
- INNER JOIN: الدمج الداخلي
- LEFT OUTER JOIN: الدمج الخارجي اليساري
- RIGHT OUTER JOIN: الدمج الداخلي اليميني
- FULL OUTER JOIN: الدمج الخارجي التام

(الكلمتان المفتَاحيتان INNER و OUTER اختياريتان.)

يوضِّح الشكل أدناه الاختلافات بين مختلف أنواع الدمج، إذ تمثل المنطقة الزرقاء النتائج المُعادة من عملية الدمج، فيما تُمثِّل المنطقة البيضاء النتائج التي لن تعيدها عملية الدمج.



وهذه صورة لتمثيل الدمج المتقاطع (Cross Join):



مصدر الصورة

على سبيل المثال، إليك الجدولين التاليين:

```
A    B
-    -
1    3
2    4
3    5
4    6
```

لاحظ أنَّ القيمتين (1،2) حصريتان للجدول A، أمَّا القيمتان (3،4) فمُشتركتان، و القيمتان (5،6) حصريتان للجدول B.

- **الدمج الداخلي:** يعيد الدمج الداخلي (inner join) تقاطع الجدولين، أي الصفوف المشترك بينهما:

```
select * from a INNER JOIN b on a.a = b.b;
select a.*,b.* from a,b where a.a = b.b;
a | b
--+--
3 | 3
4 | 4
```

- **الدمج الخارجي اليساري:** يعيد الدمج الخارجي اليساري (left outer join)ـ جميع صفوف a، بالإضافة إلى الصفوف المشتركة مع b:

```
select * from a LEFT OUTER JOIN b on a.a = b.b;
a |  b
--+-----
1 | null
2 | null
3 |    3
4 |    4
```

- **الدمج الخارجي اليميني:** وبالمثل، يعيد الدمج الخارجي اليميني (right outer join) كل صفوف B، بالإضافة إلى الصفوف المشتركة في A:

```
select * from a RIGHT OUTER JOIN b on a.a = b.b;
a    |  b
-----+----
3    |  3
4    |  4
null |  5
null |  6
```

- **الدمج الخارجي التام:** يعيد الدمج الخارجي التام (full outer join) اتحاد a و b، أي جميع الصفوف الموجودة في a وجميع الصفوف الموجودة في b. فإذا كانت هناك بيانات في A بدون بيانات مقابلة في B، فسيكون الجزء الخاص بالجدول B معدومًا (null) والعكس صحيح:

```
select * from a FULL OUTER JOIN b on a.a = b.b;
a    |  b
-----+-----
1 | null
2 | null
3 |    3
4 |    4
null |    6
null |    5
```

ج. اصطلاحات عملية الدمج

دعنا نفترض أنَّ لدينا جدولين A و B، وأنَّ بعض صفوفِهما متطابقة (وفق شرط JOIN):

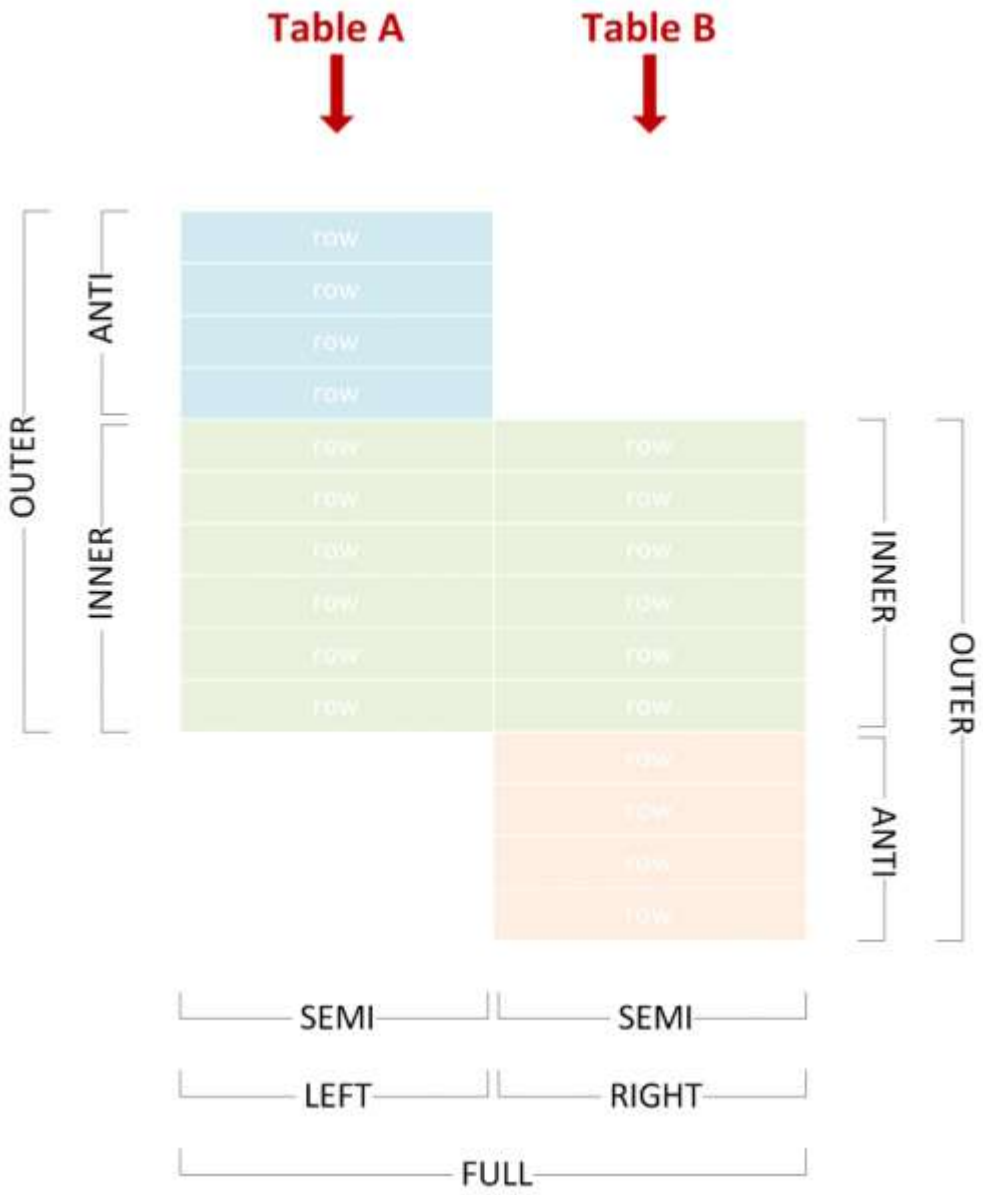


Table A
Table B
row
ANTI
INNER
OUTER
INNER
ANTI
OUTER
SEMI
SEMI
LEFT
RIGHT
FULL

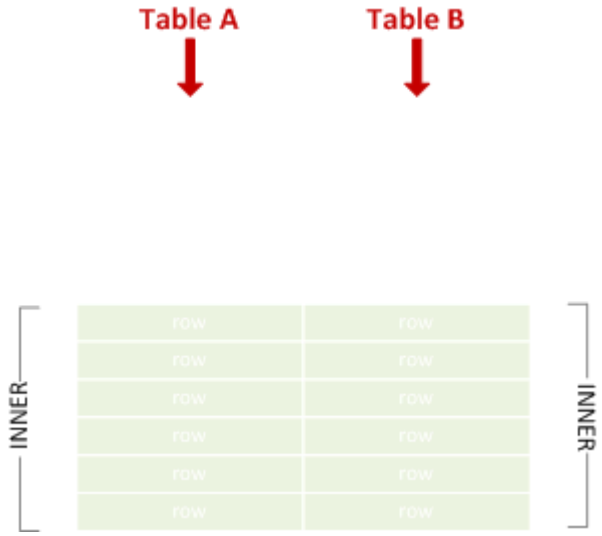
هناك عدَّة أنواع مختلفة من الدمج يمكن استخدامها لأجل تضمين أو استبعاد الصفوف التي (لا) تحقق شرط الدمج في كلا الجانبين.

تستخدم الأمثلة أدناه البيانات التالية:

```
CREATE TABLE A (
    X varchar(255) PRIMARY KEY
);
CREATE TABLE B (
    Y varchar(255) PRIMARY KEY
);
INSERT INTO A VALUES
    ('Amy'),
    ('John'),
    ('Lisa'),
    ('Marco'),
    ('Phil');
INSERT INTO B VALUES
    ('Lisa'),
    ('Marco'),
    ('Phil'),
    ('Tim'),
    ('Vincent');
```

الدمج الداخلي (Inner Join)

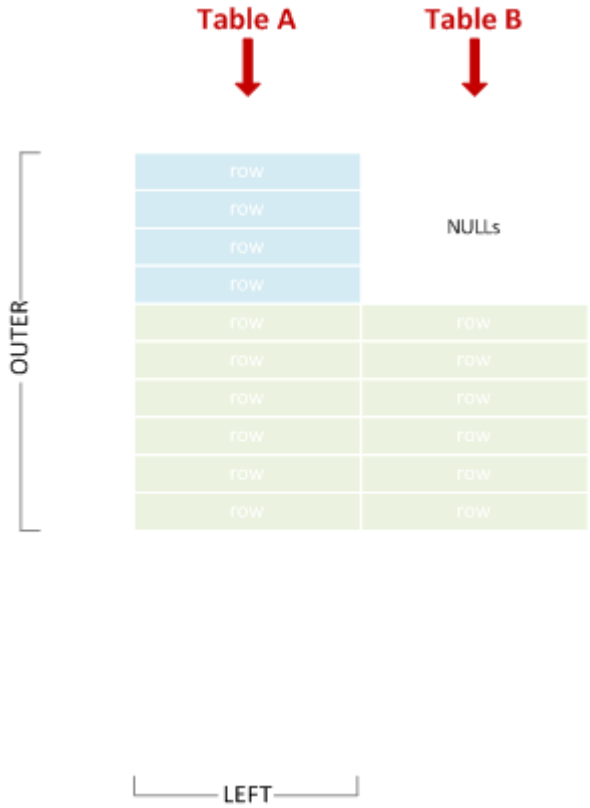
في هذه الحالة، يجمع بين الصفوف اليسرى واليمنى المتطابقة.




```
SELECT * FROM A JOIN B ON X = Y;
X      Y
------ -----
Lisa   Lisa
Marco  Marco
Phil   Phil
```

الدمج الخارجي اليساري (Left outer join)

أمَّا في حالة الدمج الخارجي اليساري (Left outer join)، يُسمَّى اختصارًا الدمجّ اليساري، فيجمع بين الصفوف اليسرى واليمنى التي تحقِّق الشرط، مع تضمين الصفوف اليسرى التي لا تحقِّق الشرط.

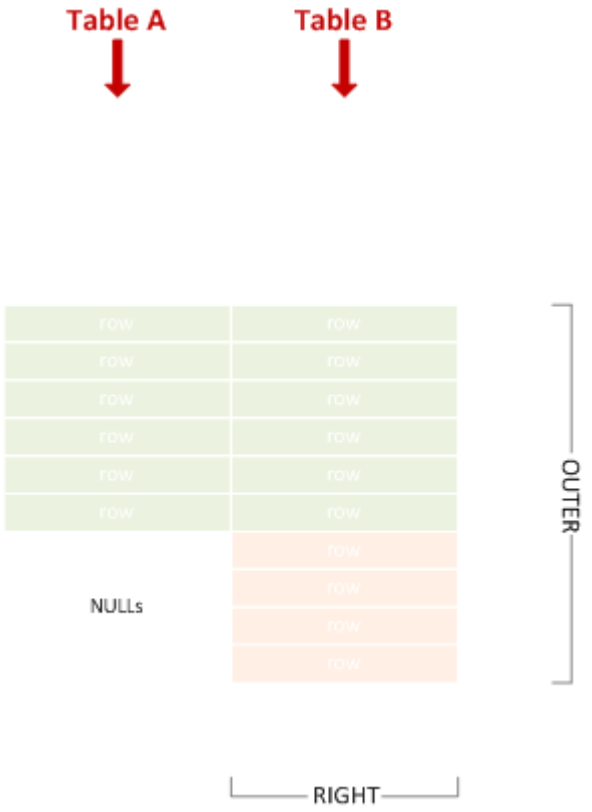



```
SELECT * FROM A LEFT JOIN B ON X = Y;
X          Y
-----      -----
Amy        NULL
John       NULL
Lisa       Lisa
Marco      Marco
Phil       Phil
```

الدمج الخارجي اليميني (Right outer join)

وفي حالة الدمج الخارجي اليميني (Right outer join)، يُسمَّى اختصارًا الدمج الأيمن،

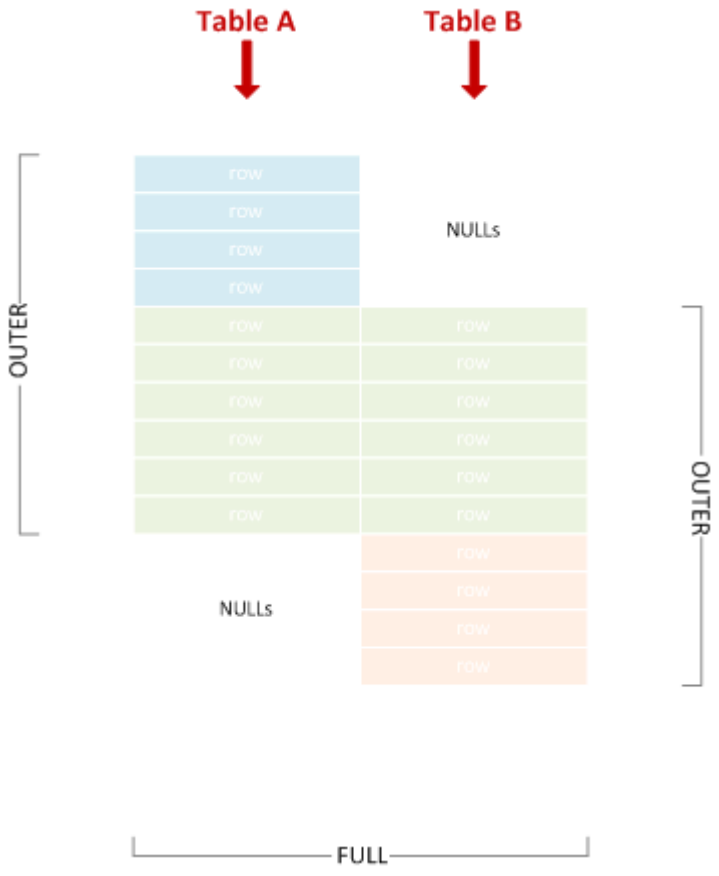
فيجمع بين الصفوف اليسرى واليمنى التي تحقِّق الشرط، مع تضمين الصفوف اليمنى التي لا تحقِّق الشرط.




```
SELECT * FROM A RIGHT JOIN B ON X = Y;
X           Y
-----       -------
Lisa        Lisa
Marco       Marco
Phil        Phil
NULL        Tim
NULL        Vincent
```

الدمج الخارجي التام (Full outer join)

ننتقل إلى الدمج الخارجي التام (Full outer join)، يُسمَّى اختصارًا الدمج التام، فهو اتحاد لعمليتي الدمج اليساري واليميني.




```
SELECT * FROM A FULL JOIN B ON X = Y;
X           Y
-----       -------
Amy         NULL
John        NULL
```

```
Lisa        Lisa
Marco       Marco
Phil        Phil
NULL        Tim
NULL        Vincent
```

الدمج شبه اليساري (Left Semi Join)

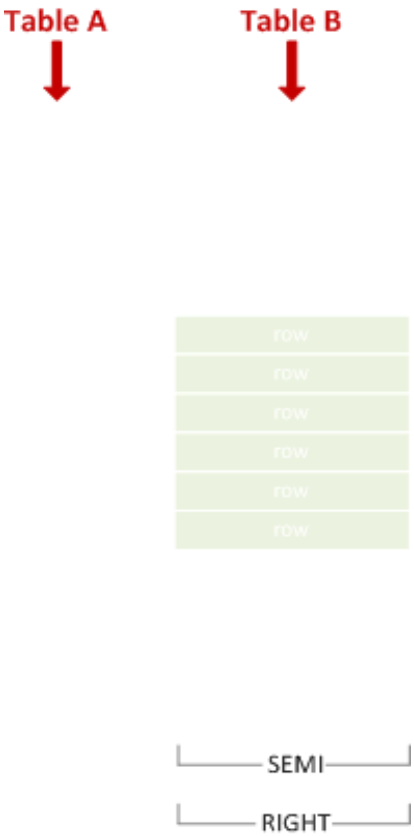
يدمج هذا النوع الصفوفَ اليُسرى التي تتطابق مع الصفوف اليمنى.


```
SELECT * FROM A WHERE X IN (SELECT Y FROM B);
X
-----
Lisa
```

```
Marco
Phil
```

الدمج شبه اليميني (Right Semi Join)

يدمج هذا النوع الصفوف اليمنى التي تطابق الصفوف اليسرى.



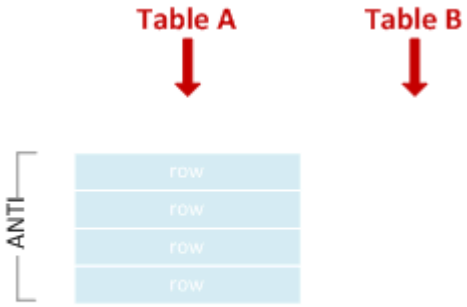

```
SELECT * FROM B WHERE Y IN (SELECT X FROM A);
Y
-----
Lisa
```

```
Marco
Phil
```

لا توجد صياغة للعبارة IN مُخصَّصة للدمج شبه اليساري أو شبه اليميني فكل ما عليك فعله هو تبديل مواضع الجدول في SQL.

الدمج شبه اليساري المعكوس (Left Anti Semi Join)

يُضمِّن هذا النوع الصفوفَ اليُسرى التي **لا** تتطابق مع الصفوف اليمنى.

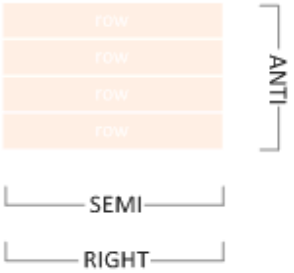


```
SELECT * FROM A WHERE X NOT IN (SELECT Y FROM B);
X
----
Amy
John
```

تنبيه: استخدام NOT IN في الأعمدة التي تقبل القيم المعدومة NULL قد يسبِّب بعض المشاكل (للمزيد من التفاصيل، زر هذه الصفحة).

الدمج شبه اليميني المعكوس (Right Anti Semi Join)

يُضمِّن هذا النوع الصفوف اليمنى التي **لا** تطابق الصفوف اليسرى.



```
SELECT * FROM B WHERE Y NOT IN (SELECT X FROM A);
Y
-------
Tim
Vincent
```

لا توجد صياغة للعبارة IN مُخصَّصة للدمج شبه اليساري أو شبه اليميني المعكوس فكل ما عليك فعله هو تبديل مواضع الجدول في SQL.

الدمج المتقاطع (Cross Join)

يُجرِي هذا النوع من الدمج جداءً ديكارتيًا (Cartesian product) بين الصوف اليسرى والصفوف اليمنى.

```
SELECT * FROM A CROSS JOIN B;
X         Y
-----     -------
Amy       Lisa
John      Lisa
Lisa      Lisa
Marco     Lisa
Phil      Lisa
Amy       Marco
John      Marco
Lisa      Marco
Marco     Marco
Phil      Marco
Amy       Phil
John      Phil
Lisa      Phil
Marco     Phil
Phil      Phil
Amy       Tim
John      Tim
Lisa      Tim
```

```
Marco     Tim
Phil      Tim
Amy       Vincent
John      Vincent
Lisa      Vincent
Marco     Vincent
Phil      Vincent
```

يكافئ الدمج المتقاطع دمجًا داخليًا ذا شرط يتحقَّق دائمًا، لذا سيعيد الاستعلام التالي النتيجة نفسها:

```
SELECT * FROM A JOIN B ON 1 = 1;
```

الدمج الذاتي (Self-Join)

يشير هذا النوع من الدمج إلى دمج الجدول إلى نفسه، ويمكن أن تكون عملية الدمج الذاتي من أيٍّ نوع من أنواع الدمج التي ناقشناها أعلاه. إليك مثالًا حول عملية دمج ذاتي داخلي (inner self-join):

```
SELECT * FROM A A1 JOIN A A2 ON LEN(A1.X) < LEN(A2.X);
X         X
----      -----
Amy       John
Amy       Lisa
Amy       Marco
John      Marco
Lisa      Marco
Phil      Marco
Amy       Phil
```

الدمج الخارجي اليساري (Left Outer Join)

يضمِّن الدمج الخارجي اليساري (المعروف أيضًا باسم الدمج اليساري أو الدمج الخارجي)

جميع صفوف الجدول الأيسر وفي حال عدم وجود صفٍّ مطابق في الجدول الأيمن، فسيُعطَى الحقل المقابل القيمةَ NULL.

سيختار المثال التالي جميع الأقسام (departments) والأسماء الأولى للموظفين الذين يعملون في تلك الأقسام. وستُعَاد الأقسام التي لا تحتوي على موظفين، مع إعطاء اسم الموظف المقابل لها القيمة NULL:

```
SELECT                    Departments.Name, Employees.FName
FROM                       Departments
LEFT OUTER JOIN  Employees
ON                              Departments.Id =
Employees.DepartmentId
```

سنحصل على الخرج التالي:

Employees.FName	Departments.Name
James	HR
John	HR
Johnathon	HR
Michael	Sales
NULL	Tech

سنشرح الاستعلام؛ يوجد جدولان في عبارة FROM، وهما:

HireDate	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
01-01-2002	1000	1	NULL	1234567890	Smith	James	1
23-03-	400	1	1	246810	Johnson	John	2

HireDate	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
2005				1214			
12-05-2009	600	2	1	1357911131	Williams	Michael	3
24-07-2016	500	1	2	1212121212	Smith	Johnathon	4

وهذا هو الجدول الثاني:

Name	Id
HR	1
Sales	2
Tech	3

في المرحلة الأولى، يُنشَأ جداء ديكارتي للجدولين، وينتج عنه جدول وسيط. تحدَّد السجلات التي تفي بشرط الدمج (والذي هو في هذه الحالة: `Departments.Id = Employees.DepartmentId`)؛ وتُمرَّر إلى المرحلة التالية من الاستعلام.

لمَّا كان هذا الدمج دمجًا خارجيًا يساريًا (`LEFT OUTER JOIN`)، فستُعَاد جميع السجلَّات الموجودة في الجانب الأيسر من عملية الدمج (أي الأقسام Departments)، في حين تُعطى السجلات الموجودة على الجانب الأيمن القيمة المعدومة (`NULL`) في حال لم تُطابق شرط الدمج.

HireDate	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
01-01-	**1000**	**1**	**NULL**	**123456**	**Smitd**	**James**	**1**

HireDate	Salary	DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
2002				**7890**			
23-03-2005	**400**	**1**	**1**	**2468101214**	**Johnson**	**John**	**2**
12-05-2009	600	2	1	1357911131	Williams	Michael	3
24-07-2016	**500**	**1**	**2**	**1212121212**	**Smitd**	**Johnatdon**	**4**
01-01-2002	1000	1	NULL	1234567890	Smith	James	1
23-03-2005	400	1	1	2468101214	Johnson	John	2
12-05-2009	**600**	**2**	**1**	**1357911131**	**Williams**	**Michael**	**3**
24-07-2016	500	1	2	1212121212	Smith	Johnathon	4
01-01-2002	1000	1	NULL	1234567890	Smith	James	1

بعد ذلك، تُقيِّم كل التعبيرات المستخدَمة في عبارة SELECT لإعادة الجدول التالي:

Employees.FName	Departments.Name
James	HR

Employees.FName	Departments.Name
John	HR
Richard	Sales
NULL	Tech

2. الدمج الضمني (Implicit Join)

يمكن أيضًا إجراء عملية الدمج على عدَّة جداول، حيث توضع في عبارة `from` مفصولة بالفاصلة الأجنبية `,` مع تحديد العلاقة بينها في العبارة `where`. تسمى هذه التقنية «الدمج الضمني» (Implicit Join)، أو الدمج الغير مباشر لأنَّها لا تحتوي فعليًا العبارةَ `join`).

تدعم جميع أنظمة إدارة قواعد البيانات (RDBMSs) هذه التقنية، ولكن ينصح بتجنُّب استخدامها للأسباب التالية:

- قد تتداخل صياغة الدمج الضمني مع صياغة الدمج المتقاطع (cross join)، وهو ما قد يؤدي إلى إعادة نتائج غير صحيحة، خاصةً إذا كان الاستعلام يحتوي الكثير من عمليات الدمج.
- إذا كنت تنوي استخدام الدمج المتقاطع، فلن يكون ذلك واضحًا من الصياغة (اكتب `CROSS JOIN` بدلًا من ذلك)، ومن المحتمل أن يعدِّلها شخص ما أثناء صيانة الشيفرة دون أن ينتبه.

سيختار المثال التالي أسماء الموظفين الأولى وكذلك أسماء الأقسام التي يعملون فيها:

```
SELECT e.FName, d.Name
FROM   Employee e, Departments d
WHERE  e.DeptartmentId = d.Id
```

سنحصل على الخرج التالي:

d.Name	e.FName
HR	James
HR	John
Sales	Richard

ا. الدمج المتقاطع (CROSS JOIN)

يُجـرِي الـدمج المتقـاطع جـداءً ديكارتيًـا (Cartesian product) على جـدولين (الجـداء الديكارتي هو عملية تُجمِّع كل صف من الجدول الأول مع كل صف من الجدول الثاني).

على سبيل المثال، إذا كان كل من الجدولين TABLEA و TABLEB يحويـان 20 صـفًا، فسـتتألَّف النتيجة المُعَادة من 20*20 = 400 صفًا.

إليك المثال التالي:

```
SELECT d.Name, e.FName
FROM   Departments d
CROSS JOIN Employees e;
```

سنحصل على الخرج التالي:

e.FName	d.Name
James	HR
John	HR
Michael	HR
Johnathon	HR
James	Sales
John	Sales
Michael	Sales

e.FName	d.Name
Johnathon	Sales
James	Tech
John	Tech
Michael	Tech
Johnathon	Tech

يوصى بكتابة `CROSS JOIN` بشكل صريح إن أردت إجراء دمج ديكارتي دفعًا للُّبس.

3. التطبيق المتقاطع والدمج الحرفي

هناك نوع خاص من الدمج يُسمَّى «الدمج الحرفي» (LATERAL JOIN، والذي يُعرَف أيضًا باسم التطبيق المتقاطع [CROSS APPLY] أو التطبيق الخارجي [OUTER APPLY] في كلٍّ من SQL Server و Oracle) وقد أضيف حديثًا إلى الإصدار 9.3 وما بعده من PostgreSQL. الفكرة الأساسية التي ينبني عليها هذا النوع من الدمج هي أنَّه سيتم تطبيق دالة (أو استعلام فرعي مُضمَّن [inline subquery]) على كل الصفوف الناتجة عن عملية الدمج؛ هذا يتيح التحكم في عملية الدمج، مثلًا يمكنك الاكتفاء بدمج أوَّل مُدخَل يحقّق شرط الدمج (matching entry) في الجدول الآخر.

يكمن الاختلاف بين الدمج العادي والدمج الحرفي في حقيقة أنَّه يمكنك استخدام عمود سبق أن دمجته في استعلام فرعي (subquery) طبَّقته تقاطعيًا (CROSS APPLY).

إليك صياغة الدمج الحرفي:

- PostgreSQL 9.3 والإصدارات الأحدث:

```
left | right | inner JOIN LATERAL
```

- SQL Server

```
CROSS | OUTER APPLY
```

`LEFT JOIN LATERAL` متكافئتـان، وكـذلك `CROSS APPLY` و `INNER JOIN LATERAL`

و `OUTER APPLY`. إليك المثال التالي (الإصدار 9.3 وما بعده من PostgreSQL):

```
SELECT * FROM T_Contacts
--LEFT JOIN T_MAP_Contacts_Ref_OrganisationalUnit ON
MAP_CTCOU_CT_UID = T_Contacts.CT_UID AND
MAP_CTCOU_SoftDeleteStatus = 1
--WHERE T_MAP_Contacts_Ref_OrganisationalUnit.MAP_CTCOU_UID IS
NULL -- 989
LEFT JOIN LATERAL
(
 SELECT
              --MAP_CTCOU_UID
              MAP_CTCOU_CT_UID
 ,MAP_CTCOU_COU_UID
 ,MAP_CTCOU_DateFrom
 ,MAP_CTCOU_DateTo
 FROM T_MAP_Contacts_Ref_OrganisationalUnit
 WHERE MAP_CTCOU_SoftDeleteStatus = 1
 AND MAP_CTCOU_CT_UID = T_Contacts.CT_UID
 /*
    AND
    (
        (__in_DateFrom <=
T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateTo)
        AND
        (__in_DateTo >=
T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateFrom)
    )
    */
 ORDER BY MAP_CTCOU_DateFrom
 LIMIT 1
) AS FirstOE
```

وهذا مثال يخصُّ SQL-Server:

```
SELECT * FROM T_Contacts
--LEFT JOIN T_MAP_Contacts_Ref_OrganisationalUnit ON
MAP_CTCOU_CT_UID = T_Contacts.CT_UID AND
MAP_CTCOU_SoftDeleteStatus = 1
--WHERE T_MAP_Contacts_Ref_OrganisationalUnit.MAP_CTCOU_UID IS
NULL -- 989
-- CROSS APPLY -- = INNER JOIN
OUTER APPLY -- = LEFT JOIN
(
 SELECT TOP 1
 --MAP_CTCOU_UID
                              MAP_CTCOU_CT_UID
 ,MAP_CTCOU_COU_UID
 ,MAP_CTCOU_DateFrom
 ,MAP_CTCOU_DateTo
 FROM T_MAP_Contacts_Ref_OrganisationalUnit
 WHERE MAP_CTCOU_SoftDeleteStatus = 1
 AND MAP_CTCOU_CT_UID = T_Contacts.CT_UID
 /*
    AND
    (
        (@in_DateFrom <=
T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateTo)
        AND
        (@in_DateTo >=
T_MAP_Contacts_Ref_OrganisationalUnit.MAP_KTKOE_DateFrom)
    )
    */
 ORDER BY MAP_CTCOU_DateFrom
) AS FirstOE
```

بالانتقال إلى `Apply`، فتُستخدَم هذه العبارة لتطبيق دالة على جدول. سننشئ جدولًا `Department` لتخزين المعلومات الخاصَّة بالأقسام. ثم ننشئ جدولًا `Employee` يحتوي معلومات

حول الموظفين. ينتمي كل موظَّف إلى قسم معيَّن، وبالتالي، فإنَّ لجدول الموظفين مرجعًا متكاملًا (referential integrity) مع جدول الأقسام.

يختار الاستعلام الأول البيانات من جدول الأقسام، ثم يستخدم العبارة CROSS APPLY لتقييم جدول الموظفين نسبة إلى سجلات جدول الأقسام. أمَّا الاستعلام التالي، فيدمج جدول الأقسام إلى جدول الموظفين، ثم يعيد جميع السجلات التي تحقُّق شرط الدمج:

```
SELECT *
FROM Department D
CROSS APPLY (
    SELECT *
    FROM Employee E
    WHERE E.DepartmentID = D.DepartmentID
) A
GO
SELECT *
FROM Department D
INNER JOIN Employee E
    ON D.DepartmentID = E.DepartmentID
```

لو نظرت إلى النتائج المُعادة، فستلاحظ أنَّ الاستعلامين يعيدان نفس النتائج؛ قد تسأل إذن: كيف تختلف CROSS APPLY عن JOIN، وهل أداؤها أحسن.

يختار الاستعلام الأول في الشيفرة التالية البيانات من جدول الأقسام، ثم يستخدم OUTER APPLY لتقييم جدول الموظفين نسبة إلى كل سجل من سجلات جدول الأقسام. تُعطَى قيم الصفوف الغير متطابقة مع أحد في جدول الموظفين القيمة المعدومة NULL، كما هو حال الصفَّين 5 و 6.

يستخدم الاستعلام الثاني الدمج الخارجي اليساري LEFT OUTER JOIN بين جدول الأقسام وجدول الموظفين. وكما هو متوقع، يعيد الاستعلام كافَّة الصفوف من جدول الأقسام؛ بما فيها الصفوف التي ليس لها مقابل في جدول الموظفين.

```
SELECT *
FROM Department D
OUTER APPLY (
    SELECT *
    FROM Employee E
    WHERE E.DepartmentID = D.DepartmentID
) A
GO
SELECT *
FROM Department D
LEFT OUTER JOIN Employee E
    ON D.DepartmentID = E.DepartmentID
GO
```

رغم أنَّ الاستعلامين أعلاه يعيدان المعلومات نفسها، فإنَّ خطة التنفيذ تختلف بعض الشيء. بيْد أنَّه لن يكون هناك اختلاف يُذكَر في الأداء.

يكون استخدام المعامل APPLY في بعض الحالات ضروريًا. ففي المثال التالي، سننشئ دالة جدولية (table-valued function)، تقبل الحقل DepartmentID كمعامل، وتعيد جميع الموظفين المنتمين إلى القسم ذي المعرِّف DepartmentID. يختار الاستعلام التالي البيانات من الجدول Department ويستخدم CROSS APPLY مع الدالة التي أنشأناها. يُمرِّر الحقل DepartmentID من كل صف من الجدول الخارجي (outer table، أي الجدول Department)، ثم يطبِّق الدالة على كل صف بشكل يماثل الاستعلامات الفرعية المرتبطة (correlated subquery).

يستخدم الاستعلام الثاني OUTER APPLY بدلًا من CROSS APPLY، وبالتالي، فعلى عكس التطبيق المتقاطع CROSS APPLY الذي لا يعيد إلا البيانات المرتبطة (correlated data)، يعيد التطبيق الخارجي OUTER APPLY البيانات غير المرتبطة أيضًا، مع وضع القيم المعدومة (NULL) في الأعمدة غير الموجودة.

```
CREATE FUNCTION dbo.fn_GetAllEmployeeOfADepartment (@DeptID AS
int)
RETURNS TABLE
AS
    RETURN
    (
    SELECT
       *
    FROM Employee E
    WHERE E.DepartmentID = @DeptID
    )
GO
SELECT
    *
FROM Department D
CROSS APPLY dbo.fn_GetAllEmployeeOfADepartment(D.DepartmentID)
GO
SELECT
    *
FROM Department D
OUTER APPLY dbo.fn_GetAllEmployeeOfADepartment(D.DepartmentID)
GO
```

قد يتبادر إلى ذهنك السؤال التالي: هل يمكننا استخدام عمليَّة دمج بسيطة بدلًا من استخدام الاستعلامات أعلاه؟ الجواب هو "لا"، إذا استبدلت CROSS / OUTER APPLY في الاستعلامات أعلاه بعمليَّتي الدمج INNER JOIN أو LEFT OUTER JOIN، وحدَّدت العبارة ON (مثلًا 1 = 1)، ثم نفَّذت الاستعلام، فسيُطرَح الخطأ:

```
The multi-part identifier "D.DepartmentID" could not be bound.
```

ذلك أنَّه يكون سياق تنفيذ الاستعلام الخارجي في عمليات الدمج مختلفًا عن سياق تنفيذ الدالة (أو الجدول المشتق [derived table])، ولا يمكنك تمرير قيمة أو متغيِّر من الاستعلام الخارجي إلى الدالة كمعامل، لهذا يجب استخدام المعامل APPLY في مثل هذه الاستعلامات.

ا. الدمج التام (FULL JOIN)

هناك نوع آخر من الدمج أقل شهرة من غيره، وهو الدمج التام (FULL JOIN). يعيد الدمج التام الخارجي (FULL OUTER JOIN) جميع صفوف الجدول الأيسر، وكذلك جميع صفوف الجدول الأيمن. ستُدرَج صفوف الجدول الأيسر التي ليس لها مُطابِقات مقابلة في الجدول الأيمن، وكذلك في الحالة المعكوسة.

ملاحظة: لا تدعم MySQL الدمج التام.

إليك المثال التالي:

```
SELECT * FROM Table1
FULL JOIN Table2
    ON 1 = 2
```

وهذا مثال آخر:

```
SELECT
      COALESCE(T_Budget.Year, tYear.Year) AS RPT_BudgetInYear
      ,COALESCE(T_Budget.Value, 0.0) AS RPT_Value
FROM T_Budget
FULL JOIN tfu_RPT_All_CreateYearInterval(@budget_year_from,
@budget_year_to) AS tYear
      ON tYear.Year = T_Budget.Year
```

إن كنت تستخدم عمليات الحذف اللينة (soft-deletes، والتي لا تحذف البيانات بشكل نهائي)، فسيَتعيَّن عليك التحقق من حالة «الحذف المهمل» (soft delete) مرة أخرى في عبارة WHERE (لأنَّ سلوك الدمج التام [FULL JOIN] يتصرف بشكل يشبه الاتحاد UNION)؛ عند إجراء الدمج التام، سيتعيَّن عليك عادةً السماح باستخدام القيمة المعدومة NULL في عبارة WHERE؛ وفي

حال نسيت ذلك، فسيتصرَّف الدمج كما لو كان دمجًا داخليًا (INNER join)، وهو ما لا تريده عند إجراء الدمج التام.

إليك المثال التالي:

```
SELECT
  T_AccountPlan.AP_UID
 ,T_AccountPlan.AP_Code
 ,T_AccountPlan.AP_Lang_EN
 ,T_BudgetPositions.BUP_Budget
 ,T_BudgetPositions.BUP_UID
 ,T_BudgetPositions.BUP_Jahr
FROM T_BudgetPositions
FULL JOIN T_AccountPlan
 ON T_AccountPlan.AP_UID = T_BudgetPositions.BUP_AP_UID
 AND T_AccountPlan.AP_SoftDeleteStatus = 1
WHERE (1=1)
AND (T_BudgetPositions.BUP_SoftDeleteStatus = 1 OR
T_BudgetPositions.BUP_SoftDeleteStatus IS NULL)
AND (T_AccountPlan.AP_SoftDeleteStatus = 1 OR
T_AccountPlan.AP_SoftDeleteStatus IS NULL)
```

4. الدمج العودي (Recursive JOIN)

يُستخدَم الدمج العَودِي عادة للحصول على بيانات من نوع أب-ابن (parent-child data).

في SQL، تُقدِّم عمليات الدمج العودية باستخدام تعبيرات الجدول العادية كما يوضِّح المثال التالي:

```
WITH RECURSIVE MyDescendants AS (
 SELECT Name
 FROM People
 WHERE Name = 'John Doe'
 UNION ALL
 SELECT People.Name
 FROM People
 JOIN MyDescendants ON People.Name = MyDescendants.Parent
```

```
)
SELECT * FROM MyDescendants;
```

5. الدمج الداخلي الصريح

يستعلم «الدمج الأولي» (basic join)، يُسمَّى أيضًا الدمج الداخلي [inner join]) عن البيانات من جدولين، حيث تُحدَّد العلاقة بينهما في عبارة `join`.

يستعلم المثال التالي عن أسماء الموظفين (`FName`) من جدول الموظفين `Employees`، وأسماء الأقسام التي يعملون فيها (`Name`) من جدول الأقسام `Departments`:

```
SELECT Employees.FName, Departments.Name
FROM   Employees
JOIN   Departments
ON Employees.DepartmentId = Departments.Id
```

سنحصل على الخرج التالي:

Departments.Name	Employees.FName
HR	James
HR	John
Sales	Richard

6. الدمج في استعلام فرعي

غالبًا ما يُستخدم الدمج في الاستعلامات الفرعية (Joining on a Subquery) للحصول على بيانات مُجمَّعة (aggregate data) من جدول يحتوي التفاصيل (الجدول الابن) وعرضها جنبًا إلى جنب مع السجلات من الجدول الأصلي (الجدول الأب).

على سبيل المثال، قد ترغب في الحصول على عدد السجلات الفرعية (child records)، أو متوسط قيم عمود معيَّن في السجلات الفرعية، أو الصف ذو القيمة الأكبر أو الأصغر.

يستخدم هذا المثال الكُنى (لتسهيل قراءة الاستعلامات التي تشمل عدَّة جداول)، يعطي المثال فكرة عامَّة عن كيفية صياغة عمليات دمج الاستعلامات الفرعية. إذ يعيد جميع صفوف الجدول الأصلي "Purchase Orders"، مع إعادة الصف الأول وحسب لكل سجل أصلي (parent record) من الجدول الفرعي PurchaseOrderLineItems.

```
SELECT po.Id, po.PODate, po.VendorName, po.Status, item.ItemNo,
      item.Description, item.Cost, item.Price
FROM PurchaseOrders po
LEFT JOIN
 (
 SELECT l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost,
l.Price, Min(l.id) as Id
 FROM PurchaseOrderLineItems l
 GROUP BY l.PurchaseOrderId, l.ItemNo, l.Description, l.Cost,
l.Price
 ) AS item ON item.PurchaseOrderId = po.Id
```

7. الاتحاد عبر UNION

تُستخدَم الكلمة المفتاحية UNION في SQL لدمج نتائج عبارتي SELECT دون تكرار. أي أنَّها تشبه عملية الاتحاد المعروفة في علم المجموعات، قسم الرياضيات.

من أجل استخدام UNION لدمج النتائج، يُشترَط أن يكون لكلا عبارتي SELECT عدد الأعمدة ونوع البيانات نفسه، ووفق الترتيب نفسه، ولكن يجوز أن تختلف أطوال الأعمدة.

إليك الجدولين التاليين:

```
CREATE TABLE HR_EMPLOYEES
(
    PersonID int,
    LastName VARCHAR(30),
    FirstName VARCHAR(30),
    Position VARCHAR(30)
```

```
);
CREATE TABLE FINANCE_EMPLOYEES
(
    PersonID INT,
    LastName VARCHAR(30),
    FirstName VARCHAR(30),
    Position VARCHAR(30)
);
```

باستخدام UNION، يمكننا الحصول على جميع المدراء (manager) العاملين في القسمين HR و Finance على النحو التالي:

```
SELECT
    FirstName, LastName
FROM
    HR_EMPLOYEES
WHERE
    Position = 'manager'
UNION ALL
SELECT
    FirstName, LastName
FROM
    FINANCE_EMPLOYEES
WHERE
    Position = 'manager'
```

تزيل العبارة UNION الصفوف المُكرَّرة من نتائج الاستعلام. لكن لمَّا كان من الممكن أن يشترك عدَّة أشخاص في نفس الاسم، ويحتلون نفس الموقع في كلا القسمين، فسنستخدم عبارة UNION ALL التي لا تزيل التكرارات.

يمكنك تكنِية (aliasing) الأعمدة المُعادة عبر وضع الكُنى في أول عبارات select على النحو التالي:

```
SELECT
    FirstName as 'First Name', LastName as 'Last Name'
FROM
    HR_EMPLOYEES
WHERE
    Position = 'manager'
UNION ALL
SELECT
    FirstName, LastName
FROM
    FINANCE_EMPLOYEES
WHERE
    Position = 'manager'
```

هناك فرق أساسي بين UNION و UNION ALL:

- تدمج UNION مجموعتي النتائج مع إزالة التكرارات من مجموعة النتائج
- تدمج UNION ALL مجموعتي النتائج دون إزالة التكرارات

من الأخطاء الشائعة استخدامُ UNION في المواضع التي لا تكون فيها حاجة إلى إزالة التكرار من النتائج، فالكلفة الإضافية على الأداء قد تكون أكبر من المكاسب الناجمة عن إزالة التكرارات.

متى تستخدم UNION؟

لنفترض أنَّك تريد ترشيح الدول بحسب قيم سمتين (attributes) مختلفتين، وأنَّك أنشأت فهارس منفصلة غير مُجمَّعة (non-clustered indexes) لكل عمود. في هذه الحالة، يمكنك استخدام UNION، التي تتيح لك استخدام كلا الفهرسين مع تَجنُّب التكرارات.

```
SELECT C1, C2, C3 FROM Table1 WHERE C1 = @Param1
UNION
SELECT C1, C2, C3 FROM Table1 WHERE C2 = @Param2
```

لن تُستخدَم إلا الفهارس البسيطة في تنفيذ الاستعلامات، كما يمكنك تقليل عدد الفهارس المنفصلة غير المُجمَّعة (separate non-clustered indexes)، وهذا سيؤدي إلى تحسين الأداء.

متى تستخدم UNION ALL؟

لنفترض أنَّك تريد ترشيح جدول ما بحسب قيمتي سمتين، بيْد أنَّك لا تحتاج إلى ترشيح السجلات المُكرَّرة (إمَّا لأنَّ ذلك لن يضر، أو أنَّك صمَّمت قاعدة البيانات بحيث لن ينتج أيَّ تكرارات عن عملية الاتحاد).

```
SELECT C1 FROM Table1
UNION ALL
SELECT C1 FROM Table2
```

يمكن أن يكون استخدام `UNION ALL` مفيدًا عند إنشاء معارض (Views) تدمج (join) بيانات صُمِّمت لكي تُقسَّم وتُوزَّع عبر عدَّة جداول (ربما لأسباب تتعلق بتحسين الأداء). ولمَّا كانت البيانات مقُسَّمة سلفًا، فإنَّ جعل محرِّك قاعدة البيانات يزيل التكرارات لن يضيف أيَّ قيمة، وسَيبطئ الاستعلام.

8

دوال التعامل مع البيانات والنصوص

يستعرض هذا الفصل عددًا من أنواع الدوال، مثل الدوال التجميعية (aggregate functions) والدوال التحليلية (analytic functions) والدوال العددية.

1. الدوال التجميعية

يستعرض هذا القسم مجموعة من الدوال التجميعية (aggregate functions) المُستخدمَة في SQL، وهي دوال تأخذ مجموعة من القيم، وتعيد قيمة واحدة.

ا. التجميع الشرطي (Conditional aggregation)

إليك جدول المدفوعات التالي:

Amount	Payment_type	Customer
100	Credit	Peter
300	Credit	Peter
1000	Credit	John
500	Debit	John

تحسب الشيفرة التالية المجموع الكلي لرصيد أو دين كل موظف في الجدول:

```
select customer,
     sum(case when payment_type = 'credit' then amount else 0
end) as credit,
     sum(case when payment_type = 'debit' then amount else 0
end) as debit
from payments
group by customer
```

سنحصل على النتيجة التالية:

Debit	Credit	Customer
0	400	Peter
500	1000	John

إليك الآن المثال التالي:

```
select customer,
     sum(case when payment_type = 'credit' then 1 else 0 end)
as credit_transaction_count,
     sum(case when payment_type = 'debit' then 1 else 0 end) as
debit_transaction_count
from payments
group by customer
```

هذا هو الخرج الناتج:

debit_transaction_count	credit_transaction_count	Customer
0	2	Peter
1	1	John

ب. ضم القوائم (List Concatenation)

تُجمِّع عملية ضم القوائم (List Concatenation) عناصر عمود أو تعبيرًا عن طريق جمع القيم في سلسلة نصية واحدة لكل مجموعة. يمكن أيضًا تحديد سلسلة نصية لفصل القيم (إما سلسلة نصية فارغة أو فاصلة)، كما يمكن تحديد ترتيب القيم المُعَادة. ورغم أنَّها ليست جزءًا من معيار SQL القياسي، إلا أنَّ كل أنظمة قواعد البيانات العلائقية تدعمها.

- MySQL

```
SELECT ColumnA
   , GROUP_CONCAT(ColumnB ORDER BY ColumnB SEPARATOR ',') AS
```

```
ColumnBs
 FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

- Oracle و DB2

```
SELECT ColumnA
   , LISTAGG(ColumnB, ',') WITHIN GROUP (ORDER BY ColumnB) AS
ColumnBs
 FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

- PostgreSQL

```
SELECT ColumnA
   , STRING_AGG(ColumnB, ',' ORDER BY ColumnB) AS ColumnBs
 FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

- SQL Server قبل 2016

```
 WITH CTE_TableName AS (
   SELECT ColumnA, ColumnB
       FROM TableName)
SELECT t0.ColumnA
 ,   STUFF((
     SELECT ',' + t1.ColumnB
     FROM CTE_TableName t1
     WHERE t1.ColumnA = t0.ColumnA
     ORDER BY t1.ColumnB
         FOR XML PATH('')), 1, 1, '') AS ColumnBs
   FROM CTE_TableName t0
 GROUP BY t0.ColumnA
 ORDER BY ColumnA;
```

- SQL Server 2017 و SQL Azure

```
SELECT ColumnA
 ,   STRING_AGG(ColumnB, ',') WITHIN GROUP (ORDER BY ColumnB)
AS ColumnBs
 FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

- SQLite (بدون ترتيب)

```
SELECT ColumnA
   , GROUP_CONCAT(ColumnB, ',') AS ColumnBs
 FROM TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

يتطلب الترتيب استخدام استعلام فرعي (subquery)، أو تعبيرًا جدوليًا CTE، وهو مجموعة نتائج مُؤقَّتة يمكنك الرجوع إليها داخل عبارات SELECT أو INSERT أو UPDATE أو DELETE الأخرى:

```
 WITH CTE_TableName AS (
    SELECT ColumnA, ColumnB
       FROM TableName
       ORDER BY ColumnA, ColumnB)
SELECT ColumnA
       , GROUP_CONCAT(ColumnB, ',') AS ColumnBs
    FROM CTE_TableName
 GROUP BY ColumnA
 ORDER BY ColumnA;
```

ج. حساب المجموع عبر SUM

تجمع الدالة `Sum()` قيم صفوف مجموعة النتائج مع استخدام `group by` أو قيم كل الصفوف بدون `group by`.

المثال التالي لا يستخدم العبارة group by:

```
select sum(salary) TotalSalary
from employees;
```

سنحصل على الخرج التالي:

TotalSalary
2500

إليك مثال آخر يستخدم `group by`:

```
select DepartmentId, sum(salary) TotalSalary
from employees
group by DepartmentId;
```

الخرج الناتج:

TotalSalary	DepartmentId
2000	1
500	2

د. حساب المتوسط عبر AVG

تعيد الدالة التجميعية `AVG()` متوسط قيم تعبير مُعيَّن، والتي عادةً ما تكون قيمًا رقميةفي عمود. لنفترض أنَّ لدينا جدولًا يحتوي على تعداد سكان مدن العالم. مثلا، سجل مدينة نيويورك سيكون من هذا القبيل:

year	population	city_name
2015	8,550,405	New York City
...	...	New York City

New York City	8,000,906	2005

يحسب الاستعلام التالي متوسط عدد سكان مدينة نيويورك في الولايات المتحدة الأمريكية في السنوات العشر الماضية:

```
select city_name, AVG(population) avg_population
from city_population
where city_name = 'NEW YORK CITY';
```

لاحظ كيف لم توضع السنة في الاستعلام، وذلك لأنّنا نريد حساب متوسط عدد السكان بمرور الوقت.

سنحصل على النتائج التالية:

city_name	avg_population
New York City	8,250,754

تنبيه: تُحوِّل الدالة AVG القيم إلى أعداد، وهذا أمر ينبغي أن تأخذه بالحسبان دائمًا، خصوصا عندما تستخدمها مع قيم التاريخ والوقت.

ه. إحصاء الصفوف عبر Count

يمكنك استخدام الدالة `Count()` لحساب عدد الصفوف:

```
SELECT count(*) TotalRows
FROM employees;
```

النتيجة:

TotalRows
4

يحصي المثال التالي الموظفين في كل قسم:

```
SELECT DepartmentId, count(*) NumEmployees
FROM employees
GROUP BY DepartmentId;
```

الخرج الناتج:

NumEmployees	DepartmentId
3	1
1	2

يمكنك إحصاء الصفوف أو المدخلات بحسب الأعمدة أو التعابير مع عدم احتساب القيم المعدومة NULL:

```
SELECT count(ManagerId) mgr
FROM EMPLOYEES;
```

النتيجة:

mgr
3

(هناك قيمة واحدة فقط معدومة في العمود managerID.)

يمكنك أيضًا استخدام DISTINCT داخل دالة أخرى (مثل COUNT) لتَجنُّب إعادة العناصر المُكرَّرة على النحو التالي:

```
SELECT COUNT(ContinentCode) AllCount
 ,         COUNT(DISTINCT ContinentCode) SingleCount
FROM Countries;
```

ستعيد الشيفرة أعلاه قيمًا مختلفة، إذ لن تحسب `SingleCount` إلا عدد القَارَّات الفريـدة (أي غير المكررة)، وذلك على خلاف `AllCount` التي ستحسب التكرارات أيضًا.

إذا طبَّقنا الشيفرة أعلاه على جدول القارات التالي:

ContinentCode
OC
EU
AS
NA
NA
AF
AF

فسنحصل على الخرج التالي:

```
AllCount: 7 SingleCount: 5
```

و. إيجاد أدنى قيمة عبر Min

تبحث الدالة `Min()` عن أصغر قيمة في العمود:

```
select min(age) from employee;
```

سيعيد المثال أعلاه أصغر قيمة في العمود `age` من جدول `employee`.

ز. إيجاد أكبر قيمة عبر Max

تبحث الدالة `Max()` عن القيمة القصوى في العمود:

```
select max(age) from employee;
```

سيعيد المثال أعلاه أكبر قيمة في العمود `age` من جدول `employee`.

2. التعامل مع الأنواع الرقمية

توفِّر SQL العديد من الدوال الأنواع الأولية أو الرقمية (scalar functions، تكون أغلب هذه الأنواع أنواعًا رقمية يمكن قياسها) المُضمَّنة التي تتيح التعامل مع تلك الأنواع، إذ تأخذ قيمة واحدة كمُدخل، وتعيد قيمة واحدة لكل صف في مجموعة النتائج، ويمكنك استخدام هذه الدوال في أي موضع يكون استعمال التعابير جائز فيه داخل عبارات T-SQL .

ا. التاريخ والوقت

في SQL، يُستخدَم النوعان `date` و `time` لتخزين المعلومات المتعلقة بالوقت. يتضمَّن هذان النوعان الوقت time والتاريخ date والتوقيت المختصر smalldatetime والتوقيت الكامل datetime والتوقيت الكامل datetime2 - مبني على 24 ساعة - والتوقيت الإزاحي datetimeoffset أي فارق التوقيت مع التوقيت العالمي الموحد UTC.

لكل واحد من هذه الأنواع تنسيق خاص كما يوضِّح الجدول التالي:

نوع البيانات	التنسيق
time	hh:mm:ss[.nnnnnnn]
date	YYYY-MM-DD
smalldatetime	YYYY-MM-DD hh:mm:ss
datetime	YYYY-MM-DD hh:mm:ss[.nnn]

نوع البيانات	التنسيق
datetime2	YYYY-MM-DD hh:mm:ss[.nnnnnnn]
datetimeoffset	YYYY-MM-DD hh:mm:ss[.nnnnnnn] [+/-]hh:mm

تعيد الدالة DATENAME اسمًا أو جزءًا محدَّدًا من قيمة التاريخ.

```
SELECT DATENAME (weekday,'2017-01-14') as Datename
```

الخرج الناتج عن الشيفرة أعلاه:

Datename
Saturday

يمكنك استخدام الدالة GETDATE لتحديد التاريخ والوقت الحاليين للحاسوب الذي يُنفِّذ شيفرة SQL الحالية كما هو مُوضَّح في المثال التالي (لا تشمل هذه الدالة اختلاف المنطقة الزمنية).

```
SELECT GETDATE() as Systemdate
```

الخرج الناتج:

Systemdate
11:11:47.7230728 2017-01-14

تعيد الدالة DATEDIFF الفرق بين تاريخين، ويُحدِّد المعامل الأوَّل المُمرَّر إلى هذه الدالة الجزء الذي تريد استخدامه من التاريخ لحساب الاختلاف. يمكن أن يساوي: year أو month أو week أو

day أو hour أو minute أو second أو millisecond. يُحـدِّد المعامـل الثـاني والثـالث تـاريخ البداية وتاريخ الانتهاء اللذين تريد حساب الفرق الزمني بينها على التوالي.

إليك المثال التالي:

```
SELECT SalesOrderID, DATEDIFF(day, OrderDate, ShipDate)
AS 'Processing time'
FROM Sales.SalesOrderHeader
```

الخرج الناتج:

Processing time	SalesOrderID
7	43659
7	43660
7	43661
7	43662

تتيح لك الدالة DATEADD إضافة مجال زمني إلى جـزء مُحـدَّد من التـاريخ كما يُوضِّـح المثـال التالي:

```
SELECT DATEADD (day, 20, '2017-01-14') AS Added20MoreDays
```

الخرج الناتج:

Added20MoreDays
00:00:00.000 2017-02-03

ب. التعديلات على الحروف (Character modifications)

توفِّر SQL بعض الدوال التي يمكنها معالجة الأحـرفِ، مثلا، يمكن تحويـل الأحـرف إلى أحـرف

كبيرة أو صغيرة، أو تحويل الأرقام إلى أرقام منسَّقة تنسيقًا خاصًا. فتحوِّل الدالة `lower(char)` الأحرف المُمرَّرة إليها إلى أحرف صغيرة:

```
SELECT customer_id, lower(customer_last_name) FROM customer;
```

يعيد الاستعلام أعلاه الاسم الأخير صغيرًا، أي يُحوِّل `SMITH` إلى `smith`.

ج. دوال الضبط والتحويل

الدالة `@@SERVERNAME` هي إحدى الأمثلة عن دوال الضبط (configuration function) في SQL، إذ تُوفِّر هذه الدالة اسم الخادم المحلي الذي يُنفِّذ تعليمات SQL.

```
SELECT @@SERVERNAME AS 'Server'
```

الناتج:

Server
SQL064

في SQL، تحدث معظم عمليات التحويلات بين أنواع البيانات ضمنيًا، ودون أيِّ تدخل من المستخدم. إن أردت تنفيذ عملية تحويل لا يمكن إجراؤها ضمنيًا، فيمكنك استخدام الدالتين `CAST` أو `CONVERT`.

صياغة `CAST` أبسط من صياغة `CONVERT`، بيد أنَّ إمكانياتها محدودة. سنستخدم في المثال التالي كلا الدالتين `CAST` و `CONVERT` لتحويل نوع بيانات الوقت (datetime) إلى النوع `varcha r`. تستخدم الدالة `CAST` دائمًا التنسيق الافتراضي مثلًا تُمثَّل التواريخ والأوقات بالتنسيق YYYY-MM-DD بالمقابل، تستخدم الدالة `CONVERT` تنسيق التاريخ والوقت الذي تُحدِّده أنت.

سنختار في المثال التالي التنسيق 3، والذي يُمثِّل التنسيق dd/mm/yy:

```
USE AdventureWorks2012
```

```
GO
SELECT FirstName + ' ' + LastName + ' was hired on ' +
     CAST(HireDate AS varchar(20)) AS 'Cast',
     FirstName + ' ' + LastName + ' was hired on ' +
     CONVERT(varchar, HireDate, 3) AS 'Convert'
FROM Person.Person AS p
JOIN HumanResources.Employee AS e
ON p.BusinessEntityID = e.BusinessEntityID
GO
```

ستحصل على الخرج التالي:

Convert	Cast
David Hamiltion was hired on 04/02/03	David Hamiltion was hired on 2003-02-04

هناك مثال آخر على دوال التحويل، وهي الدالة PARSE. تُحوِّل هذه الدالة سلسلة نصية إلى نوع بيانات آخر.

في صياغة الدالة، عليك تحديد السلسلة النصية التي ترغب في تحويلها متبوعةً بالكلمة المفتاحية AS، ثمَّ تكتب نوع البيانات المطلوب. اختياريًا، يمكنك أيضًا تحديد الإعداد الثقافي، والذي يُحدِّد تنسيق السلسلة النصية. في حال لم تُحدِّده، فستُستخدَم لغة الجلسة.

إذا تَعذَّر تحويل السلسلة النصية إلى تنسيق عددي أو تاريخ أو وقت، فسيُطرَح خطأ. وسيتعيَّن عليك حينئذٍ استخدام CAST أو CONVERT لإجراء عملية التحويل.

```
SELECT PARSE('Monday, 13 August 2012' AS datetime2 USING 'en-
US') AS 'Date in English'
```

الخرج التالي:

Date in English
00:00:00.0000000 2012-08-13

د. الدوال المنطقية والرياضية

تقدِّم SQL دالتين منطقيتين، وهما CHOOSE و IIF. تعيد الدالة CHOOSE عنصرًا من قائمة من القيم استنادًا إلى فهرسه في القائمة وينبغي أن يكون المعامل الأول، الذي يُمثِّل الفهرس، عددًا صحيحًا بينما تحدِّد المعاملات التالية قيم القائمة.

سنَستخدِم في المثال التالي الدالة CHOOSE لإعادة المُدخَل الثاني في القائمة المعطاة.

```
SELECT CHOOSE(2, 'Human Resources', 'Sales', 'Admin',
'Marketing' ) AS Result;
```

النتيجة:

Result
Sales

تعيد الدالة IIF القيمة true إن تَحقَّق شرطها، أو تعيد القيمة false خلاف ذلك. في صياغة عبارة الشرط، يُحدَّد معامل التعبير الشرطي (boolean_expression) التعبير المنطقي. فيما يُحدِّد المعامل الثاني (true_value) القيمة التي يجب إعادتها إذا لم يتحقَّق الشرط، ويُحدِّد المعامل الثالث (false_value) القيمة التي يجب أن تُعَاد خلاف ذلك.

يستخدم المثال التالي الدالة IIF لإعادة إحدى قيمتين: إذا كانت مبيعات الموظف السنوية تتجاوز 200000، فسيكون ذلك الموظف مؤهَّلًا للحصول على مكافأة أو لن يكون مؤهَّلًا للحصول على مكافأة إن لم يتحقَّق شرط المبيعات.

```
SELECT BusinessEntityID, SalesYTD,
     IIF(SalesYTD > 200000, 'Bonus', 'No Bonus') AS 'Bonus?'
FROM Sales.SalesPerson
GO
```

هذا هو الناتج:

?Bonus	SalesYTD	BusinessEntityID
Bonus	559697.5639	274
Bonus	3763178.1787	275
No Bonus	172524.4512	285

تحوي SQL العديد من الدوال الرياضية التي يمكنك استخدامها لإجراء عمليات حسابية على المُدخلات ثمَّ إعادة نتائج عددية منها الدالة SIGN، والتي تُعيد قيمة تمثِّل إشارة التعبير (expression sign) إذ تشير القيمة 1- إلى تعبير سالب، فيما تشير القيمة 1+ إلى تعبير موجب، أمَّا 0 فيشير إلى الصفر!

في المثال التالي، القيمة المُدخلَة هي عدد سالب، لذا تُعَاد النتيجة 1-.

```
SELECT SIGN(-20) AS 'Sign'
```

الناتج:

Sign
1-

هناك دالة رياضية أخرى، وهي الدالة POWER. والتي تحسب قوة أو أس تعبير برفعه إلى قوة مُحدَّدة. في صياغة هذه الدالة، يُحدِّد المعامل الأول التعبير العددي، فيما يُحدِّد المعامل الثاني الأس أو القوة:

```
SELECT POWER(50, 3) AS Result
```

النتيجة:

Result
125000

3. الدوال التحليلية

تُستخدَم الدوال التحليلية (analytic functions) لحساب قيمة مُعيَّنة بناءً على مجموعة من القيم. على سبيل المثال، يمكنك استخدام الدوال التحليلية لحساب المجاميع الجارية (running totals)، أو النسب المئوية، أو النتيجة الأكبر داخل مجموعة.

أ. LAG و LEAD

توفِّر الدالة LAG البيانات الخاصَّة بالصفوف التي تسبق الصف الحالي في مجموعة النتائج. على سبيل المثال، يمكنك في عبارة SELECT موازنة قيم الصف الحالي مع قيم الصف السابق. يمكنك استخدام تعبير عددي لتحديد القيم التي يجب موازنتها.

يمثِّل معامل الإزاحة (offset) عدد الصفوف السابقة للصف الحالي التي ستُستخدَم في الموازنة. في حال عدم تحديده، فستُستخدَم القيمة الافتراضية 1.

يُحدِّد المعامل الافتراضي default القيمة التي يجب إعادتها عندما يكون التعبير الموجود في الموضع offset معدومًا (NULL). إذا لم تُحدَّد قيمة لهذا المعامل، فستُستخدَم القيمة الافتراضية NULL.

توفِّر الدالة LEAD بيانات عن الصفوف التي تعقُب الصف الحالي في مجموعة الصفوف. على سبيل المثال، في عبارة SELECT، يمكنك موازنة قيم الصف الحالي مع قيم الصف اللاحق. يمكن تحديد القيم التي يجب موازنتها باستخدام تعبير رقمي.

يُمثِّل معامل الإزاحة (offset) عدد الصفوف اللاحقة للصف الحالي التي ستُستخدَم في الموازنة. يُحدِّد المعامل default القيمة التي ينبغي أن تُعَاد عندما يكون التعبير الموجود عند

موضع الإزاحة معدومًا (NULL). إذا لم تُحدِّد هذين المعاملين، فستُستخدَم القيمتان الافتراضيتان لهذين المعاملين، واللتان تساويان 1 و NULL على التوالي.

يستخدم المثال التالي الدالتين LEAD و LAG لموازنة قيم المبيعات الحالية لكل موظف مع قيم الموظفين المذكورين قبله وبعده، مع ترتيب السجلات بناءً على قيمة العمود BusinessEntityID.

```
SELECT BusinessEntityID, SalesYTD,
LEAD(SalesYTD, 1, 0) OVER(ORDER BY BusinessEntityID) AS "Lead
value",
LAG(SalesYTD, 1, 0) OVER(ORDER BY BusinessEntityID) AS "Lag
value"
FROM SalesPerson;
```

الخرج الناتج:

Lag value	Lead value	SalesYTD	BusinessEntityID
0.0000	3763178.1787	559697.5639	274
559697.5639	4251368.5497	3763178.1787	275
3763178.1787	3189418.3662	4251368.5497	276
4251368.5497	1453719.4653	3189418.3662	277
3189418.3662	2315185.6110	1453719.4653	278
1453719.4653	1352577.1325	2315185.6110	279

ب. PERCENTILE_DISC و PERCENTILE_CONT

تسرد الدالة PERCENTILE_DISC قيمة أوَّل مُدخَل يكون التوزيع التراكمي (cumulative distribution) عنده أعلى من المئين (percentile) الذي قدَّمته باستخدام المعامل

numeric_literal. تُجمَّع القيم حسب مجموعة الصفوف (rowset) أو حسب التوزيع (partition) كما هو مُحدَّد في عبارة WITHIN GROUP.

تشبه PERCENTILE_CONT الدالة PERCENTILE_DISC، بيْد أنَّها تُعيد متوسِّط مجموع أول مُدخل يحقِّق الشرط مع المُدخَل التالي.

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
    CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY
SickLeaveHours ASC)
    AS "Cumulative Distribution",
    PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
       OVER(PARTITION BY JobTitle) AS "Percentile Discreet"
FROM Employee;
```

لإيجاد القيمة التي تطابق أو تتجاوز المئين 0.5، عليك تمرير المئين كقيمة عددية صرفة (numeric literal) إلى دالة المئين الكسري PERCENTILE_DISC. وينتج عن تطبيق هذه الدالة على مجموعة النتائج قائمة مؤلفة من قيم الصف التي يكون التوزيع التراكمي عندها أعلى من المئين المُحدَّد.

Percentile Discreet	Cumulative Distribution	SickLeaveHours	JobTitle	BusinessEntity ID
56	0.25	55	Application Specialist	272
56	0.75	56	Application Specialist	268
56	0.75	56	Application Specialist	269
56	1	57	Application Specialist	267

يمكنك أيضًا استخدام دالة المئين المتصل PERCENTILE_CONT (أي Percentile

Continuous)، والتي ينتج عن تطبيقها على مجموعة النتائج متوسط مجموع قيمة النتيجة مع أعلى قيمة موالية تحقّق الشرط.

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
    CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY
SickLeaveHours ASC)
    AS "Cumulative Distribution",
    PERCENTILE_DISC(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
       OVER(PARTITION BY JobTitle) AS "Percentile Discreet",
    PERCENTILE_CONT(0.5) WITHIN GROUP(ORDER BY SickLeaveHours)
       OVER(PARTITION BY JobTitle) AS "Percentile Continuous"
FROM Employee;
```

الخرج الناتج:

Percentile Continuous	Percentile Discreet	Cumulative Distribution	SickLeaveHours	JobTitle	BusinessEntityID
56	56	0.25	55	Application Specialist	272
56	56	0.75	56	Application Specialist	268
56	56	0.75	56	Application Specialist	269
56	56	1	57	Application Specialist	267

ج. FIRST_VALUE

يمكنك استخدام الدالة `FIRST_VALUE` لتحديد القيمة الأولى في مجموعة نتائج مرتَّبة:

```
SELECT StateProvinceID, Name, TaxRate,
     FIRST_VALUE(StateProvinceID)
     OVER(ORDER BY TaxRate ASC) AS FirstValue
```

```
FROM SalesTaxRate;
```

في هذا المثال، تُستخدَم الدالة FIRST_VALUE لإعادة قيمة الحقل ID الخاص بالولاية أو المقاطعة التي لها أدنى معدَّل للضريبة. فيما تُستخدَم العبارة OVER لترتيب معدَّلات الضريبة للحصول على أدنى معدَّل.

إليك جدول الضرائب:

FirstValue	TaxRate	Name	StateProvinceID
74	5.00	Utah State Sales Tax	74
74	6.75	Minnesota State Sales Tax	36
74	7.00	Massachusetts State Sales Tax	30
74	7.00	Canadian GST	1
74	7.00	Canadian GST	57
74	7.00	Canadian GST	63

د. LAST_VALUE

تعيد الدالة LAST_VALUE القيمة الأخيرة في مجموعة نتائج مرتبة.

```
SELECT TerritoryID, StartDate, BusinessentityID,
    LAST_VALUE(BusinessentityID)
    OVER(ORDER BY TerritoryID) AS LastValue
FROM SalesTerritoryHistory;
```

يستخدم المثال أعلاه الدالة LAST_VALUE لإعادة القيمة الأخيرة لكل مجموعة من الصفوف في مجموعة القيم المُرتَّبة.

LastValue	BusinessentityID	StartDate	TerritoryID
283	280	2005-07-01 00.00.00.000	1
283	284	2006-11-01 00.00.00.000	1
283	283	2005-07-01 00.00.00.000	1
275	277	2007-01-01 00.00.00.000	2
275	275	2005-07-01 00.00.00.000	2
277	275	2007-01-01 00.00.00.000	3

٥. PERCENT_RANK و CUME_DIST

تحسب الدالة PERCENT_RANK ترتيب الصف بالنسبة لمجموعة الصفوف، إذ تُحسَب النسبة المئوية نسبةً إلى عدد الصفوف في المجموعة التي تقل قيمتها عن الصف الحالي. تُعطَى للقيمة الأولى في مجموعة النتائج دائمًا النسبة المئوية 0. بالمقابل، فالنسبة المئوية للقيمة العليا - أو الأخيرة - في المجموعة تساوي دائمًا 1.

تحسب الدالة CUME_DIST الموضع النسبي (relative position) لقيمة معيَّنة في مجموعة من القيم من خلال تحديد النسبة المئوية للقيم التي تصغُر أو تساوي تلك القيمة. تُسمَّى هذه العملية التوزيع التراكمي (cumulative distribution).

سنستخدم في هذا المثال عبارة ORDER لتقسيم - أو تصنيف - الصفوف التي أعَادتها العبارة S

ELECT بناءً على المُسمَّيات الوظيفية للموظَّفين، مع ترتيب النتائج في كل مجموعة على أساس عدد ساعات الإجازات المرضية التي استخدمها الموظفون.

```
SELECT BusinessEntityID, JobTitle, SickLeaveHours,
PERCENT_RANK() OVER(PARTITION BY JobTitle ORDER BY
SickLeaveHours DESC)
     AS "Percent Rank",
CUME_DIST() OVER(PARTITION BY JobTitle ORDER BY SickLeaveHours
DESC)
     AS "Cumulative Distribution"
FROM Employee;
```

الخرج الناتج:

Cumulative Distribution	Percent Rank	SickLeaveHours	JobTitle	BusinessEntityID
0.25	0	57	Application Specialist	267
0.75	0.333333333333333	56	Application Specialist	268
0.75	0.333333333333333	56	Application Specialist	269
1	1	55	Application Specialist	272
1	0	48	Assitant to the Cheif Financial Officer	262

تُرتِّب الدالة PERCENT_RANK المُدخلات في كل مجموعة. فمقابل كل مُدخل، تحسب النسبة المئوية للمدخلات الأخرى في المجموعة التي لها قيم أصغر من المُدخل المُمرَّر.

الدالة `CUME_DIST` مشابهة للدالة السابقة، بيْد أنَّها تُعيد النسبة المئوية للقيم التي تَصغُر القيمة الحالية أو تساويها.

4. دوال النافذة (Window Functions)

ا. التحقق من وجود قيم مكررة في عمود

بفرض وجود جدول البيانات التالي:

unique_tag	example	id
unique_tag	example	1
simple	foo	2
simple	bar	42
hello	baz	3
world	quux	51

يعيد المثال التالي كل هذه الصفوف مع راية تُحدِّد ما إذا كان الوسم `tag` مُستخدمًا من قبل صف آخر.

```
SELECT id, name, tag, COUNT(*) OVER (PARTITION BY tag) > 1 AS
flag FROM items
```

سنحصل على الخرج التالي:

flag	tag	name	id
false	unique_tag	example	1
true	simple	foo	2

flag	tag	name	id
true	simple	bar	42
false	hello	baz	3
false	world	quux	51

في حالة لم تكن قاعدة بياناتك تدعم OVER و PARTITION، فيمكنك استخدام الشيفرة التاليـة للحصول على النتيجة نفسها:

```
SELECT id, name, tag, (SELECT COUNT(tag) FROM items B WHERE tag
= A.tag) > 1 AS flag FROM items A
```

ب. البحث عن التكرارات في جزء من الأعمدة

يسـتخدم المثـال التـالي دالـة نافـذة (Window Function، أي دالـة تجـري حسـابات على مجموعة من الصفوف، كما تتيح الوصول إلى بيانات السجلات التي تسبق السجل الحالي أو التي تليه) لعرض جميع الصفوف المُكرَّرة في جزء من الأعمدة:

```
WITH CTE (StudentId, Fname, LName, DOB, RowCnt)
as (
SELECT StudentId, FirstName, LastName, DateOfBirth as DOB,
SUM(1) OVER (Partition By FirstName,
LastName, DateOfBirth) as RowCnt
FROM tblStudent
)
SELECT * from CTE where RowCnt > 1
ORDER BY DOB, LName
```

ج. إيجاد السجلات الخارجة عن التسلسل

إليك الجدول التالي:

STATUS_BY	STATUS_TIME	STATUS	ID
USER_1	2016-09-28-19.47.52.501398	ONE	1
USER_2	2016-09-28-19.47.52.501511	ONE	3
USER_3	2016-09-28-19.47.52.501517	THREE	1
USER_2	2016-09-28-19.47.52.501521	TWO	3
USER_4	2016-09-28-19.47.52.501524	THREE	3

يجب أن تُرتَّب العناصر بحسب قيمة الحقل STATUS، بداية من القيمة "ONE" ثم "TWO" ثم "THREE". لاحظ أنَّ التسلسل في الجدول غير مُرتَّب، إذ أنَّ هناك انتقالًا فوريًا من "ONE" إلى "THREE". عليك إيجاد طريقة للعثور على المستخدمين (STATUS_BY) الخارجين عن الترتيب. تساعد الدالة التحليلية LAG() في حل هذه المشكلة، إذ تعيد لكل صف، قيمة الصف السابق له:

```
SELECT * FROM (
 SELECT
  t.*,
 LAG(status) OVER (PARTITION BY id ORDER BY status_time) AS
prev_status
 FROM test t
) t1 WHERE status = 'THREE' AND prev_status != 'TWO'
```

في حالة لم تكن قاعدة بياناتك تدعم LAG، يمكنك استخدام الشيفرة التالية للحصول على النتيجة نفسها:

```
SELECT A.id, A.status, B.status as prev_status, A.status_time,
B.status_time as prev_status_time
FROM Data A, Data B
WHERE A.id = B.id
AND   B.status_time = (SELECT MAX(status_time) FROM Data where
status_time < A.status_time and id =
A.id)
AND   A.status = 'THREE' AND NOT B.status = 'TWO'
```

د. حساب المجموع الجاري

إليك جدول البيانات التالي:

amount	date
200	2016-03-12
50-	2016-03-11
100	2016-03-14
100	2016-03-15
250-	2016-03-10

بحسب المثال التالي المجموع الجاري (running total) للعمود `amount` في الجدول أعلاه:

```
SELECT date, amount, SUM(amount) OVER (ORDER BY date ASC) AS
running
FROM operations
ORDER BY date ASC
```

الخرج الناتج:

running	amount	date
250-	250-	2016-03-10
300-	50-	2016-03-11
100-	200	2016-03-12
0	100	2016-03-14
100-	100	2016-03-15

ه. إضافة إجمالي الصفوف المختارة لكل صف

يضيف المثال التالي إجمالي الصفوف المختارة لكل صف:

```
SELECT your_columns, COUNT(*) OVER() as Ttl_Rows FROM
your_data_set
```

Ttl_Rows	name	id
5	example	1
5	foo	2
5	bar	3
5	baz	4
5	quux	5

بدلًا من استخدام استعلامين، الأول للحصول على المجموع، والثاني للحصول على الصف، يمكنك استخدام التجميع (aggregate) كدالة نافذة (window function) واستخدام مجموعة

النتائج الكاملة كنافذة (window). يمكن أن يُجنِّبك هذا تعقيدات عمليات الدمج الذاتي (self joins) الإضافية.

و. الحصول على أحدث س صفًّا في عدة مجموعات

إليك البيانات التالية:

Completion_Date	User_ID
2016-07-20	1
2016-07-21	1
2016-07-20	2
2016-07-21	2
2016-07-22	2

إن استخدمت القيمة `n = 1` في المثال التالي، ستحصل على أحدث صف لكل مُعرِّف `user_id`:

```
;with CTE as
(SELECT *,
 ROW_NUMBER() OVER (PARTITION BY User_ID
 ORDER BY Completion_Date DESC) Row_Num
FROM Data)
SELECT * FORM CTE WHERE Row_Num <= n
```

الخرج سيكون:

Row_Num	Completion_Date	User_ID
1	2016-07-21	1

1	2016-07-22	2

5. دوال التعامل مع النصوص

الدوال النصية (String Functions) هي دوال تُنفَّذ على قيم نصية، وتعيد إمَّا قيمًا عددية أو قيمًا نصية. مثلًا، يمكن استخدام الدوال النصية لدمج البيانات، أو استخراج أجزاء من السلاسل النصية، أو موازنة السلاسل النصية أو تحويلها من الأحرف الكبيرة إلى الصغيرة، أو العكس.

ا. دمج السلاسل النصية

في SQL القياسية (ISO / ANSI)، يُرمَز لمعامل ضم السلاسل النصية (string concatenation) بالرمز || وهذه الصياغة مدعومة من قبل كافَّة أنظمة معالجة قواعد البيانات الرئيسية خلا SQL Server:

```
SELECT 'Hello' || 'World' || '!'; -- ==> HelloWorld!
```

تدعم العديد من أنظمة معالجة قواعد البيانات الدالة CONCAT التي تضم السلاسل النصية:

```
SELECT CONCAT('Hello', 'World'); -- ==> 'HelloWorld'
```

تدعم أيضًا بعض قواعد البيانات استخدام CONCAT لضم أكثر من سلسلتين نصيتين (باستثناء Oracle):

```
SELECT CONCAT('Hello', 'World', '!'); -- ==> 'HelloWorld!'
```

في بعض أنظمة معالجة قواعد البيانات، يجب تحويل الأنواع غير النصية قبل ضمها إلى بعضها بعضًا:

```
SELECT CONCAT('Foo', CAST(42 AS VARCHAR(5)), 'Bar'); -- ==>
'Foo42Bar'
```

تجري بعض قواعد البيانات (مثل Oracle) تحويلات ضمنيَّة غير مُفرِّطة (implicit

lossless conversions)، أي لا ينتج عنها أي ضياع للبيانات. على سبيل المثال، يعيد تطبيق الدالة CONCAT على النوعين CLOB و NCLOB قيمة من النوع NCLOB. فيما يعيد تطبيق الدالة CONCAT على عدد، وعلى قيمة من النوع varchar2 قيمةً من النوع varchar2:

```
SELECT CONCAT(CONCAT('Foo', 42), 'Bar') FROM dual; -- ==>
Foo42Bar
```

يمكن لبعض قواعد البيانات استخدام المعامل + غير القياسي (في معظم الأحيان مع الأعداد وحسب):

```
SELECT 'Foo' + CAST(42 AS VARCHAR(5)) + 'Bar';
```

لا تدعم إصدارات SQL Server قبل 2012 الدالة CONCAT، لذا فالمعامل + هو الطريقة الوحيدة لضم السلاسل النصية فيها:

ب. طول سلسلة نصية

SQL Server

تُستخدَم الدالة LEN لحساب طول سلسلة نصية، بيد أنَّها لا تَحسُب المسافات البيضاء الزائدة.

```
SELECT LEN('Hello') -- 5
SELECT LEN('Hello '); -- 5
```

على خلاف LEN، تحسب الدالة DATALENGTH طول سلسلة نصية بما فيها المسافات الزائدة:

```
SELECT DATALENGTH('Hello') -- 5
SELECT DATALENGTH('Hello '); -- 6
```

تجدر الإشارة إلى أنَّ الدالة DATALENGTH تُعيِد طول التمثيل البتِّي (byte representation) في الذاكرة للسلسلة النصية، والذي تتعلق قيمته بمجموعة المحارف (charset) المستخدمة لتخزين السلسلة النصية.

```
DECLARE @str varchar(100) = 'Hello '
SELECT DATALENGTH(@str) --  6

DECLARE @nstr nvarchar(100) = 'Hello '
SELECT DATALENGTH(@nstr) --  12
```

عادة ما تكون قيم varchar عبارة عن سلسلة نصية من النوع ASCII حيث يُخزَّن كل حرف في بايت، وعادة ما تكون قيم nvarchar عبارة عن سلسلة نصية من النوع unicode حيث يُخزَّن كل حرف في بايتين.

Oracle

يستخدم نظام Oracle الدالة `Length` لحساب طول سلسلة نصية:

```
SELECT Length('Bible') FROM dual; -- 5
SELECT Length('righteousness') FROM dual; -- 13
SELECT Length(NULL) FROM dual; -- NULL
```

ج. تقليم المسافات الفارغة

تُستخدَم الدالة `Trim` لإزالة المسافات البيضاء الموجودة في بداية أو نهاية نتائج الاستعلام.

توجد في MSSQL عدَّة دوال للتقليم كما يوضِّح المثال التالي:

```
SELECT LTRIM('  Hello  ') -- ==> 'Hello  '
SELECT RTRIM('  Hello  ') -- ==> '  Hello'
SELECT LTRIM(RTRIM('  Hello  ')) -- ==> 'Hello'
```

هذا المثال يعمل في MySql و Oracle:

```
SELECT TRIM('  Hello  ') -- ==> 'Hello'
```

د. تحويل حالة حروف النصوص

تحوِّل الدالة UPPER حروف سلسلة نصية إلى حروف كبيرة (في حالة اللغات التي تدعم الحروف الكبيرة والصغيرة مثل اللغة الإنجليزية)، أمَّا الدالة LOWER فتفعل العكس:

```
SELECT UPPER('HelloWorld') -- ==> 'HELLOWORLD'
SELECT LOWER('HelloWorld') -- ==> 'helloworld'
```

ه. تقسيم السلاسل النصية

تقسِّم الدالة SPLIT سلسلة نصية بحسب فاصل حرفي محدَّد. لاحظ أنَّ STRING_SPLIT() دالةٌ جدولية (table-valued function).

```
SELECT value FROM STRING_SPLIT('Lorem ipsum dolor sit amet.', '
');
```

سنحصل على النتيجة التالية:

```
value
-----
Lorem
ipsum
dolor
sit
amet.
```

و. البحث والاستبدال

تستبدل الدالة REPLACE بجزءٍ محدَّدٍ داخل سلسلة نصية جزءًا آخر، وتُصَاغ وفق الشكل التالي:

```
REPLACE( S , O , R )
```

- S: السلسلة النصية التي سيُبحَث فيها

- O: السلسلة النصية المراد استبدالها
- R: السلسلة النصية المراد وضعها مكان السلسلة الأصلية

إليك مثال كامل:

```
SELECT REPLACE( 'Peter Steve Tom', 'Steve', 'Billy' ) -- Peter
Billy Tom
```

ز. التعابير النمطية

MySQL ≥ 3.19

تُستخدَم الدالة REGEXP لمطابقة نمط محدَّد أو مخمَّن من النص داخل السلسلة النصية وذلك عبر استعمال التعابير النمطية (regular expression) التي تُعرَّف بسلسلة نصية أخرى.

```
SELECT 'bedded' REGEXP '[a-f]' -- True
SELECT 'beam' REGEXP '[a-f]' -- False
```

ح. اقتطاع سلسلة نصية

تعيد الدالة SUBSTRING جزءًا من سلسلة نصية كما يوَضّح المثال التالي:

```
SELECT SUBSTRING('Hello', 1, 2) -- ==> 'He'
SELECT SUBSTRING('Hello', 3, 3) -- ==> 'llo'
```

غالبًا ما تُستخدَم SUBSTRING مع الدالة LEN() للحصول على آخر n حرف من سلسلة نصية ذات طول غير معروف.

```
DECLARE @str1 VARCHAR(10) = 'Hello', @str2 VARCHAR(10) =
'FooBarBaz';
SELECT SUBSTRING(@str1, LEN(@str1) - 2, 3) -- ==> 'llo'
SELECT SUBSTRING(@str2, LEN(@str2) - 2, 3) -- ==> 'Baz'
```

تنبيه: تبدأ فهارس السلاسل النصية في SQL من القيمة 1.

ط. إضافة سلسلة نصية لأخرى

تحشر الدالة `Stuff` سلسلة نصّية داخل أخرى، إذ تستبدل 0 حرف أو أكثر في موضع مُعيَّن.

هذه صياغة الدالة:

```
STUFF ( character_expression , start , length ,
replaceWith_expression )
```

يحشـر المثـال التوضـيحي التـالي السلسـلة النصـية 'Hello' في الموضـع 4 من السلسـلة 'FooBarBaz' ويضعها مكان `Bar`:

```
SELECT STUFF('FooBarBaz', 4, 3, 'Hello') -- ==> 'FooHelloBaz'
```

تنبيه: يُحسب الموضع `start` انطلاقا من القيمة 1.

ي. الدالتان LEFT و RIGHT

تعيـد الدالـة `RIGHT` آخـر n حـرف من سلسـلة نصـية، فيمـا تعيـد `LEFT` أول n حـرف من سلسلة نصية.

إليك المثال التالي:

```
SELECT LEFT('Hello',2) -- He
SELECT RIGHT('Hello',2) -- lo
```

لا يحتـوي نظـام SQL Oracle على الـدالتين `LEFT` و `RIGHT`. بيـد أنَّه يمكن محاكاتهمـا باستخدام الدالتين `SUBSTR` و `LENGTH` على النحو التالي:

```
SELECT SUBSTR('Hello',1,2) --  He
SELECT SUBSTR('Hello',LENGTH('Hello')-2+1,2) -- lo
```

ك. عكس سلسلة نصية

تعكس الدالة REVERSE السلاسل النصية:

```
SELECT REVERSE('Hello') -- ==> olleH
```

ل. تكرار سلسلة نصية

تـدمج الدالـة REPLICATE سلسـلة نصـية إلى نفسـها عـددًا محـدَّدًا من المـرَّات كمـا يوضِّـح المثال التالي:

```
SELECT REPLICATE ('Hello',4) -- ==> 'HelloHelloHelloHello'
```

م. تحديث محتوى سلسلة نصية

تُستخدَم الدالة REPLACE في SQL لتحديث محتوى سلسلة نصية وتُستدعَى هـذه الدالـة عـبر الصياغة REPLACE() في MySQL و Oracle و SQL Server. وتُصاغ على النحو التالي:

```
REPLACE (str, find, repl)
```

يسـتبدل المثـال التـالي بتكـرارات السلسـلة النصـية South السلسـلة النصـية Southern في جدول الموظفين Employees التالي:

Address	FirstName
South New York	James
South Boston	John
South San Diego	Michael

إليك الاستعلام التالي:

```
SELECT
    FirstName,
    REPLACE (Address, 'South', 'Southern') Address
FROM Employees
ORDER BY FirstName
```

سنحصل على النتيجة التالية:

Address	FirstName
Southern New York	James
Southern Boston	John
Southern San Diego	Michael

يمكننا استخدام الدالة REPLACE لإجراء تغييرات دائمة في الجدول على النحو التالي:

```
Update Employees
Set city = (Address, 'South', 'Southern');
```

هناك مقاربة أخرى أشهر تتمثل في استخدام REPLACE مع عبارة WHERE على النحو التالي:

```
Update Employees
Set Address = (Address, 'South', 'Southern')
Where Address LIKE 'South%';
```

ن. البحث ضمن سلسلة نصية

تعيد هذه الدالة فهرس أول ظهور لسلسلة نصية فرعية (أو تعيد 0 إن لم يُعثر عليها).

```
SELECT INSTR('FooBarBar', 'Bar') -- 4
SELECT INSTR('FooBarBar', 'Xar') -- 0
```

س. إعادة جزء محدَّد من نص أو شبه نص

SQL Server

تعيد الدالة PARSENAME جزءًا محدَّدًا من كائن نصي string(object name). قد يحتوي اسم الكائن object name على اسم كائن شبه نصي (string like object)، أو اسم المالك (owner name) أو اسم قاعدة البيانات أو اسم الخادم. يمكنك معرفة المزيد من التفاصيل من هذه الصفحة: MSDN:PARSENAME.

هذه صياغة الدالة PARSENAME:

```
PARSENAME('NameOfStringToParse',PartIndex)
```

- يمكنك العثور على اسم الكائن، في الفهرس رقم 1:

```
SELECT
PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',1) --
`ObjectName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',1)
-- `Student`
```

- للحصول على اسم المخطط (schema)، استخدم الفهرس 2:

```
SELECT
PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',2) --
`SchemaName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',2)
-- `school`
```

- للحصول على اسم قاعدة البيانات، استخدم الفهرس 3:

```
SELECT
PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',3) --
`DatabaseName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',3)
-- `SchoolDatabase`
```

- للحصول على اسم الخادم، استخدم الفهرس 4:

```
SELECT
PARSENAME('ServerName.DatabaseName.SchemaName.ObjectName',4) --
`ServerName`
SELECT PARSENAME('[1012-1111].SchoolDatabase.school.Student',4)
-- `[1012-1111]`
```

تعيد `PARSENAME` القيمة `null` في حال كان الجزء المُعيَّن غير موجود في الكائن النصي.

9

الاستعلامات الفرعية والإجراءات

يستعرض هذا الفصل بعض المواضيع المتقدمة عن تنفيذ الشيفرات في SQL، مثل الاستعلامات الفرعية، وكتل التنفيذ، والإجراءات المُخزَّنة، والمنبِّهات (triggers)، والعمليات (transactions).

1. الاستعلامات الفرعية

الاستعلامات الفرعية (subqueries) هي استعلامات داخلية أو متشعِّبة داخل استعلام آخر في SQL. يُمكِن إضافة الاستعلامات الفرعية داخل `FROM` أو `SELECT` أو `WHERE`.

ا. في عبارة FROM

تتصرَّف الاستعلامات الفرعية في عبارة `FROM` بشكل مشابه للجداول المؤقتة المُنشَأة أثناء تنفيذ استعلام، والمفقودة إثر ذلك.

```
SELECT Managers.Id, Employees.Salary
FROM (
    SELECT Id
    FROM Employees
    WHERE ManagerId IS NULL
) AS Managers
JOIN Employees ON Managers.Id = Employees.Id
```

يمكنك استخدام الاستعلامات الفرعية لتعريف جدول مؤقَّت واستخدامه في عبارة `FROM` الخاصَّة بالاستعلام الخارجي.

تبحث الشيفرة التالية عن المدن في جدول الطقس `weather` التي تتغيَّر درجات الحرارة اليومية الخاصَّة بها بأكثر من 20 درجة:

```
SELECT * FROM (SELECT city, temp_hi - temp_lo AS temp_var FROM
weather) AS w
WHERE temp_var > 20;
```

النتيجة:

temp_var	city
21	ST LOUIS
31	LOS ANGELES
23	LOS ANGELES
31	LOS ANGELES
27	LOS ANGELES
28	LOS ANGELES
28	LOS ANGELES
32	LOS ANGELES

ب. في عبارة SELECT

إليك مثال على استعلام فرعي في SELECT:

```
SELECT
     Id,
     FName,
     LName,
    (SELECT COUNT(*) FROM Cars WHERE Cars.CustomerId =
Customers.Id) AS NumberOfCars
FROM Customers
```

ج. في عبارة WHERE

يمكنك استخدام استعلام فرعي لترشيح مجموعة النتائج. على سبيل المثال، تعيد الشيفرة التالية الموظفين الأعلى أجرًا فقط:

```
SELECT *
FROM Employees
WHERE Salary = (SELECT MAX(Salary) FROM Employees)
```

يبحث المثال التالي عن المدن (من مثال المدن) الـتي يقـل تعـداد سـكانها عن متوسـط درجـة الحرارة فيها (يتم الحصول عليها عن طريق استعلام فرعي):

```
SELECT name, pop2000 FROM cities
WHERE pop2000 < (SELECT avg(pop2000) FROM cities);
```

في المثـــال أعلاه، يُحـــدِّد الاســـتعلام الفـــرعي `SELECT avg(pop2000) FROM` شـــرط عبارة `WHERE`.

النتيجة:

pop2000	name
776733	San Francisco
348189	ST LOUIS
146866	Kansas City

د. الاستعلامات الفرعية المترابطة

الاســـتعلامات الفرعيـــة المترابطـــة (Correlated Subqueries)، والمعروفـــة أيضًـــا باسم المتزامنـــة [Synchronized] أو المتَّســـقَة [Coordinated]) هي اســـتعلامات متشـــعِّبة تحتـــوي مرجعًا يشير إلى الصف الحالي في الاستعلام الخارجي:

```
SELECT EmployeeId
     FROM Employee AS eOuter
     WHERE Salary > (
       SELECT AVG(Salary) -- (*)
       FROM Employee eInner
```

```
        WHERE eInner.DepartmentId = eOuter.DepartmentId
  )
```

الاستعلام الفرعي في السطر (*) مترابط مع الاستعلام الخارجي لأنَّه يشير إلى الصف Employee من الجدول eOuter من الاستعلام الخارجي.

ه. ترشيح نتائج استعلام باستعلام مُنفَّذ على جدول آخر

يختار الاستعلام التالي جميع الموظفين غير الموجودين في جدول المشرفين Supervisors:

```
SELECT *
FROM Employees
WHERE EmployeeID not in (SELECT EmployeeID
 FROM Supervisors)
```

يمكن تحقيق النتائج نفسها باستخدام الدمج اليساري LEFT JOIN:

```
SELECT *
FROM Employees AS e
LEFT JOIN Supervisors AS s ON s.EmployeeID=e.EmployeeID
WHERE s.EmployeeID is NULL
```

2. كتل التنفيذ

تُستخدم الكلمتان المفتاحيتان BEGIN و END لبدء كتلة تنفيذ (Execution block) وإنهائها على التوالي، كما يوضح المثال التالي:

```
BEGIN
      UPDATE Employees SET PhoneNumber = '5551234567' WHERE Id
= 1;
      UPDATE Employees SET Salary = 650 WHERE Id = 3;
END
```

3. الإجراءات المخزَّنة

يمكن إنشاء إجراءات مخزَّنة (Stored Procedures) عبر واجهة المستخدم الرسومية الخاصة ببرنامج إدارة قاعدة البيانات (مثال SQL Server)، أو من خلال عبارة SQL كما يلي:

```
-- تحديد الاسم والمعاملات
CREATE PROCEDURE Northwind.getEmployee
 @LastName nvarchar(50),
 @FirstName nvarchar(50)
AS
-- تحديد الاستعلام المراد تنفيذه
SELECT FirstName, LastName, Department
FROM Northwind.vEmployeeDepartment
WHERE FirstName = @FirstName AND LastName = @LastName
AND EndDate IS NULL;
```

يمكن استدعاء الإجراء على النحو التالي:

```
EXECUTE Northwind.getEmployee N'Ackerman', N'Pilar';
-- أو
EXEC Northwind.getEmployee @LastName = N'Ackerman', @FirstName
= N'Pilar';
GO
-- أو
EXECUTE Northwind.getEmployee @FirstName = N'Pilar', @LastName
= N'Ackerman';
GO
```

4. المنبهات (Triggers)

المنبهات (Triggers) هي إجراءات مُخزَّنة تُستدعَى تلقائيًا عند وقوع أحداث معينة، مثل، إدراج صف في عمود، أو تحديث عمود ما، أو غيرها من الأحداث.

ا. إنشاء منبّه

ينشئ هذا المثال منبّهًا يُدرج سجلًا في جدول ثانٍ (`MyAudit`) عند إدراج سجل ما في الجدول الذي عُرِّف المنبّه عليه (`MyTable`).

في المثال التالي، الجدول "inserted" هو جدول خاص تستخدمه Microsoft SQL Server لتخزين الصفوف المُتأثِّرة (affected rows) خلال عبارتي `INSERT` و `UPDATE`؛ يوجد أيضًا جدول "deleted" خاص يؤدي نفس الوظيفة في عبارات `DELETE`.

```
CREATE TRIGGER MyTrigger
     ON MyTable
     AFTER INSERT
AS
BEGIN
     -- إضافة سجل إلى الجدول MyAudit
     INSERT INTO MyAudit(MyTableId, User)
     (SELECT MyTableId, CURRENT_USER FROM inserted)
END
```

المثال التالي يستخدم منبّهًا لإدارة سلة المحذوفات عبر الجدول "deleted":

```
CREATE TRIGGER BooksDeleteTrigger
         ON MyBooksDB.Books
         AFTER DELETE
AS
     INSERT INTO BooksRecycleBin
         SELECT *
         FROM deleted;
GO
```

5. العمليات (Transactions)

العمليات (Transactions) هي سلسلة من تعليمات SQL تُجرَى على قاعدة بيانات، وتُعامَل هذه السلسلة كما لو كانت عملية واحدة، إذ إمَّا أن تُنفَّذ جميعًا، ونقول أنَّه تم الالتزام بها

(committed)، أو لا تُنفَّذ أية تعليمة منها، ونقول أنَّه تم التراجع عنها (rolled back).

يوضِّح المثال التالي عملية بسيطة:

```
BEGIN TRANSACTION
    INSERT INTO DeletedEmployees(EmployeeID, DateDeleted, User)
    (SELECT 123, GetDate(), CURRENT_USER);
    DELETE FROM Employees WHERE EmployeeID = 123;
COMMIT TRANSACTION
```

يمكنك التراجع عن العملية في حال حدث خطأ في الشيفرة:

```
BEGIN TRY
    BEGIN TRANSACTION
       INSERT INTO Users(ID, Name, Age)
       VALUES(1, 'Bob', 24)

       DELETE FROM Users WHERE Name = 'Todd'
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    ROLLBACK TRANSACTION
END CATCH
```

10

تخطيط الجداول وترتيب التنفيذ وتنظيم الشيفرة

يستعرض هذا الفصل عددًا من المواضيع المتعلقة بتخطيطات الجداول وكتاب شيفرات SQL، وهي كيفيَّة تصميم جداول قواعد البيانات، وكيفية استخلاص المعلومات المتعلِّقة بقاعدة البيانات عبر معلومات المُخطَّط، والترتيب الذي تُنفَّذ وفقه عبارات واستعلامات SQL ويتطرق أخيرًا إلى أفضل الممارسات المُتعَارف عليها لكتابة شيفرات SQL نظيفة، وكذلك تأمين الشيفرات عبر التحوط من هجمات حقن SQL.

1. تصميم الجداول (Table Design)

لا تنحصر وظائف أنظمة قواعد البيانات العلائقية في تخزين البيانات في الجداول، وكتابة عبارات SQL لجلبها. إن كان تصميم الجداول سيئًا، فقد يؤدي ذلك إلى إبطاء تنفيذ الاستعلامات، ويمكن أن يؤثِّر على عمل قاعدة البيانات، بحيث لا تعمل كما هو متوقع. لذا لا ينبغي النظر إلى جداول قاعدة البيانات كما لو كانت مجرَّد جداول عادية؛ إذ يتوجَّب أن تتَّبع مجموعة من القواعد حتى تكون علائقية حقًّا.

هذه هي القواعد الخمسة التي ينبغي أن تتوفَّر في أي جدول علائقي:

1. أن تكون كل القيم ذرِّية (atomic)، أي يجب أن تكون قيمة كل حقل من كل صف قيمة واحدة.
2. يجب أن تنتمي بيانات كل حقل إلى نفس نوع البيانات.
3. يجب أن يكون لكل حقل اسمًا فريدًا.
4. يجب أن يحتوي كل صف في الجدول على قيمة واحدة على الأقل تجعله متفرِّدًا عن السجلات الأخرى في الجدول.
5. لا ينبغي أن يكون لترتيب الصفوف والأعمدة أيّ تأثير.

إليك مثال على جدول يتوافق مع القواعد الخمس أعلاه:

Manager	DOB	Name	Id
3	11/02/1971	Fred	1
3	11/02/1971	Fred	2
2	08/07/1975	Sue	3

لنتحقق من القواعد السابقة:

- القاعدة 1: كل القيم ذرّية. إذ لا تحوي الحقول `Id` و `Name` و `DOB` و `Manager` إلا قيمًا فردية (single) فقط.
- القاعدة 2: لا يحتوي الحقل `Id` إلا على الأعداد الصحيحة، فيما يحتوي الحقل `Name` حصرًا على القيم النصية (يمكننا إضافة أنَّها جميعًا تتألف من أربعة أحرف أو أقل)، فيما يحتوي الحقل `DOB` على تواريخ من نوع صالح، ويحتوي الحقل `Manager` على أعداد صحيحة (يمكننا إضافة قيد التوافق مع حقل المفاتيح الرئيسية في جدول المدراء managers).
- القاعدة 3: الأسماء `Id` و `Name` و `DOB` و `Manager` هي أسماء عناوين فريدة للحقول
- داخل الجدول.
- القاعدة 4: يميِّز الحقل `Id` كل السجلَّات، ويجعل كل سجل مختلفًا عن السجلات الأخرى داخل الجدول.

هذا مثال على جدول ذي تصميم سيء:

Name	DOB	Name	Id
3	11/02/1971	Fred	1
3	11/02/1971	Fred	1

Name	DOB	Name	Id
1 ,2	Friday the 18th July 1975	Sue	3

لنتحقَّق من القواعد السابقة:

- القاعدة 1: يحتوي الحقل الثاني على قيمتين، 2 و 1.
- القاعدة 2: يحتوي الحقل DOB على نوعي بيانات مختلفين، نوع التاريخ، ونوع النصوص.
- القاعدة 3: هناك حقلان لهما الاسم نفسه ("name").
- القاعدة 4: السجل الأول والثاني متماثلان تمامًا.
- القاعدة 5: هذه القاعدة مُستوفاة.

2. مخطط المعلومات (Information Schema)

مخطَّط المعلومات (Information Schema) هو استعلام يوفِّر معلومات مفيدة للمستخدمين النهائيين عن أنظمة إدارة قواعد البيانات (RDBMS).

تتيح مثل هذه الاستعلامات للمستخدمين إمكانية العثور السريع على جداول قاعدة البيانات التي تحوي أعمدة مُعيَّنة، كما يحدث عندما ترغب في ربط البيانات من جدولين بشكل غير مباشر عبر جدول ثالث دون معرفة مُسبقَة بالجداول التي قد تحتوي على مفاتيح أو أعمدة أخرى مشتركة مع الجداول المستهدفة.

يستخدم المثال التالي تعبير T-SQL، ويبحث عن مُخطَّط المعلومات الخاص بقاعدة البيانات:

```
SELECT *
FROM INFORMATION_SCHEMA.COLUMNS
WHERE COLUMN_NAME LIKE '%Institution%'
```

تحتوي النتيجة على قائمة بالأعمدة المُطابقَة، وأسماء جداولها، ومعلومات أخرى مفيدة.

3. ترتيب التنفيذ

تُنفَّذ عبارات استعلامات SQL وفق ترتيب مُحدَّد. تستعرض هذه الفقرة هذا الترتيب:

```
/*(8)*/ SELECT /*9*/ DISTINCT /*11*/ TOP
/*(1)*/ FROM
/*(3)*/ JOIN
/*(2)*/ ON
/*(4)*/ WHERE
/*(5)*/ GROUP BY
/*(6)*/ WITH {CUBE | ROLLUP}
/*(7)*/ HAVING
/*(10)*/ ORDER BY
/*(11)*/ LIMIT
```

إليك الترتيب الذي تتم به معالجة الاستعلامات، مع وصف مختصر لكل منها (تشير VT إلى "Virtual Table" أي جدول وهمي، وتُوضِّح كيف يتم إنتاج مختلف البيانات أثناء معالجة الاستعلام):

1. FROM: تُنفَّذ جداء ديكارتي (دمجًا متقاطعًا [cross join]) بين الجدولين الأولين في عبارة FROM، ونتيجة لذلك، يُنشَأ جدول وهمي VT1.
2. ON: ترشِّح الجدول الوهمي VT1 ولا تُدرِج إلا الصفوف التي تعيد TRUE إلى الجدول الوهمي VT2.
3. OUTER: في حال الدمج الخارجي OUTER JOIN (على عكس الدمج المتقاطع CROSS JOIN أو الدمج الداخلي INNER JOIN)، تُضَاف صفوف الجدول أو الجداول المحفوظة (preserved table) التي لم يُعثَر فيها على تطابق إلى صفوف الجدول الوهمي VT2 كصفوف خارجية، ويَنتُج عن ذلك الجدول VT3. في حال كان هناك أكثر من جدولين في عبارة FROM، تُطبَّق الخطوات من 1 إلى 3 بشكل متكرر بين نتيجة عملية الدمج الأخيرة والجدول التالي في عبارة FROM إلى أن تُعالَج جميع الجداول.

4. WHERE: ترشِّح الجدول VT3. ولا تُدرِج إلا الصفوف التي تعيد TRUE إلى الجدول VT4
5. GROUP BY: تُقسِّم صفوف الجدول الوهمي VT4 إلى مجموعات بناءً على قائمة الأعمدة المُحدَّدة في عبارة GROUP BY وينجم عن ذلك إنشاء جدول VT5.
6. CUBE | ROLLUP: تُضَاف مجموعات أجزاء (Supergroups، مجموعات مُؤلَّفة من مجموعات) إلى صفوف VT5، ويَنتُج الجدول الوهمي VT6.
7. HAVING: ترشِّح الجدول VT6. ولا تُدرِج إلا المجموعات التي تعيد القيمة TRUE إلى الجدول VT7.
8. SELECT: تُعالَج قائمة SELECT، ويُنشأ الجدول VT8.
9. DISTINCT: تُزال الصفوف المكرّرة من VT8. ويُنشأ الجدول VT9.
10. ORDER BY: تُرتَّب صفوف الجدول VT9 وفقًا لقائمة الأعمدة المحدّدة في عبارة ORDER BY، كما يُنشأ مُؤشّر - cursor - (VC10).
11. TOP: يُختار العدد أو النّسبة المئوية المحدّدة من الصفوف من بداية الجدول VC10. ويُنشَأ الجدول VT11 ثُم يُعَاد إلى المُستدعي (العبارة LIMIT لها نفس وظيفة TOP في بعض لهجات SQL، مثل Postgres و Netezza.).

4. تنظيم شيفرات SQL وتأمينها

هذه بعض النصائح والقواعد حول كيفية كتابة استعلامات SQL تراعي أفضل الممارسات، وذات مقروئية عالية.

ا. أسماء الجداول والأعمدة

هناك طريقتان شائعتان لكتابة أسماء الجداول والأعمدة، وهما CamelCase و snake_case كما يوضح المثال التالي:

```
SELECT FirstName, LastName
FROM Employees
WHERE Salary > 500;
SELECT first_name, last_name
FROM employees
WHERE salary > 500;
```

يجب أن تعطي الأسماء فكرة عمَّا هو مُخزَّن في الكائن المُسمَّى. هناك نقاش محتدم حول ما إذا كان الأفضل أن تكون أسماء الجداول بصيغة المفرد أو الجمع، ولكنَّ الشائع استخدام صيغة الجمع.

تُقلِّل إضافة سابقات أو لاحقات، مثل `tbl` أو `col`، إلى الأسماء من مقروئية الشيفرة، لذلك يُفضَّل تجنبها. إلا أنَّها قد تكون ضرورية في بعض الأحيان لتَجنُّب التداخل مع الكلمات المفتاحية في SQL، وغالبًا ما تُستخدم مع المنبهات (triggers) والفهارس (والتي لا تُذكَر أسماؤها في الاستعلامات عادةً).

ب. الكلمات المفتاحية

الكلمات المفتاحية في SQL ليست حسَّاسة لحالة الأحرف ولكن تغلُب كتابتها بأحرف كبيرة.

ج. المسافات البادئة

لا يوجد معيار مقبول ومُوحَّد للمسافات البادئة (Indenting) لكنَّ الجميع يتفق على أنَّ حشر كل شيء في سطر واحد أمر سيء مثل:

```
SELECT d.Name, COUNT(*) AS Employees FROM Departments AS d JOIN
Employees AS e ON d.ID =
e.DepartmentID WHERE d.Name != 'HR' HAVING COUNT(*) > 10 ORDER
BY COUNT(*) DESC;
```

أضعف الإيمان أن تضع كل عبارة في سطر جديد، مع تقسيم السطور الطويلة:

```
SELECT d.Name,
 COUNT(*) AS Employees
FROM Departments AS d
JOIN Employees AS e ON d.ID = e.DepartmentID
WHERE d.Name != 'HR'
HAVING COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

في بعض الأحيان، تُوضع نفس المسافة البادئة قبل الأسطر التي تَعقُب الكلمات المفتاحية في SQL:

```
SELECT   d.Name,
 COUNT(*) AS Employees
FROM     Departments AS d
JOIN     Employees AS e ON d.ID = e.DepartmentID
WHERE    d.Name != 'HR'
HAVING COUNT(*) > 10
ORDER BY COUNT(*) DESC;
```

(يمكن القيام بذلك أيضًا عند محاذاة الكلمات المفتاحية في SQL إلى اليمين.)

هناك طريقة شائعة أخرى، وهي وضع الكلمات المفتاحية المُهمَّة في سطور خاصَّة على النحو التالي:

```
SELECT
    d.Name,
 COUNT(*) AS Employees
FROM
    Departments AS d
JOIN
    Employees AS e
 ON d.ID = e.DepartmentID
WHERE
    d.Name != 'HR'
HAVING
    COUNT(*) > 10
```

```
ORDER BY
    COUNT(*) DESC;
```

تُحسِّن المحاذاة الرأسية للعبارات المتماثلة من مقروئية الشيفرة:

```
SELECT Model,
       EmployeeID
FROM Cars
WHERE CustomerID = 42
   AND Status = 'READY';
```

استخدام عدَّة أسطر يُصعِّب تضمين أوامر SQL في لغات البرمجة الأخرى. إلا أنَّ العديد من اللغات لديها آلية للتعامل مع السلاسل النصية مُتعدِّدة الأسطر، مثل `@"..."` في C# أو `"""..."""` في بايثون أو `R"(...)"` في C++.

د. العبارة SELECT *

تعيد العبارة `SELECT *` جميع الأعمدة بنفس ترتيب ظهورها في الجدول. عند استخدام `SELECT *`، فقد تتغيَّر البيانات المُعادة من الاستعلام كلَّما تغيَّر تعريف الجدول. وهذا يضعف توافقية الإصدارات المختلفة من التطبيق أو قاعدة البيانات مع بعضها بعضًا. علاوة على ذلك، فإنَّ قراءة الأعمدة غير الضرورية قد يرفع من مساحة التخزين المُستخدَمة، أو الدخل / الخرج الشبكي (network I/O)، لذا عليك دائمًا تحديد العمود (أو الأعمدة) الذي تريد استردادها صراحة:

```
SELECT *                                  -- تجنّب هذا
SELECT ID, FName, LName, PhoneNumber      -- هذا أفضل
FROM Emplopees;
```

(لا تنطبق هذه الاعتبارات عند إجراء استعلامات تفاعلية [interactive queries].)

بالمقابل، ليس هناك ضرر من استخدام `SELECT *` في استعلام فرعي لعبارة `EXISTS`، ذلك أنَّ `EXISTS` تتجاهل البيانات الفعلية على أي حال (إذ تكتفي بالتحقق من أنّه تمّ العثور على صفّ

واحد على الأقل). للسبب نفسه، لا فائدة من إدراج أي عمود (أو أعمدة) مُعيَّنة في عبارة EXISTS، لذلك يُفضَّل استخدام SELECT *:

```
-- سرد الأقسام التي لم يُعيّن فيها أيّ موظف حديثا
SELECT ID,
       Name
FROM Departments
WHERE NOT EXISTS (SELECT *
 FROM Employees
 WHERE DepartmentID = Departments.ID
 AND HireDate >= '2015-01-01');
```

ه. عمليات الدمج

يجب دائمًا استخدام عمليات الدمج الصريحة (Explicit joins)؛ لأنَّ عمليات الدمج المضنية (implicit joins) تطرح العديد من المشاكل، مثلًا:

- في عمليات الدمج الضمنية، يكون شرط الدمج داخل عبارة WHERE مخلوطًا مع شروط أخرى، وذلك يُصعِّب معرفة الجداول المدمجة وكيفية دمجها؛ أضف إلى تعاظم خطر حدوث أخطاء بسبب هذه النقطة.
- في SQL القياسية، عمليات الدمج الصريحة هي الطريقة الوحيدة لاستخدام الدمج الخارجي:

```
SELECT d.Name,
e.Fname || e.LName AS EmpName
FROM Departments AS d
LEFT JOIN Employees AS e ON d.ID = e.DepartmentID;
```

- يتيح الدمج الصريح استخدام عبارة USING كما يوضِّح المثال التالي:

```
SELECT RecipeID,
Recipes.Name,
```

```
COUNT(*) AS NumberOfIngredients
FROM Recipes
LEFT JOIN Ingredients USING (RecipeID);
```

(يتطلب هذا أن يستخدم كلا الجدولين اسم العمود نفسه. فتزيل `USING` تلقائيًا العمود المُكـرَّر من النتيجة، وهكذا سيُعيد الدمج في الاستعلام أعلاه عمود `RecipeID` واحد.)

5. حقن SQL

حقن SQL هي تقنية يستخدمها القراصنة للوصول إلى جداول قاعدة بيانـات موقـع معيَّن عن طريق حقن تعليمات SQL في حقل إدخال.

إذا لم يكن خادم الويب مُجهَّزا للتعامل مع هجمات حقن SQL، فيمكن للمخترقين خداع قاعدة البيانات، وجعلها تنفِّذ شيفرة SQL إضافية قد تمكِّنهم من ترقيـة صـلاحيات حسـاباتهم، أو الوصـول إلى المعلومات الشخصية لحساب آخر، أو إجراء أي تعديلات أخرى على قاعدة البيانات.

لنفترض أنَّ استدعاء معالج تسجيل الدخول إلى موقعك يبدو كما يلي:

```
https://somepage.com/ajax/login.ashx?
username=admin&password=123
```

الآن في login.ashx، ستقرأ القيم التالية:

```
strUserName = getHttpsRequestParameterString("username");
strPassword = getHttpsRequestParameterString("password");
```

يمكنك استعلام قاعدة البيانات للتحقق ممَّا إذا كان هنـاك مسـتخدم لـه كلمـة المـرور هـذه، لـذا ستنشئ استعلام SQL التالي:

```
txtSQL = "SELECT * FROM Users WHERE username = '" + strUserName
+ "' AND password = '"+ strPassword +"'";
```

سـيعمل هـذا الاسـتعلام بلا مشـاكل إذا لم يحتـو اسـم المسـتخدم وكلمـة المـرور على علامـات

اقتباس. ولكن إن احتوى أحد المعاملات على علامات اقتباس، فإنَّ شيفرة SQL المُرسلة إلى قاعدة البيانات ستبدو كما يلي:

```
-- strUserName = "d'Alambert";
txtSQL = "SELECT * FROM Users WHERE username = 'd'Alambert' AND
password = '123'";
```

سينتج عن هذا خطأ في الصياغة، لأنَّ علامة الاقتباس بعد d في d'Alambert تنتهي بشيفرة SQL.

يمكنك تصحيح هذا عن طريق تهريب (escaping) علامات الاقتباس في اسم المستخدم وكلمة المرور على النحو التالي:

```
strUserName = strUserName.Replace("'", "''");
strPassword = strPassword.Replace("'", "''");
```

هناك حلٌّ آخر أفضل، وهو استخدام المعاملات:

```
cmd.CommandText = "SELECT * FROM Users WHERE username =
@username AND password = @password";
cmd.Parameters.Add("@username", strUserName);
cmd.Parameters.Add("@password", strPassword);
```

إذا لم تستخدم المعاملات، ونسيت استبدال علامات الاقتباس ولو في قيمة واحدة، فيمكن للقرصان استخدام هذا لتنفيذ أوامر SQL في قاعدة البيانات الخاصة بك. على سبيل المثال، يمكن للقرصان أن يعيِّن كلمة المرور إلى القيمة التالية:

```
lol'; DROP DATABASE master; --
```

وبعدها ستبدو SQL كالتالي:

```
"SELECT * FROM Users WHERE username = 'somebody' AND password =
'lol'; DROP DATABASE master; --'";
```

لسوء الحظ، هذه شيفرة SQL صحيحة، وستنفِّذها قاعدة البيانات DB! هذا النوع من الهجمات يسمَّى حقن SQL.

هناك أشياء أخرى كثيرة يمكن أن يقوم بها القرصان، مثل سرقة عناوين البريد الإلكتروني الخاصة بالمستخدمين، أو سرقة كلمات المرور خاصتهم، أو سرقة أرقام بطاقات الائتمان، أو سرقة أي نوع من البيانات في قاعدة البيانات. لهذا السبب، عليك دائمًا تهريب السلاسل النصية.

لمَّا كان النسيان طبيعة في الإنسان، ينصح الكثيرون باستخدام المعاملات دائمًا. لأنَّ إطارات لغة البرمجة المُستخدمَة تتكفَّل بتهريبها نيابة عنك.

ا. مثال على حقن بسيط

إذا تم إنشاء عبارة SQL على النحو التالي:

```
SQL = "SELECT * FROM Users WHERE username = '" + user + "' AND
password ='" + pw + "'";
db.execute(SQL);
```

سيكون بمقدور القرصان سرقة بياناتك عن طريق إعطاء كلمة مرور من هذا القبيل `pw' or '1'='1`؛ وهكذا تصبح عبارة SQL الناتجة على النحو التالي:

```
SELECT * FROM Users WHERE username = 'somebody' AND password
='pw' or '1'='1'
```

العبارة `'1'='1'` صحيحة دائمًا، لذلك سيتم اختيار كل الصفوف. لمنع هذا، استخدم معاملات SQL على النحو التالي:

```
SQL = "SELECT * FROM Users WHERE username = ? AND password
= ?";
db.execute(SQL, [user, pw]);
```

11

مواضيع متقدمة في SQL

سيعرض هذا الفصل عددًا من المواضيع المتقدمة في SQL مثل العروض (Views)، وإنشاء السلاسل إدارة الصلاحيات، واستخدام ملفات XML في الاستعلامات، والمفاتيح الرئيسية والفهارس وأرقام الصفوف.

1. العروض Views

ا. العروض البسيطة

تُستخدَم العروض (View) لترشيح الصفوف من الجدول الأساسي، أو الاكتفاء بعرض بعض الأعمدة منه فقط:

```
CREATE VIEW new_employees_details AS
SELECT E.id, Fname, Salary, Hire_date
FROM Employees E
WHERE hire_date > date '2015-01-01';
```

يختار (select) المثال التالي من النتائج المعروضة في العرض (view):

```
select * from new_employees_details
```

الخرج الناتج:

Hire_date	Salary	FName	Id
24-07-2016	500	Johnathon	4

ب. العروض المركبة

يمكن أن تكون العروض مُعقَّدة ومركَّبة (complex views) مثل تجميعات (aggregations)، أو عمليات دمج، أو استعلامات فرعية، ...إلخ. المهم أن تتأكَّد دائمًا من إضافة أسماء الأعمدة لكل شيء تختاره:

```
Create VIEW dept_income AS
SELECT d.Name as DepartmentName, sum(e.salary) as TotalSalary
FROM Employees e
JOIN Departments d on e.DepartmentId = d.id
GROUP BY d.Name;
```

يمكنك الآن الاختيار (SELECT) كما تختار من أي جدول عادي:

```
SELECT *
FROM dept_income;
```

الخرج الناتج:

DepartmentName	TotalSalary
HR	1900
Sales	600

ج. العروض المادية

العروض المادية (Materialized Views) هي العروض التي تكون نتائجها مُخزَّنة في ذاكرة مادية، وتُحدَّث دوريًا حتى تظل متزامنة مع القيم الحالية، وهي مفيدة في تخزين نتائج الاستعلامات المُعقَّدة طويلة الأمد التي لا تعتمد على نتائج الوقت الحقيقي (realtime results).

يمكن إنشاء العروض المادية في Oracle و PostgreSQL كما تُوفِّر أنظمة قواعد البيانات الأخرى وظائف مماثلة، مثل العروض المُفهرَسة (indexed views) في SQL Server، أو جداول الاستعلام المادية (materialized query tables) في DB2.

هذا مثال على العروض المادية في PostgreSQL:

```
CREATE TABLE mytable (number INT);
INSERT INTO mytable VALUES (1);
CREATE MATERIALIZED VIEW myview AS SELECT * FROM mytable;
```

```
SELECT * FROM myview;
number
--------
1
(1 row)
INSERT INTO mytable VALUES(2);
SELECT * FROM myview;
number
--------
1
(1 row)
REFRESH MATERIALIZED VIEW myview;
SELECT * FROM myview;
number
--------
1
2
(2 rows)
```

2. استعمال الفهارس (Indexes)

الفهارس هي بنيات تحتوي على مؤشِّرات تشير إلى محتويات جدول مُرتَّب ترتيبًا مُحدَّدًا، تساعد بذلك على لتحسين أداء الاستعلامات والعثور على النتائج بسرعة؛ فهي تشبه فهرس الكتاب، حيث تُفهرَس الصفحات (صفوف الجدول) بأرقام مُميَّزة تُسهِّل الوصول إليها مباشرةً دون البحث في كامل الصفحات أو البيانات للعثور على ما تبحث عنه.

توجد عدَّة أنواع من الفهارس، ويمكنك إنشاؤها على أيِّ جدول. يُحسِّن استخدَام الفهارس مع الأعمدة في عبارات `WHERE` أو `JOIN` أو `ORDER BY` أداء الاستعلامات تحسينًا كبيرًا.

ا. إنشاء فهرس

تنشئ الشيفرة التالية فهرسًا للعمود `EmployeeId` في الجدول `Cars`:

```
CREATE INDEX ix_cars_employee_id ON Cars (EmployeeId);
```

يحسِّن استخدام الفهرس سرعة الاستعلامات التي تحاول أن تُرتِّب أو تختار الصفوف وفقًا لقيم `EmployeeId`، كما في المثال التالي:

```
SELECT * FROM Cars WHERE EmployeeId = 1
```

يمكن أن يحتوي الفهرس على أكثر من عمود واحد كما في المثال التالي:

```
CREATE INDEX ix_cars_e_c_o_ids ON Cars (EmployeeId, CarId,
OwnerId);
```

في هذه الحالة، سيكون الفهرس مفيدًا للاستعلامات التي تحاول أن تُرتِّب أو تختار السجلات وفقًا لجميع الأعمدة المُضمَّنة في حال كانت مجموعة الشروط مُرتَّبة بالطريقة نفسها. هذا يعني أنَّه عند البحث عن البيانات وجلبها، سيكون بمقدورك العثور على الصفوف المُراد إعادتها بالاستعانة بالفهرس بسرعة بدلًا من هدر الوقت بالبحث في كامل الجدول. فمثلًا، يستخدم المثال التالي الفهرس الثاني:

```
SELECT * FROM Cars WHERE EmployeeId = 1 Order by CarId DESC
```

يفقد الفهرس هذه المزايا في حال كان الترتيب مختلفًا كما يبيِّن المثال التالي:

```
SELECT * FROM Cars WHERE OwnerId = 17 Order by CarId DESC
```

لم يعد الفهرس مفيدًا الآن، لأنَّه يتوجَّب على قاعدة البيانات أن تسترجع الفهرس كاملًا، والمرور على جميع قيم الحقل `EmployeeId` والحقل `CarID` بُغيَة إيجاد العناصر التي تحقق الشرط `OwnerId = 17`.

رغم كل ما قلناه، إلا أنَّه من الممكن أن يُستخدَم الفهرس رغم كل شيء؛ فقد يخمِّن مُحسِّن الاستعلامات (query optimizer) أنَّ استرداد الفهرس، والترشيح بحسب قيمة الحقل `OwnerId`،

ثم استرداد الصفوف المطلوبة حصرًا (أي ذات القيمة 17)، سيكون أسرع من استرداد الجدول بالكامل، خاصةً إن كان الجدول كبيرًا.

ب. محو فهرس أو تعطيله وإعادة إنشائه

تُستخدَم التعليمة DROP لمسح الفهارس أيضًا. ففي المثال التالي، تمحو DROP فهرسًا يُسمَّى ix_cars_employee_id في الجدول Cars:

```
DROP INDEX ix_cars_employee_id ON Cars;
```

تحذف DROP الفهرس نهائيًا، وفي حال كان الفهرس مُجمَّعًا (clustered)، فسيُزَال التجميع، ولن يكون بالإمكان إعادة بنائه دون تكرار عملية إنشاء الفهرس من جديد، وهي عمليَّة يمكن أن تكون بطيئة ومكلفة من الناحية الحسابية.

هناك حل آخر، وهو تعطيل (disable) الفهرس بدل حذفه:

```
ALTER INDEX ix_cars_employee_id ON Cars DISABLE;
```

يسمح هذا للجدول بالاحتفاظ ببنية الفهرس وبياناته الوصفية (metadata) إضافةً إلى إحصائيات الفهرس، وهكذا سيَسهُل تقييم التغيير الحاصل؛ وإذا لزم الأمر، سيكون من الممكن تفعيل الفهرس وإعادة بنائه لاحقًا دون الاضطرار إلى إعادة إنشائه من الصفر:

```
ALTER INDEX ix_cars_employee_id ON Cars REBUILD;
```

ج. الفهارس المُرتَّبة

إذا كانت الفهارس مُرتَّبَة بنفس الترتيب الذي ستُسترجَع به، فلن تُجرِي التعليمة SELECT أيَّ ترتيب إضافي أثناء الاسترجاع.

بفرض لدينا الفهرس التالي:

```
CREATE INDEX ix_scoreboard_score ON scoreboard (score DESC);
```

فعند تنفيذ الاستعلام التالي:

```
SELECT * FROM scoreboard ORDER BY score DESC;
```

لن يُجري نظام قاعدة البيانات أيَّ ترتيب إضافي طالما أنَّه سيبحث في الفهرس وفق ذلك الترتيب.

د. الفهرس الجزئي أو المُرشَّح

تتيح SQL Server و SQLite إنشاء فهارس تحوي جزءًا من الأعمدة، وكذلك جزءًا من الصفوف ويدعى هذا النوع من الفهارس «بالفهارس الجزئية أو المُرشَّحة» (Partial or Filtered Index).

بفرض أنَّ لدينا جدول للطلبيات يدعى orders وفيه الحقل `order_state_id` الذي تمثِّل القيمة 2 فيه الحالة «مكتملة»، وتمثِّل القيمة 1 فيه الحالة «غير مكتملة».

إليك الاستعلام التالي:

```
SELECT id, comment
     FROM orders
  WHERE order_state_id =  1
     AND product_id = @some_value;
```

يتيح لك استخدام مفهوم الفهرسة الجزئية تقييد الفهرس (limit the index)، بحيث لا تُضمَّن إلا الطلبات التي لم تكتمل بعد:

```
CREATE INDEX Started_Orders
 ON orders(product_id)
 WHERE order_state_id = 1;
```

سيؤدي هذا إلى تقليل كمّية المؤشّرات التي تنشئها عملية الفهرسة، ويوفِّر مساحة التخزين، ويُقلِّل من تكلفة تحديث ذلك الفهرس.

ه. الفهارس المُتكتِّلة والفريدة والمُرتَّبة

يمكن إسناد عدد من الخصائص لكل فهرس تُحدَّد ساعة إنشائه، أو يمكن أن تضاف إليه لاحقًا.

```
CREATE CLUSTERED INDEX ix_clust_employee_id ON
Employees(EmployeeId, Email);
```

تنشئ عبارة SQL أعلاه فهرسًا مُتكتِّلًا (clustered index) جديدًا لجدول الموظفين `Employees`. والفهارس المُتكتِّلة هي فهارس تتحكَّم في البنية الفعلية للجدول؛ إذ يُرتَّب الجدول الذي يطبِّق عليه هذا النوع من الفهرس ترتيبًا يُطابَق بنية هذا الفهرس. نتيجة لهذا، لا يمكن أن يكون للجدول أكثر من فهرس مُتكتِّلة واحد. ما يعني أنَّ الشيفرة أعلاه ستفشل في حال إضافة فهرس المُتكتِّلة ثانٍ (وتُسمَّى الجداول التي لا تحتوي على فهارس مُتكتِّلة «كوْمَات» [heaps]).

ينشئ المثال التالي فهرسًا فريدًا (unique index) للعمود `Email` في جدول العملاء `Customers`. علاوة على تسريع الاستعلام، يفرض الفهرس أن تكون عناوين البريد الإلكتروني فريدة (غير مُتكرِّرَة) في العمود. وإن حاولت إدراج صفٍّ يحوي بريدًا إلكترونيًا موجودًا سلفًا، فستفشل عملية الإدراج أو التحديث (افتراضيًا).

```
CREATE UNIQUE INDEX uq_customers_email ON Customers(Email);
```

ينشئ المثال التالي فهرسًا لجدول العملاء `Customers`، إذ يضع هذا الفهرس على الجدول قيودًا تنص على أنَّ الحقل `EmployeeID` ينبغي أن يكون فريدًا. (ستفشل هذه العملية إن لم يكن العمود فريدًا؛ أي، إن كان هناك موظف آخر يحمل نفس القيمة.)

```
CREATE UNIQUE INDEX ix_eid_desc ON Customers(EmployeeID);
```

تنشئ الشيفرة التالية فهرسًا مُرتَّبًا ترتيبًا تنازليًا. افتراضيا، تُرتَّب الفهارس (على الأقل في MSSQL server) تصاعديًا، لكن يمكن تغيير هذا السلوك كما يوضِّح المثال التالي:

```
CREATE INDEX ix_eid_desc ON Customers(EmployeeID Desc);
```

و. إعادة بناء فهرس

مع مرور الوقت، قد تصبح الفهارس المُتشعِّبة من النمط B-Tree مُجزَّأة (fragmented) نتيجة عمليات التحديث والحذف والإدراج. في نظام SQLServer، هناك نوعان من الفهارس، الفهارس الداخلية، والتي تكون فيها صفحة الفهرس نصف فارغة (half empty)، والفهارس الخارجية، والتي لا يتطابق فيها ترتيب الصفحة المنطقي مع الترتيب الفعلي). إعادة بناء الفهارس تشبه إلى حدٍّ بعيد حذفها ثمَّ إعادة إنشائها.

يمكن إعادة بناء الفهرس باستخدام الصياغة التالية:

```
ALTER INDEX index_name REBUILD;
```

إعادة بناء الفهارس هي عملية حاجرة (offline operation) افتراضيًا، أي أنَّها تحجر الجدول وتمنع الوصول إليه أو التعديل عليه أثناء تطبيقها، لكنَّ العديد من أنظمة معالجة قواعد البيانات (RDBMS) لا تطبق عملية الحجر تلك على الجدول أثناء إعادة البناء وتسمح بالوصول إليه (تدعى هذه العملية online rebuilding)، كما توفِّر بعض أنظمة قواعد البيانات بدائل أخرى لإعادة بناء الفهارس، مثل `REORGANIZE` (في SQLServer) أو `COALESCE / SHRINK SPACE` (في Oracle).

ز. الإدراج باستخدام فهرس فريد

ستفشل الشيفرة التالية في حال عُيِّن فهرس فريد للعمود `Email` في جدول العملاء:

```
UPDATE Customers SET Email = "richard0123@example.com" WHERE id
= 1;
```

تقترح هذه الشيفرة بديلًا ممكنًا في مثل هذه الحالة:

```
UPDATE Customers SET Email = "richard0123@example.com" WHERE id
= 1 ON DUPLICATE KEY;
```

3. التسلسلات

التسلسلات (Sequences) هي سلاسل من الأعداد. المثال التالي ينشئ تسلسلًا يبدأ من 1000، ويتزايد بمقدار 1.

```
 CREATE SEQUENCE orders_seq
 START WITH     1000
 INCREMENT BY   1;
```

في المثال التالي، سنستخدم مرجعًا (`seq_name.NEXTVAL`) يشير إلى القيمة التالية في التسلسل.

تنبيه: في كل عبارة، تكون هناك قيمة واحدة فقط من التسلسل. أي أنَّه إذا كانت هناك عدة مراجع إلى القيمة التالية في التسلسل (NEXTVAL) في عبارة مُعيَّنة، فستشير جميع تلك المراجع إلى نفس الرقم من السلسلة.

يمكن استخدام القيمة التالية `NEXTVAL` في عبارات الإدراج `INSERTS`:

```
INSERT INTO Orders (Order_UID, Customer)
 VALUES (orders_seq.NEXTVAL, 1032);
```

كما يمكن أن تُستخدَم في عمليات التحديث:

```
UPDATE Orders
SET Order_UID = orders_seq.NEXTVAL
WHERE Customer = 581;
```

ويمكن أيضًا أن تُستخدم في عبارات الاختيار SELECT:

```
SELECT Order_seq.NEXTVAL FROM dual;
```

4. المرادفات (Synonyms)

المرادف (Synonym) هو كُنيةَ أو اسم بديل لكائن في قاعدة بيانات، وقد يكون هذا الكائن جدولًا أو معرضًا أو إجراءً مُخزَّنًا، أو سلسلة ...إلخ.

يوضِّح المثال التالي كيفيَّة إنشاء المرادفات:

```
CREATE SYNONYM EmployeeData
FOR MyDatabase.dbo.Employees
```

5. العبارة TRY / CATCH

تُستخدَم عبارة TRY / CATCH لإمساك الأخطاء في الشيفرات التي يُتوقَّع أن تطرح أخطاء.

ستفشل عمليتا الإدراج في المثال التالي بسبب خطأ في صياغة التاريخ:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, 'not a date', 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

في المثال التالي، ستكتمل عَمليتا الإدراج دائمًا:

```
BEGIN TRANSACTION
BEGIN TRY
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    INSERT INTO dbo.Sale(Price, SaleDate, Quantity)
    VALUES (5.2, GETDATE(), 1)
    COMMIT TRANSACTION
END TRY
BEGIN CATCH
    THROW
    ROLLBACK TRANSACTION
END CATCH
```

6. GRANT و REVOKE

تُستخدَم العبارة GRANT لمنح الإذن لمستخدم ما بإجراء عملية على قاعدة البيانات، فيما تُستخدَم العبارة REVOKE لسحب الإذن أو الصلاحية منه.

يمنح المثال التالي الإذن للمستخدمَين User1 و User2 بإجراء العمليتين SELECT و UPDATE على الجدول Employees.

```
GRANT SELECT, UPDATE
ON Employees
TO User1, User2;
```

يسحب المثال التالي من المستخدمَين User1 و User2 صلاحية تنفيذ العمليتين SELECT و UPDATE على جدول الموظفين.

```
REVOKE SELECT, UPDATE
ON Employees
FROM User1, User2;
```

7. استخدام ملفات XML في SQL

يمكن تعريف جدول بيانات من ملف XML واستخدامه في استعلامات SQL مثل أيٍّ جدول عادي:

```
DECLARE @xmlIN XML = '<TableData>
<aaa Main="First">
    <row name="a" value="1" />
    <row name="b" value="2" />
    <row name="c" value="3" />
</aaa>
<aaa Main="Second">
    <row name="a" value="3" />
    <row name="b" value="4" />
    <row name="c" value="5" />
</aaa>
<aaa Main="Third">
    <row name="a" value="10" />
    <row name="b" value="20" />
    <row name="c" value="30" />
</aaa>
</TableData>'
SELECT t.col.value('../@Main', 'varchar(10)') [Header],
t.col.value('@name', 'VARCHAR(25)') [name],
t.col.value('@value', 'VARCHAR(25)') [Value]
FROM @xmlIn.nodes('//TableData/aaa/row') AS t (col)
```

الخرج الناتج:

```
Header        name      Value
First          a          1
First          b          2
First          c          3
Second         a          3
Second         b          4
Second         c          5
```

```
Third          a          10
Third          b          20
Third          c          30
```

8. رقم الصف (row number)

يمكن استخدام أرقام الصفوف في استعلامات SQL عبر الدالة ROW_NUMBER.

يحذف المثال التالي جميع السجلات خلا السجل الأخير (جدول بعلاقة "واحد إلى كثير" - 1 to Many)

```
WITH cte AS (
   SELECT ProjectID,
     ROW_NUMBER() OVER (PARTITION BY ProjectID ORDER BY
InsertDate DESC) AS rn
   FROM ProjectNotes
)
DELETE FROM cte WHERE rn > 1;
```

تضمِّن الشيفرة التالية رقم الصف وفقًا لترتيب الطلبية:

```
SELECT
   ROW_NUMBER() OVER(ORDER BY Fname ASC) AS RowNumber,
   Fname,
   LName
FROM Employees
```

يقسِّم المثال التالي أرقام الصفوف إلى مجموعات وفقًا لمعيار مُحدِّد:

```
SELECT
  ROW_NUMBER() OVER(PARTITION BY DepartmentId ORDER BY
DepartmentId ASC) AS RowNumber,
  DepartmentId, Fname, LName
FROM Employees
```

9. التعابير الجدولية الشائعة

ا. توليد قيم

لا توفِّر معظم قواعد البيانات طريقة أصلية لإنشاء سلاسل الأرقام؛ بيْد أنَّه يمكن استخدام تعبيرات الجدول الشائعة أو التعبيرات الجدولية (common table expressions) مع العوديَّة (recursion) لمحاكاة هذا النوع من الوظائف.

يولِّد المثال التالي تعبيرًا جدوليًا يُسمَّى Numbers، واسم عموده i، ويحتوي أرقام الصفوف (1-5):

```
-- إعطاء اسم الجدول"Numbers` واسم العمود `i` لتخزين الأعداد
WITH Numbers(i) AS (
 -- البداية
 SELECT 1
 -- المعامل UNION ALL ضروري لأجل العودية
 UNION ALL
 -- تعبير التكرار
 SELECT i + 1
 -- التعبير الجدولي الذي أعلنا عنه والمُستخدم كمصدر للعودية
 FROM Numbers
 -- عبارة إنهاء العودية
 WHERE i < 5
)
-- استخدام التعبير الجدولي المُنشأ كما لو كان جدولا عاديا
SELECT i FROM Numbers;
```

الخرج الناتج:

i
1
2

i
3
4
5

يمكن استخدام هذه الطريقة مع أي مجال من الأعداد، وكذلك مع أنواع أخرى من البيانات.

ب. الترقيم العودي لشجيرة

المثال التالي يوضِّح كيفية ترقيم شجيرة (subtree) عوديًا:

```
WITH RECURSIVE ManagedByJames(Level, ID, FName, LName) AS (
 -- البدء بهذا الصف
 SELECT 1, ID, FName, LName
 FROM Employees
 WHERE ID = 1
 UNION ALL
 --  الحصول على الموظفين الذين يعملون تحت إمرة أيّ من المدراء
المُختارين سابقا
 SELECT ManagedByJames.Level + 1,
           Employees.ID,
           Employees.FName,
           Employees.LName
 FROM Employees
 JOIN ManagedByJames
 ON Employees.ManagerID = ManagedByJames.ID
 ORDER BY 1 DESC -- depth-first search البحث الأولي-العميق
)
SELECT * FROM ManagedByJames;
```

الخرج الناتج:

LName	FName	ID	Level
Smith	James	1	1
Johnson	John	2	2
Smith	Johnathon	4	3
Williams	Michael	3	2

ج. الاستعلامات المؤقتة

تتصرَّف الاستعلامات المؤقتة (Temporary query) مثل الاستعلامات المُتشعِّبة (nested subqueries)، إلا أنَّ صياغتها مختلفة.

```
WITH ReadyCars AS (
   SELECT *
    FROM Cars
    WHERE Status = 'READY'
)
SELECT ID, Model, TotalCost
FROM ReadyCars
ORDER BY TotalCost;
```

الخرج الناتج:

TotalCost	Model	ID
200	Ford F-150	1
230	Ford F-150	2

هذا استعلام فرعي مكافئ:

```
SELECT ID, Model, TotalCost
FROM (
    SELECT *
    FROM Cars
    WHERE Status = 'READY'
) AS ReadyCars
ORDER BY TotalCost
```

د. التسلق العودي لشجرة

المثال التالي يوضِّح كيفية تسلق شجرة عوديًا (recursively going up in a tree):

```
WITH RECURSIVE ManagersOfJonathon AS (
 -- البدء بهذا الصف
 SELECT *
 FROM Employees
 WHERE ID = 4
 UNION ALL
 -- الحصول على مدراء كل الصفوف المُختارة سابقا
 SELECT Employees.*
 FROM Employees
 JOIN ManagersOfJonathon
 ON Employees.ID = ManagersOfJonathon.ManagerID
)
SELECT * FROM ManagersOfJonathon;
```

الخرج الناتج:

DepartmentId	ManagerId	PhoneNumber	LName	FName	Id
1	2	1212121212	Smith	Johnathon	4
1	1	2468101214	Johnson	John	2
1	NULL	1234567890	Smith	James	1

ه. التوليد العودي للتواريخ

يولِّد المثال التالي تواريخ مع تضمين الجداول الزمنية لفِرَق العمل:

```
DECLARE @DateFrom DATETIME = '2016-06-01 06:00'
DECLARE @DateTo DATETIME = '2016-07-01 06:00'
DECLARE @IntervalDays INT = 7
-- Transition Sequence =  وقت الاستراحة في المناوبات الليلية
والنهارية
-- RR (Rest & Relax) = 1
-- DS (Day Shift) = 2
-- NS (Night Shift) = 3
;WITH roster AS
(
    SELECT @DateFrom AS RosterStart, 1 AS TeamA, 2 AS TeamB, 3
AS TeamC
    UNION ALL
    SELECT DATEADD(d, @IntervalDays, RosterStart),
       CASE TeamA WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END
AS TeamA,
       CASE TeamB WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END
AS TeamB,
       CASE TeamC WHEN 1 THEN 2 WHEN 2 THEN 3 WHEN 3 THEN 1 END
AS TeamC
    FROM roster WHERE RosterStart < DATEADD(d, -@IntervalDays,
@DateTo)
)
SELECT RosterStart,
       ISNULL(LEAD(RosterStart) OVER (ORDER BY RosterStart),
RosterStart + @IntervalDays) AS RosterEnd,
       CASE TeamA WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN
'NS' END AS TeamA,
       CASE TeamB WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN
'NS' END AS TeamB,
       CASE TeamC WHEN 1 THEN 'RR' WHEN 2 THEN 'DS' WHEN 3 THEN
'NS' END AS TeamC
FROM roster
```

النتيجة المُعَادة:

Results Messages

	RosterStart	RosterEnd	TeamA	TeamB	TeamC
1	2016-06-01 06:00:00.000	2016-06-08 06:00:00.000	RR	DS	NS
2	2016-06-08 06:00:00.000	2016-06-15 06:00:00.000	DS	NS	RR
3	2016-06-15 06:00:00.000	2016-06-22 06:00:00.000	NS	RR	DS
4	2016-06-22 06:00:00.000	2016-06-29 06:00:00.000	RR	DS	NS
5	2016-06-29 06:00:00.000	2016-07-06 06:00:00.000	DS	NS	RR

و. استخدام CONNECT BY في Oracle مع تعبير جدولي عودي

توفِّر الوظيفة CONNECT BY المُستخدمة في Oracle العديد من الميزات المفيدة التي لا يوجد لها مثيل في التعبيرات الجدولية العودية القياسية في SQL.

يحاول المثال التالي محاكاة هذه الميزات (مع بعض الإضافات التكميلية) باستخدام صياغة SQL Server. هذه الوظائف مفيدة للغاية لمطوري Oracle إذ توفِّر لهم العديد من الميزات في الاستعلامات المتشعبة (hierarchical queries) غير الموجودة في قواعد البيانات الأخرى، كما أنَّها مفيدة أيضًا في توضيح استخدامات الاستعلامات المتشعبة عمومًا.

```
 WITH tbl AS (
     SELECT id, name, parent_id
         FROM mytable)
 ,     tbl_hierarchy AS (
     /* Anchor */
     SELECT 1 AS "LEVEL"
         --, 1 AS CONNECT_BY_ISROOT
         --, 0 AS CONNECT_BY_ISBRANCH
         , CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN 0
ELSE 1 END AS CONNECT_BY_ISLEAF
         , 0 AS CONNECT_BY_ISCYCLE
         , '/' + CAST(t.id AS VARCHAR(MAX)) + '/' AS
SYS_CONNECT_BY_PATH_id
         , '/' + CAST(t.name AS VARCHAR(MAX)) + '/' AS
```

```
SYS_CONNECT_BY_PATH_name
         , t.id AS root_id
         , t.*
     FROM tbl t
    WHERE t.parent_id IS NULL -- START WITH parent_id IS NULL
 UNION ALL
 /* العودية*/
 SELECT th."LEVEL" + 1 AS "LEVEL"
         --, 0 AS CONNECT_BY_ISROOT
         --, CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN
1 ELSE 0 END AS CONNECT_BY_ISBRANCH
         , CASE WHEN t.id IN (SELECT parent_id FROM tbl) THEN 0
ELSE 1 END AS CONNECT_BY_ISLEAF
         , CASE WHEN th.SYS_CONNECT_BY_PATH_id LIKE '%/' +
CAST(t.id AS VARCHAR(MAX)) + '/%'
        THEN 1 ELSE 0 END AS CONNECT_BY_ISCYCLE
         , th.SYS_CONNECT_BY_PATH_id + CAST(t.id AS
VARCHAR(MAX)) + '/' AS SYS_CONNECT_BY_PATH_id
         , th.SYS_CONNECT_BY_PATH_name + CAST(t.name AS
VARCHAR(MAX)) + '/' AS     SYS_CONNECT_BY_PATH_name
         , th.root_id
         , t.*
     FROM tbl t
         JOIN tbl_hierarchy th ON (th.id = t.parent_id) --
CONNECT BY PRIOR id = parent_id
     WHERE th.CONNECT_BY_ISCYCLE = 0) -- NOCYCLE
SELECT th.*
 --, REPLICATE(' ', (th."LEVEL" - 1) * 3) + th.name AS
tbl_hierarchy
     FROM tbl_hierarchy th
     JOIN tbl CONNECT_BY_ROOT ON (CONNECT_BY_ROOT.id =
th.root_id)
  ORDER BY th.SYS_CONNECT_BY_PATH_name; -- ORDER SIBLINGS BY
name
```

هذا شرح لميزات CONNECT BY الموضَّحة أعلاه:

- العبارات
  - CONNECT BY: تحدِّد العلاقة التي تعرِّف التشعُّب
  - START WITH: تحدِّد العقدة الجذرية (root nodes).
  - ORDER SIBLINGS BY: تُحدِّد ترتيب النتائج
- المعاملات
  - NOCYCLE: توقِف معالجة فرع معيَّن عند رصد شعبة دورية (loop) لأنَّ الشعب الصالحة هي الشعب غير الدورية (Directed Acyclic)، أي الشعب التي لا يمكن العودة عبرها إلى العقدة نفسها.
- العمليات
  - PRIOR: تحصل على البيانات من العقدة الأب (node's parent).
  - CONNECT_BY_ROOT: تحصل على البيانات من العقدة الجذرية.
- أشباه الأعمدة Pseudocolumns
  - LEVEL: تشير إلى مسافة العقدة من جذرها.
  - CONNECT_BY_ISLEAF: تشير إلى عقدة بدون فروعها.
  - CONNECT_BY_ISCYCLE: تشير إلى عقدة ذات مرجع دائري (circular reference).
- الدوال
  - SYS_CONNECT_BY_PATH: تعيد سلسلة نصية تمثِّل المسار من الجذر إلى العقدة.