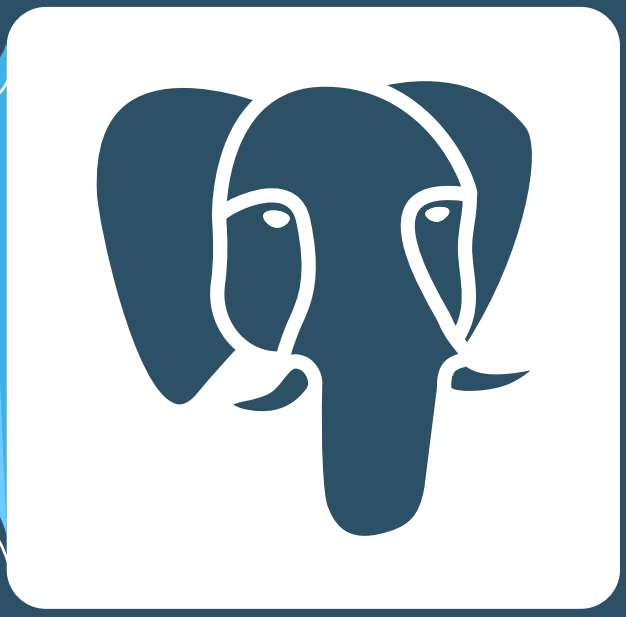


الدليل العملي إلى قواعد بيانات PostgreSQL

مصطفى عطا العايش

أكاديمية حسوب

الدليل العملي إلى قواعد بيانات PostgreSQL

تأليف

مصطفى عطا العايش

تحرير وإشراف

جميل بيلوني

إخراج فني

فرج الشامي

عن الناشر

أُنتج هذا الكتاب برعاية شركة حسوب وأكاديمية حسوب.

تهدف أكاديمية حسوب إلى توفير دروس وكتب عالية الجودة في مختلف المجالات وتقدم دورات شاملة لتعلُّم البرمجة بأحدث تقنياتها معتمدةً على التطبيق العملي الذي يؤهل الطالب لدخول سوق العمل بثقة.

حسوب مجموعة تقنية في مهمة لتطوير العالم العربي. تبني حسوب منتجات تركّز على تحسين مستقبل العمل، والتعليم والتواصل. تدير حسوب أكبر منصتي عمل حر في العالم العربي مستقل وخمسات ويعمل فيهما فريق شاب وشغوف من مختلف الدول العربية.

جدول المحتويات

1. نظرة عامة على Postgres وتاريخها **10**

1.1. لمحة تاريخية 10

1.2. أهم خصائص قواعد بيانات Postgres 12

1.3. متى تختار Postgres؟ 14

1.4. نظرة عامة على نموذج الخادم/عميل في Postgres 14

1.5. خلاصة الفصل 15

2. تثبيت Postgres والتعرف على أساسيات إدارتها لقواعد البيانات **16**

2.1. تثبيت Postgres 16

2.2. أساسيات إدارة قواعد بيانات Postgres 18

2.3. استخدام قاعدة بيانات جاهزة 21

2.4. خلاصة الفصل 22

3. أساسيات استخدام صدفة psql **23**

3.1. أوامر استعراض قاعدة البيانات والجداول 24

3.2. أوامر أخرى أساسية 28

3.3. خلاصة الفصل 30

4. استخدام أساسيات SQL في Postgres **31**

4.1. إنشاء الجداول 31

4.2. تعديل الجداول 35

4.3. إدخال البيانات 36

4.4. استعراض الجداول 37

4.5. الاستعلام عن البيانات 38

4.6. التعديل UPDATE والحذف DELETE 50

4.7. الربط Join 56
4.8. خلاصة الفصل 71

5. مزايا متقدمة في Postgres 72

5.1. العرض View 72
5.2. عبارات الجداول الشائعة (CTE) 78
5.3. دوال النوافذ (Window Functions) 82
5.4. الفهارس Indexes 89
5.5. خلاصة الفصل 92

6. أنواع بيانات خاصة في قواعد بيانات Postgres 93

6.1. المصفوفات (Arrays) 93
6.2. أنواع البيانات التعدادية (Enumerated Data Types) 99
6.3. عرض وتعديل قيم التعداد 101
6.4. نوع البيانات HStore 103
6.5. بيانات بصيغة JSON 106
6.6. التعامل مع التاريخ والوقت 107
6.7. خلاصة الفصل 113

7. إدارة النسخ الاحتياطي في قواعد بيانات Postgres 114

7.1. النسخ الاحتياطي والاستعادة 114
7.2. خلاصة الفصل 122

8. أساسيات إدارة الذاكرة 123

8.1. مسارات تخزين البيانات 123
8.2. معرفة حجم قاعدة البيانات 125
8.3. معرفة حجم الجدول 126
8.4. معرفة حجم الفهرس (index) 127

8.5. قياس حجم الجدول مع الفهارس 127
8.6. خلاصة الفصل 130

9. إدارة الأداء وذاكرة التخزين المؤقتة 131

9.1. خطة التنفيذ (Execution plan) 131
9.2. قيود شرطية على إنشاء الفهارس 136
9.3. ذاكرة التخزين المؤقتة (Cache) 137
9.4. خلاصة الفصل 141

10. أوامر متقدمة في صَدَفة psql 142

10.1. أصناف أوامر psql 142
10.2. أوامر الاتصال 143
10.3. أوامر استعراض أخرى مهمة 144
10.4. أوامر التنسيق 146
10.5. استعراض تاريخ الاستعلامات وحفظه 149
10.6. أوامر التعامل مع المتغيرات 149
10.7. الأوامر الشرطية 150
10.8. أوامر نظام التشغيل 151
10.9. الخروج من صَدَفة postgres 152
10.10. خلاصة الفصل 152
10.11. خاتمة الكتاب 152

تقديم

يأخذ هذا الكتاب بيدك في أمثلة عملية ومتنوعة تغطي أساسيات SQL وأساسيات إدارة قواعد بيانات Postgres، وتمر على مزايا مهمة في Postgres تزيد من فاعلية الاستعلامات ووضوح عبارتها، ويصل هذا الكتاب إلى مواضيع متقدمة في إدارة قواعد البيانات كالنسخ الاحتياطي وتتبع استخدام الذاكرة، وبذلك يتميز بأنه مناسب للمبتدئ والمتوسط ويضع قدمك على سكة المستوى المتقدم في قواعد بيانات Postgres.

بني هذا الكتاب بدايةً على دليل PostgresGuide لمؤلفه Craig Kerstiens المرخص تحت رخصة CC BY-NC أي كان ترجمةً لذلك الدليل فقط، ولكن لمَّا وجدنا الإصدار المستعمل فيه من Postgres قديمًا وينقص الدليل عدة مواضيع إضافية، آثرنا البناء عليه وتحسينه وإضافة كل ما نراه ناقصًا ليحقق الكتاب مبدأ 80/20.

فما يميزه عن غيره، فهو أننا اتبعنا في بداية العمل على هذا الكتاب قاعدة باريتو 80/20، حيث حرصنا على تغطية 20% من مواضيع Postgres التي تُستخدَم في 80% من الحالات في الواقع والمجال العملي، إذ حاولنا الابتعاد عن التفاصيل غير العملية أو قليلة الاستخدام، أي أن هذا الكتاب لا يتطرق إلى شرح أو استخدام الأدوات المتقدمة والتي قد يندر استخدام بعضها (مذكورة في هذا القسم من التوثيق) ولا يتطرق أيضًا إلى كيفية تطوير Postgres بكتابة شيفرات برمجية خاصة (مذكورة في هذا القسم من التوثيق)، فلم نرد تعليمك شيئًا لن تستخدمه إلا في حالات قليلة (تكون قد نسيتها إلى حين استعمالها ;-)).

وفقًا لذلك، حاولنا جاهدين أن يكون هذا الكتاب دليلًا عمليًا ومرجعًا سريعًا للمبرمج، يتعرف فيه على الجزء الأكثر أهمية مما قد يجهله عن قواعد بيانات Postgres ويختبر فيه في ذات الوقت معرفته بالمزايا المتنوعة لقواعد البيانات تلك. باختصار، صُمم هذا الكتاب ليكون رحلة ممتعة فعالة مليئة بالأمثلة المفيدة الواضحة في قواعد بيانات Postgres.

يغطي هذا الكتاب ما يلي:

- لمحة عن الأسس النظرية لقواعد البيانات
- تغطية فعالة لأساسيات استخدام لغة SQL عمليًا
- تغطية شاملة لما تتميز به Postgres عن SQL الصرفة (أي لغة قواعد البيانات العامة)
- أساسيات إدارة قواعد بيانات Postgres (إدارة الذاكرة، الأداء، النسخ الاحتياطي وغيرها)
- أهم التعليمات اللازم معرفتها في صَدَفة psql (واجهة سطر الأوامر الخاصة بPostgres)

في حال أردت أولًا التعمق في SQL، فننصحك بالبدء بكتاب «ملاحظات للعاملين بلغة SQL» أولًا فمعلوم أن لغة SQL هي اللغة الأم لقواعد بيانات SQL (ومنها Postgres)، فلن تجد هذا الكتاب يتعمق كثيرًا في تعليمات SQL لأننا شرحناها بالتفصيل مسبقًا في كتاب ملاحظات للعاملين بلغة SQL ذاك. ننصحك بعد الانتهاء من هذا الكتاب الانتقال إلى كتاب «بوستجريسكل كتاب الوصفات» إن أردت التوسع أكثر في PostgreSQL، فتلك خارطة الطريق التي ننصحك بها لتعلم لغة SQL وقواعد بيانات PostgreSQL من الأساسيات وحتى الاحتراف.

أخيرًا، نرجو أن نكون قد وفقنا في هذا العمل بتوفير دليل عملي نافع يثري المكتبة العربية، والله ولي التوفيق.

جميل بيلوني - مصطفى عطا العايش

07/11/2020

1. نظرة عامة على Postgres وتاريخها

يجدر بنا معرفة بعض المزايا الأساسية لقواعد بيانات Postgres في مقدمة هذا الكتاب، وذلك كي يكون واضحًا للقارئ الفوائد التي سيجنيها من التعرف إليها، وتطوير قدراته في التعامل معها.

إن أردت فهم أي موضوع فهمًا جيدًا، فيجب أن تملك معرفة تاريخية جيدة عنه خصوصًا بداية نشأته وسببها وآثار ذلك وحتى الحاضر؛ وبناءً على ذلك، سنطلع أولًا على تاريخ Postgres منذ بداية ظهورها وحتى يومنا هذا.

1. 1. لمحة تاريخية

1. 1. 1. الولادة الأولى لقواعد بيانات Postgres

في عام 1986 نشر Michael Stonebraker ورقة بحثية في جامعة بيركلي، كاليفورنيا بعنوان The Design of Postgres ليعلن ولادة قواعد بيانات Postgres الأولى، ذكر في ورقته أن قاعدة البيانات هذه هي النسخة المحسنة المطورة من قواعد بيانات سابقة لها اسمها INGRES (أنشئت عام 1975)، ومن هنا جاءت التسمية POST inGRES أي أنها لاحقة لقواعد INGRES، كما ذكر أن أهم أهداف إنشائها هو دعم تخزين أنواع معقدة، والسماح للمستخدمين بإنشاء امتدادات للغة، وغيره من الأهداف المتعلقة بالتخزين والمعالجة، وكانت في ولادتها هذه من أوائل أنظمة قواعد البيانات التي تتيح استخدام أنواع البيانات المتعددة، مع إمكانية شرح العلاقات بين

الجداول بشكل كامل، إلا أنها كانت في ذلك الوقت لا تستخدم لغة الاستعلامات المُهيكلة SQL بل لغة مشابهة خاصة بها.

نُشرَت بعد ذلك في عام 1989 النسخة الأولى من اللغة لعدد قليل من المستخدمين، تبعتها النسخة 2 عام 1990 مع بعض التحسينات، والنسخة 3 في عام 1991 مع تحسينات في إدارة التخزين وتحسينات على محرك الاستعلام الأساسي، ولكن في عام 1993 بلغت كمية طلبات الدعم والتحسينات حدا تجاوز إمكانيات فريق التطوير في ذلك الوقت،فتم إيقاف المشروع في 30 حزيران 1994.

1. 1. 2. الولادة الثانية

قامت جامعة بيركلي بفتح مصدر POSTGRES مما سمح لجميع المستخدمين باستخدام الشيفرة البرمجية والتعديل عليها، فقام Andrew Yu و Jolly Chen المتخرجَين من جامعة بيركلي في عام 1994 بجعل Postgres تستخدم لغة الاستعلامات المُهيكلة SQL وتم إنشاء صدفة psql ونُشرت النسخة الجديدة Postgres95 عام 1995 بعد فترة تجريبية قصيرة، وبرخصة مفتوحة المصدر أيضًا.

1. 1. 3. ظهور PostgreSQL و postgresql.org

تم تغيير اسم نظام قواعد البيانات Postgres95 إلى PostgreSQL للدلالة على أنها تستخدم لغة SQL عام 1996 وظهر أخيرًا الموقع postgresql.org في ذلك العام لتظهر النسخة 6 من النظام عام 1997 ثم تبدأ مسيرة التطوير مفتوحة المصدر من خلال المطورين المتطوعين حول العالم تحت مسمّى (مجموعة تطوير Postgres العالمية).

> سنستعمل في هذا الكتاب الاسم Postgres اختصارًا للاسم PostgreSQL.

1. 2. أهم خصائص قواعد بيانات Postgres

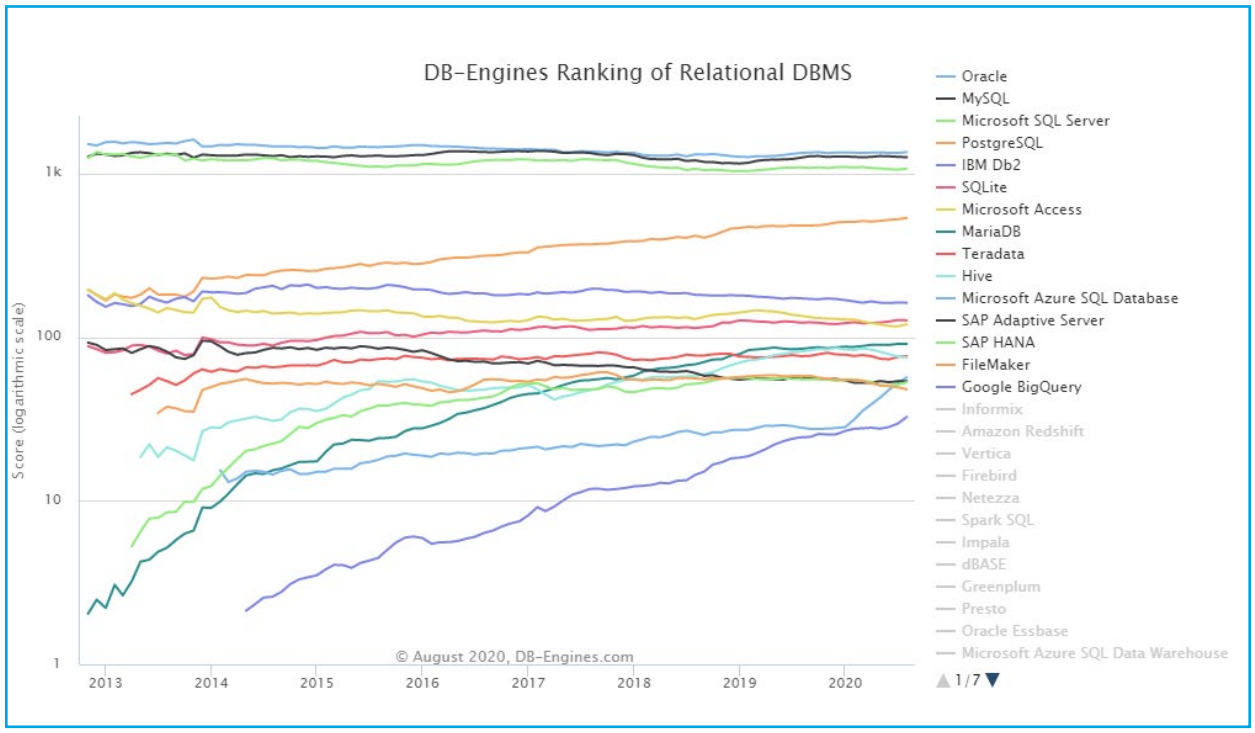
1. 2. 1. أكثر قواعد البيانات تقدّمًا

إن هـذا هـو شـعار قواعـد بيانـات Postgres (وليـس تقييمًـا حقيقيًّـا) ولكنهـا ظلّـت ملتزمـة بـه منـذ نشـأتها عـام 1986، وذلـك عـن طريـق التحسـين المسـتمر وإضافـة المزايـا الجديـدة وتلافـي الأخطاء القديمة.

1. 2. 2. الأسرع نموًّا

حسـب إحصائيـات موقـع db-engines حيـث يعطـي علامـة score لـكل نظـام قاعـدة بيانـات حسـب عـدة عوامـل، منهـا وتيـرة الأسـئلة التقنية علـى المواقـع التقنيـة المشـهورة مثـل Stack Overflow وكذلـك عـدد عـروض العمـل المطلوبـة علـى مواقـع شـهيرة مثـل Indeed، وغيرهـا مـن العوامـل التـي تشـير إلـى ازديـاد الاهتمـام وعـدد المسـتخدمين والمحترفيـن لهـذه الأنظمـة.

ربما يمكنك أن ترى النمو المتسارع لقواعد بيانات PostgreSQL من الشكل التالي بوضوح:



إن ميـزة النمـو فـي الأنظمـة تعنـي أن تعلّمـك اليـوم لقواعـد البيانـات PostgreSQL هـو اسـتثمار صحيـح للمسـتقبل سـواء للعمـل فـي وظيفـة تطلـب منـك خبـرة Postgres أو لكـي تسـتخدمها فـي موقعـك الخـاص.

1. 2. 3. نموذج تخزين البيانات فيها من النوع SQL

نقصد في هذه الفقرة تثبيت فكرة كون Postgres من النوع SQL وليس NoSQL، وذلك يعني باختصار أن قواعد بيانات Postgres تخزّن البيانات ضمن جداول لها قوالب معدّة مسبقًا، ويمكنك الاطلاع على مقال شرح الفروقات بين قواعد بيانات SQL ونظيراتها NoSQL للمزيد من المعلومات عن هذا الموضوع.

1. 2. 4. تسمح بأنواع بيانات غير مُهيكلة

قام المبرمجون المشاركون في تطوير قواعد بيانات Postgres بإضافة النوع JSONB الذي يسمح بتخزين كائنات JSON ضمن الجداول، وبذلك تكون قد استحوذت أيضًا على بعض مزايا قواعد بيانات NoSQL رغم كونها قواعد بيانات مهيكلة.

1. 2. 5. مفتوحة المصدر

يمكن تنزيل الشيفرة المصدرية لقواعد بيانات Postgres من المستودع الرسمي على موقع github وهي مكتوبة بلغة C، ولكونها مفتوحة المصدر فيمكن للمبرمج فهم آلية العمل الدقيقة لأي تفصيل يبحث عنه، كما يمكنه تحسينه وتطويره ونشره إن أراد ليكون جزءًا من نسخة مستقبلية من قواعد البيانات Postgres، أو ليكون رقعة (Patch) لنسخة حالية موجودة.

1. 2. 6. قابلة للتوسيع

ذكرنا قبل قليل أنه يمكن تعديل الشيفرة المصدرية لقواعد بيانات Postgres، ولكننا الآن نتحدث عن إمكانية كتابة توسيعات لها، دون المساس بالشيفرة المصدرية أو الحاجة إلى الاطلاع عليها، وهذا يعني أنه بإمكانك كتابة توابع جديدة خاصة بك وربطها بقاعدة البيانات لاستخدامها لاحقا.

1. 2. 7. موثقة توثيقًا مفصلًا

يمكنك الاطلاع على توثيق قواعد بيانات Postgres الذي يشرح كافة التفاصيل مع تقديم أمثلة لكل منها ودليل تدريبي للمبتدئين كذلك، وهو يشمل كافة المواضيع المتعلقة بها بدءًا من أبسط عبارات SQL وانتهاءً بكيفية توسيع اللغة وكتابة شيفرات برمجية لتحسينها وتطويرها.

1. 3. متى تختار Postgres؟

للإجابة على هذا السؤال، من المهم التفريق بين المبتدئ والمتوسط، فبالنسبة للمبتدئ، فإن أهم ما يحتاج إليه للبدء هو سهولة التثبيت وسهولة التعلم ووجود المصادر العربية، وهذا من أهم ميزات Postgres بالنسبة للمبرمج العربي المبتدئ فأكاديمية حسوب تعتني بإغناء المحتوى العربي الخاص بها نظرًا لسرعة نموها وانتشارها المستمر.

أما لمن تجاوز الأساسيات وبدأ يهتم بالخطوة التالية، فإنك الآن قادر على كتابة استعلامات SQL بمهارة، وترغب الآن بتحسين كفاءة الاستعلامات، أو زيادة موثوقية قاعدة البيانات لديك لمنع حدوث ضياع للبيانات أو لتسريع عمل قاعدة البيانات أو تخفيض حجم قاعدة البيانات في الذاكرة، وفي هذه الحالة فإن Postgres تتيح لك إدارة سهلة وفعالة للأداء، للذاكرة ولكتابة استعلامات أوضح وأكثر سرعة في التنفيذ، كما أن الأدوات المرفقة مع قاعدة بيانات Postgres سهلة التثبيت والاستخدام، ستكون بداية قوية لك في إدارة قواعد البيانات.

في حال كنت متقدمًا في استخدام قواعد البيانات، فقد تضطر لإنشاء توابع خاصة بك، أو لتعديل أمور جوهرية في محرك قواعد البيانات نفسه لعمل تعديلات مخصصة لتطبيقك أو لمنتج مميز له مزايا خاصة، فهناك العديد من الشركات اعتمدت على Postgres لتطوير قواعد بياناتها الخاصة مثل Sun وRed Hat وAmazon وYahoo والقائمة تطول.

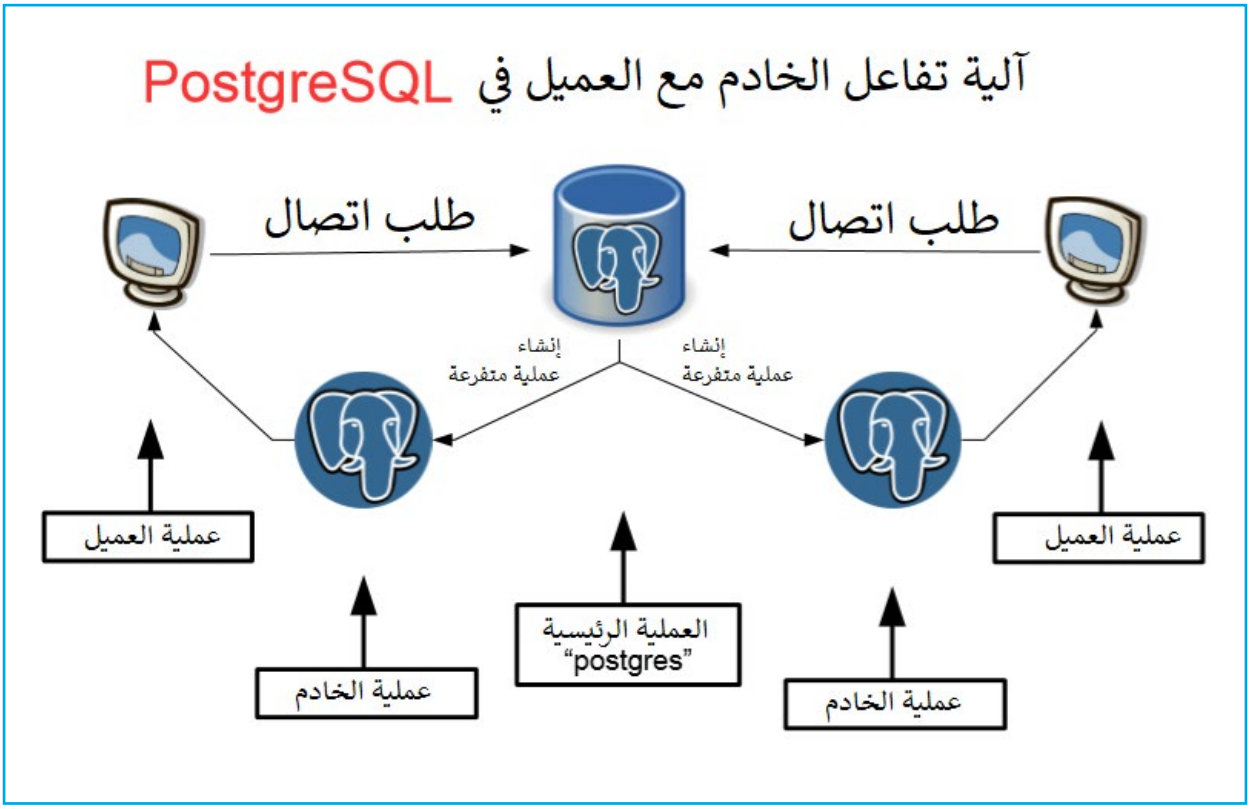
1. 4. نظرة عامة على نموذج الخادم/عميل في Postgres

من المفيد قبل البدء التعرف على بنية نظام قواعد بيانات PostgreSQL وفهم كيفية ارتباط أجزاء النظام ببعضها.

تستخدم PostgreSQL نموذج خادم/عميل (client/server) بحيث تتكون الجلسة من الجزأين التاليين:

- عملية الخادم، اسمها `postgres` تدير ملفات قاعدة البيانات، وتستقبل اتصالات التطبيقات من طرف العميل، وتنفّذ العمليات التي يطلبها العميل على قاعدة البيانات.
- تطبيق العميل الخاص بالمستخدم، (الواجهة الأمامية)، هو التطبيق الذي يريد القيام بعمليات على قاعدة البيانات، ويمكن أن يكون بأشكال عديدة: واجهة سطر أوامر `psql`،

تطبيق رسومي، خادوم ويب أو أداة صيانة قواعد بيانات ما، بعض هذه الأدوات تأتي مرفقة مع نظام قواعد بيانات PostgreSQL كما يمكن أن تكون من تطوير المستخدمين.



يتواصل البرنامج العميل مع الخادم عن طريق الشبكة بواسطة بروتوكول TCP/IP كما هو حال تطبيقات الخادم/عميل المعتادة، وهذا يعني أن العميل لا يمكنه الوصول إلى الملفات الموجودة على الجهاز الخاص بالخادم إذا كان كل منهما على جهاز مختلف.

يمكن للخادم استقبال عدة اتصالات بآن واحد من العملاء، حيث تقوم العملية الأساسية للخادم postgres بعمل fork بحيث تتفرع لعدة عمليات كل منها يعالج أحد هذه الاتصالات لتبقى العملية الأساسية متحررة طوال الوقت من الطلبات وتنتظر استقبال الطلبات الجديدة، وعند انتهاء تنفيذ الطلب يتم تحرير العملية المرتبطة بها وإزالتها (طبعا يبقى كل ذلك غير مرئي للعميل).

1. 5. خلاصة الفصل

عرضنا في هذا الفصل لمحة تاريخية عن Postgres وعن خصائصها والمزايا التي تقدمها كما أجبنا عن سؤال مهم يُسأل دومًا قبل استخدام أي شيء وهو متى نستخدم Postgres ثم ألقينا نظرة على نموذج الاتصال الذي تتبعه Postgres. حان الآن وقت بدء العمل مع PostgreSQL وهذا ما سنتطرق إليه في الفصل التالي.

2. تثبيت Postgres والتعرف على أساسيات إدارتها لقواعد البيانات

سنتعرف في هذا الفصل على طريقة تثبيت Postgres على مختلف أنظمة التشغيل، كما سنهيئ قاعدة البيانات للعمل، ونتعلم كيفية إعطاء الصلاحيات للمستخدمين، ثم نبدأ باستخدام قاعدة البيانات الخاصة بنا.

2. 1. تثبيت Postgres

سنذكر في هذه الفقرة الخطوات الأساسية لتثبيت Postgres على أنظمة التشغيل المختلفة. لقد استعملنا في هذا الكتاب الإصدار 12 من Postgres وقد يكون الإصدار الحالي وقت قراءتك للكتاب مختلفًا، فانتبه إلى ذلك. مهما يكن، تبقى الخطوات ومبدأ العمل نفسه مهما اختلفت الإصدارات، ويمكنك استعمال إصدار أحدث من الإصدار المستعمل ولكن أرجو أن تطلع على سجل التغييرات آنذاك لتجنب الوقوع في أية مشاكل خصوصًا إن أردت استعمال قاعدة بيانات Postgres في بيئة إنتاجية.

2. 1. 1. نظام لينكس

تختلف طريقة التثبيت حسب نوع التوزيعة، ولذلك سنذكر طريق التثبيت في التوزيعات الرئيسية.

أ. أداة yum (فيدورا، ريد هات، سنتوس، لينكس العلمي ...)

يشرح المثال التالي كيفية تثبيت PostgreSQL 12 على نظام CentOS 7.6 x64:

- توجه إلى PostgresSQL Yum Repository واختر إصدار PostgreSQL الذي تريد تثبيته ومن ثم حدد نظام التشغيل الخاص بك وإصداره والمعمارية (استخدمنا في هذا الكتاب نظام سنتوس/ ريدهات الإصدار 7، والمعمارية x86_64).
- نزّل حزمة RPM للمنصة الخاصة بك من الموقع أو نفّذ الأمر التالي من الطرفية Terminal:

```
curl -O https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-x86_64/pgdg-redhat-repo-latest.noarch.rpm
```

- ثبّت الحزمة كما يلي:

```
rpm -ivh pgdg-redhat-repo-latest.noarch.rpm
```

> قد تحتاج إلى استخدام sudo لتنفيذ الأمر السابق.

- ابحث سريعًا لعرض الحزم المتاحة لـ postgres باستخدام الأمر التالي:

```
yum list postgres*
```

> من الممكن أن يُظهر لك الأمر السابق إصدارات قديمة، لذا تأكد من اختيار النسخة المناسبة التي تريد تثبيتها ومن توافق جميع الحزم في رقم الإصدار للخادوم والعميل وأداة contrib (قد لا يكون ذلك ضروريًا دومًا ولكن لأخذ الاحتياط من المشاكل التي قد تظهر).

- ثبّت الحزمة التي تختارها بشكل مشابه للأمر التالي:

```
sudo yum install postgresql12 postgresql12-devel postgresql12-libs postgresql12-server postgresql12-contrib
```

ثبّتنا في الأمر السابق عدة حزم مرتبطة بالإصدار 12، ففيها كلا برنامجي الخادم والعميل، وكذلك المكاتب والملفات الرأسية headers، وكذلك المكاتب والإضافات للمساهمين في هذا المشروع مفتوح المصدر، إذ فضّلنا تثبيتها جميعًا كي تتمكن لاحقًا من استخدام أي منها.

> في حال ظهور خطأ ما أثناء التثبيت فلا تيأس، ابحث عن الخطأ الذي قد يظهر لك ولا شك أنك ستجد إجابة لسؤالك.

ب. أداة apt (أوبنتو، ديبيان، مينت...)

يمكنك تطبيق الأمر التالي باستخدام `apt-get`:

```
sudo apt-get install postgresql
```

ج. أداة pacman (توزيعة Arch Linux)

نستخدم الأمر `pacman` كما يلي:

```
sudo pacman -S postgresql
```

يمكنك أيضًا الرجوع إلى هذا الفيديو، تثبيت وإعداد قاعدة بيانات PostgreSQL لمزيد من التفاصيل حول عملية التثبيت على أنظمة لينكس.

2. 1. 2. نظام ويندوز

يمكنك استخدام أداة التثبيت لنظام ويندوز (ستحتاج إلى VPN إذا كنت في سوريا أو السودان، كالعادة)، وبعد أن تحمّل أداة التثبيت اتبع خطوات التثبيت المعروفة.

2. 1. 3. نظام ماك

يمكنك تنزيل أداة التثبيت لنظام ماك من هذا الرابط واتباع خطوات التثبيت المعروفة كذلك.

2. 2. أساسيات إدارة قواعد بيانات Postgres

لنتمكن من البدء باستخدام Postgres علينا تهيئة قاعدة البيانات، ثم سيُتاح لنا بناء قاعدة بيانات جديدة، وهذا ما سنتعرف عليه في هذه الفقرة.

2. 2. 1. إنشاء عنقود جديد لقواعد بيانات Postgres

بعد أن ثبّت خادم قواعد بيانات Postgres على جهازك، يمكننا الآن البدء باستخدامه، ولكن يجب إنشاء ما يُسمَّى بعنقود قواعد البيانات (Database Cluster) أولًا.

عنقود قواعد البيانات هو مجموعة من قواعد البيانات المُدارة عبر خادوم واحد، ولإنشائه علينا إنشاء المسارات التي ستُخزّن فيها بيانات قاعدة البيانات، كما علينا توليد جداول الإعدادات المشتركة وإنشاء قاعدتي البيانات template1 و postgres، يتم ذلك بشكل تلقائي كما سنرى بعد قليل، فلا تقلق.

قاعدة البيانات template1 تُمثّل قالبًا يتم استخدامه عند إنشاء أي قاعدة بيانات جديدة، أما قاعدة البيانات postgres فهي قاعدة بيانات افتراضية مصممة ليتم استخدامها من المستخدمين والأدوات والتطبيقات الأخرى.

ننشئ أولًا عنقود قاعدة بيانات باستخدام التوجيه `initdb` ضمن صدفة bash كما يلي:

```
sudo /usr/pgsql-12/bin/postgresql-12-setup initdb
```

ملحوظة: قمنا بكتابة المسار الكامل لبرمجية الإعداد والتهيئة (postgres-setup) الخاصة بقواعد بيانات Postgres، ففي حال لم يتم العثور عليها في جهازك، فتأكد من أنك قمت بتثبيتها بطريقة صحيحة، ثم تأكد من أن مسار التثبيت الخاص بها.

يُظهر تنفيذ الأمر السابق المخرجات التالية:

```
Initializing database ... OK
```

ثم يمكنك بدء وتفعيل PostgreSQL باستخدام الأمرين التاليين:

```
sudo systemctl start postgresql-12
sudo systemctl enable postgresql-12
```

سنحصل من الأمر السابق على المخرجات التالية:

```
Created symlink from /etc/systemd/system/multi-user.target.wants/
postgresql-12.service to /usr/lib/systemd/system/postgresql-12.
service.
```

والآن أصبح خادم قاعدة بيانات PostgreSQL مفعّلًا ويمكننا استخدامه.

2. 2. 2. إدارة المستخدمين وإنشاء قاعدة بيانات بدائية

بعد أن ثبتنا قواعد بيانات Postgres وأنشأنا عنقود قواعد البيانات، ستكون الخطوة القادمة هي إضافة وإدارة صلاحيات مستخدمي قواعد البيانات المخزنة، حيث يمكن إنشاء مستخدمين والسماح لهم باستخدام قاعدة بيانات محددة دون الأخرى، أو السماح لهم بتنفيذ أوامر معيّنة، ومنعهم من تنفيذ غيرها.

للدخول إلى الصَّدَفَة (shell) الرئيسية الخاصة بـ Postgres نشغّل برنامج الصدفة بالأمر `psql`، ولكن علينا أولا الانتقال من حساب المستخدم العادي إلى الحساب postgres وذلك عن طريق الأمر التالي:

```
sudo -i -u postgres
```

لتتأكد من المستخدم الذي تعمل عليه، استخدم الأمر `whoami` وقد تظهر لك مخرجات مشابهة لما يلي:

```
[mostafa@hsoub ~]$ whoami
mostafa
```

وعندما تُبدِّل المستخدم نعود وننفّذ الأمر `whoami` مرة أخرى كما يلي:

```
[mayesh@hsoub ~]$ sudo -i -u postgres
[sudo] password for mostafa:
-bash-4.2$ whoami
postgres
```

يمكنك الآن الدخول إلى قاعدة البيانات بتنفيذ الأمر `psql` للبدء بالعمل، ثم سنقوم بإنشاء حساب مستخدم لك، باستخدام الأمر التالي:

```
CREATE USER mostafa WITH PASSWORD 'password';
```

أُنشئ الآن حساب جديد باسم mostafa وبكلمة مرور password، والخطوة التالية هي إنشاء قاعدة بيانات ومنح المستخدم mostafa صلاحية الوصول لها.

```
CREATE DATABASE my_data_base;
```

أُنشئت الآن قاعدة بيانات اسمها my_data_base، وسنمنح الآن الوصول إليها للمستخدم mostafa بالأمر التالي:

```
GRANT ALL PRIVILEGES ON DATABASE my_data_base TO mostafa;
```

مُنح الآن المستخدم mostafa كل الصلاحيات في قاعدة البيانات، حيث أن هنالك عدة أنواع مختلفة من الصلاحيات:

```
SELECT, INSERT, UPDATE, DELETE, RULE, REFERENCES, TRIGGER, CREATE,
TEMPORARY, EXECUTE, USAGE
```

أما إذا أردنا منح واحدة منها فقط، فيمكننا تنفيذ الأمر التالي:

```
GRANT SELECT ON DATABASE my_data_base TO mostafa;
```

تسمح `GRANT SELECT` في هذا الأمر للمستخدم mostafa باستخدام استعلامات `SELECT` فقط في قاعدة البيانات my_data_base.

2. 3. استخدام قاعدة بيانات جاهزة

سنصمم العديد من الجداول في قاعدة البيانات الخاصة بنا في هذا الكتاب، ولكننا سنستعين بقاعدة بيانات جاهزة يمكن تنزيلها، لنتمكن من المتابعة دون الحاجة إلى إنشاء بعض الجداول وإدراج البيانات الواردة ضمنها.

2. 3. 1. التثبيت المحلي

سيتعيَّن عليك أوَّلًا تنزيل البيانات، ثم تنزيلها في قاعدة البيانات.

> تُنفَّذ التعليمات التالية في صَدَفة bash المُعتادة، وليس ضمن psql.

```
curl -L -O http://cl.ly/173L141n3402/download/example.dump
createdb hsoubguide
pg_restore --no-owner --dbname hsoubguide example.dump
```

سنتحدث عن الأمر `pg_restore` في فقرة لاحقة، وهو مسؤول عن استرجاع قاعدة بيانات من ملف، أما الأمر `createdb` فهو يُنشئ قاعدة بيانات في Postgres اسمها hsoubguide.

2. 3. 2. الاتصال بقاعدة البيانات

بعد أن أنشأت قاعدة البيانات الخاصة بك hsoubguide فعليك الآن الدخول إلى psql والاتصال بقاعدة البيانات هذه، ويمكنك القيام بذلك بطريقتين:

- تحديد قاعدة البيانات عن عن طريق التوجيه dbname كما يلي:

```
psql --dbname hsoubguide
```

- تحديد قاعدة البيانات من داخل صَدفة psql باستخدام الأمر \c كما يلي:

```
postgres=# \c hsoubguide
You are now connected to database "hsoubguide" as user "postgres".
```

2. 4. خلاصة الفصل

ثبَّتنا في هذا الفصل Postgres، وتعرفنا على كيفية الدخول إلى صدفة psql ثم أنشأنا قاعدة بيانات بدائية، وتعرفنا على كيفية تحديد صلاحية المستخدمين، وسنتعرف في الفصل التالي على بعض الأوامر التي يمكننا تنفيذها داخل صدفة psql ثم ننتقل إلى تنفيذ أوامر SQL بعد ذلك.

3. أساسيات استخدام صدفة psql

سنستخدم صدفة `psql` طوال الوقت في هذا الكتاب، وسنستخدم بعض الرايات فيها أيضًا، لذا سنخصص هذا الفصل للتعرف على بعض أهم الأوامر فيها.

صدفة `psql` هي برنامج الواجهة التفاعلية للاتصال بـ Postgres ولها العديد من الرايات للتحكم بالاتصال منها:

- الراية `-h` لتحديد المضيف المراد الاتصال به (سواء عن طريق عنوان IP أو عن طريق اسم المضيف إن كان يمكن لخادوم DNS التعرف إليه)
- الراية `-U` لتحديد اسم المستخدم المراد الاتصال من خلاله
- الراية `-p` المنفَذ port المراد الاتصال عبره (المنفذ الافتراضي هو 5423)

```
psql -h localhost -U username hsoubguide
```

كما يمكن استخدام سلسلة نصية كاملة كوسيط واحد، تحتوي محددات الدخول إلى قاعدة البيانات:

```
psql "dbname=hsoubguide host=10.11.108.107 user=postgres
password=pass123456 port=5432 sslmode=require"
```

بعد نجاح الاتصال يمكن البدء بتطبيق الاستعلامات، كما يمكن استخدام أوامر معينة، ويمكن تنفيذ الأمر `?\` للحصول على قائمة بجميع الأوامر المتاحة، والتي سنشرح بعضًا من أهمها في الفقرات التالية.

3. 1. أوامر استعراض قاعدة البيانات والجداول

سنستعرض في هذه الفقرة العديد من الأوامر لاستعراض قواعد البيانات والجداول، ووصف هذه الجداول.

3. 1. 1. استعراض جميع قواعد البيانات الموجودة

استخدم الأمر `\l` لاستعراض قائمة بجميع قواعد البيانات المُخزَّنة:

```
hsoubguide=# \l
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
hsoubguide	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	
postgres	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	
template0	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +
					postgres=CTc/postgres
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +
					postgres=CTc/postgres

3. 1. 2. استعراض جميع قواعد البيانات مع معلومات إضافية

بإضافة الرمز `+` إلى الأمر `\l` يمكن عرض قائمة لجميع قواعد البيانات المخزنة مع معلومات إضافية عن كل منها.

```
hsoubguide=# \l+
```

Name	Owner	Encoding	Collate	Ctype	Access privileges	Size	Tablespace	Description
hsoubguide	postgres	UTF8	en_ GB.UTF-8	en_ GB.UTF-8		8273 kB	pg_ default	
postgres	postgres	UTF8	en_ GB.UTF-8	en_ GB.UTF-8		8345 kB	pg_ default	default administrative connection datab
ase								
template0	postgres	UTF8	en_ GB.UTF-8	en_ GB.UTF-8	=c/postgres +	8049 kB	pg_ default	unmodifiable empty database

Name	Owner	Encoding	Collate	Ctype	Access privileges	Size	Tablespace	Description
					postgres=CTc/postgres			
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +	8049 kB	pg_default	default template for new databases
					postgres=CTc/postgres			

3. 1. 3. استعراض الجداول في قاعدة البيانات

لاستعراض قائمة بالجداول الموجودة في قاعدة البيانات، نستخدم الأمر `\dt` كما يلي:

```
hsoubguide=# \dt
```

Schema	Name	Type	Owner
public	basket_a	table	postgres
public	basket_b	table	postgres
public	departments	table	postgres
public	employee_departments	table	postgres
public	employees	table	postgres
public	marks	table	postgres
public	names	table	postgres
public	phones	table	postgres
public	products	table	postgres
public	purchase_items	table	postgres
public	purchases	table	postgres
public	size_calc	table	postgres
public	student	table	postgres
public	table1	table	postgres
public	table2	table	postgres
public	test_explain	table	postgres
public	test_table	table	postgres
public	users	table	postgres
public	users2	table	postgres

3. 1. 4. وصف جدول

لوصف جدول، نستخدم الأمر `\d` مع اسم الجدول كما يلي:

```
hsoubguide=# \d employees
```

Column	Type	Collation	Nullable	Default
id	integer		not null	
last_name	character varying(55)			
salary	integer			

```
Indexes:
    "employees_pkey" PRIMARY KEY, btree (id)
```

3. 1. 5. استعراض الجداول مع بعض المعلومات الإضافية

بإضافة الرمز `+` إلى أي أمر من أوامر الاستعراض، فإنها تضيف إلى المخرجات المزيد من المعلومات، فعند إضافة الرمز `+` إلى الأمر `\dt` يمكننا استعراض المزيد من المعلومات عن الجداول الموجودة في قاعدة البيانات:

```
hsoubguide=# \dt+
```

Schema	Name	Type	Owner	Size	Description
public	basket_a	table	postgres	8192 bytes	
public	basket_b	table	postgres	8192 bytes	
public	departments	table	postgres	8192 bytes	
public	employee_ departments	table	postgres	8192 bytes	
public	employees	table	postgres	8192 bytes	
public	marks	table	postgres	8192 bytes	
public	names	table	postgres	8192 bytes	
public	phones	table	postgres	8192 bytes	
public	products	table	postgres	16 kB	
public	purchase_items	table	postgres	304 kB	

Schema	Name	Type	Owner	Size	Description
public	purchases	table	postgres	96 kB	
public	size_calc	table	postgres	0 bytes	
public	student	table	postgres	16 kB	
public	table1	table	postgres	8192 bytes	
public	table2	table	postgres	8192 bytes	
public	test_explain	table	postgres	8192 bytes	
public	test_table	table	postgres	8192 bytes	
public	users	table	postgres	16 kB	
public	users2	table	postgres	8192 bytes	

3. 1. 6. وصف جدول مع معلومات إضافية

للحصول على المزيد من المعلومات عن جدول ما، يمكننا استخدام الرمز + مع الأمر \d وبعدها اسم الجدول:

```
hsoubguide=# \d+ users
```

Column	Type	Collation	Nullable	Default	Storage	Stats target	Description
id	integer		not null	nextval('users_id_seq'::regclass)	plain		
email	character varying(255)				extended		
password	character varying(255)				extended		
details	hstore				extended		
created_at	timestamp with time zone				plain		
deleted_at	timestamp with time zone				plain		

```
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "purchases" CONSTRAINT "purchases_user_id_fkey" FOREIGN KEY
(user_id) REFERENCES users(id)
Access method: heap
```

3. 1. 7. استعراض المستخدمين وخصائصهم

بتنفيذ الأمر `\dg` يمكننا استعراض قائمة الأدوار، وهي قائمة المستخدمين وخصائص كل منهم، كما يلي:

```
hsoubguide=# \dg
```

Role name	Attributes	Member of
mostafa		{}
postgres	Superuser, Create role, Create DB, Replication, Bypass RLS	{}

3. 2. أوامر أخرى أساسية

3. 2. 1. تحرير الاستعلامات في محرر النصوص

يمكن فتح مخزن الاستعلامات ضمن محرر النصوص الافتراضي (مثل vi أو nano) داخل صدَفة `psql` باستخدام الأمر `\e` وهو مفيد لكتابة الاستعلامات الطويلة وتحريرها قبل تنفيذها، وعند فتحه سنجده يحوي آخر استعلام تمت كتابته، مما يسمح لنا بتعديله لإعادة تنفيذه.

كما يمكن استعراض آخر محتويات هذا المخزن باستخدام الأمر `\p`.

```
hsoubguide=# SELECT *
hsoubguide-# FROM products
hsoubguide-# LIMIT
hsoubguide-# 1
hsoubguide-# ;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}

```
hsoubguide=# \p
SELECT *
FROM products
LIMIT
1
;
```

3. 2. 2. تشغيل توقيت الاستعلام

لا يكون توقيت تنفيذ الاستعلام مُتاحًا للعرض في الحالة الافتراضية، ولكن يمكننا تفعيله من خلال الأمر التالي:

```
hsoubguide=# SELECT * FROM products LIMIT 1;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}

```
Time: 0.723 ms
```

حيث سيتيح ذلك الأمر إظهار توقيت الاستعلام بالميلي ثانية.

3. 2. 3. الحصول على مساعدة بخصوص تعليمات SQL

يمكن استخدام الأمر \h يليه اسم التعليمة في SQL لعرض التوثيق الخاص بهذه التعليمة:

```
hsoubguide=# \h VACUUM

Command:     VACUUM
Description: garbage-collect and optionally analyze a database
Syntax:
VACUUM [ ( option [, ...] ) ] [ table_and_columns [, ...] ]
VACUUM [ FULL ] [ FREEZE ] [ VERBOSE ] [ ANALYZE ] [ table_and_
columns [, ...] ]
```

```
where option can be one of:

    FULL [ boolean ]
    FREEZE [ boolean ]
    VERBOSE [ boolean ]
    ANALYZE [ boolean ]
    DISABLE_PAGE_SKIPPING [ boolean ]
    SKIP_LOCKED [ boolean ]
    INDEX_CLEANUP [ boolean ]
    TRUNCATE [ boolean ]

and table_and_columns is:

    table_name [ ( column_name [, ...] ) ]

URL: https://www.postgresql.org/docs/12/sql-vacuum.html
```

3. 2. 4. الخروج من صَدَفة postgres

قد تقضي وقتًا طويلًا داخل صدفة psql، ليس حبًا بها، ولكن لعدم معرفة كيفية الخروج منها، لذلك لا تنسَ أن الأمر \q هو الذي يُخرجك من صدفة psql.

```
hsoubguide=# \q

bash-4.2$
```

3. 3. خلاصة الفصل

تعرفنا في هذا الفصل على الأوامر الأكثر استخدامًا في صدفة psql، منها استعراض الجداول وقواعد البيانات، ولكن لا يزال هناك العديد من الأوامر الأخرى التي يمكنك استكشافها من خلال الأمر \?.

4. استخدام أساسيات SQL في Postgres

يتناول هذا الفصل الطرق الأساسية للاستعلام عن البيانات وعرض الجداول، وربط البيانات بين عدة جداول، كما سنتعرف فيه على كيفية تعديل البيانات ضمن الجدول أو حذفها، والعديد من التعليمات والتوابع الأكثر أهمية في Postgres.

4. 1. إنشاء الجداول

سنشرح في هذه الفقرة كيفية إنشاء الجداول المذكورة في المثال السابق يدويًا.

> إن كنت نزّلت قاعدة البيانات من الفصل السابق، فيمكنك الاطلاع على التعليمات الحقيقية التي تم من خلالها إنشاء الجداول، وذلك من خلال الأوامر التالية في صدفة bash:

```
pg_dump -st products hsoubguide
pg_dump -st purchases hsoubguide
pg_dump -st purchase_items hsoubguide
pg_dump -st users hsoubguide
```

تقوم التعليمة السابقة بإظهار جميع التعليمات اللازمة لإنشاء الجدول المذكور بالحالة التي هو عليها الآن، إلا أننا سنركّز في هذا الفصل على التعليمات الأساسية فقط.

في حال قمت بتنزيل النسخة الكاملة من قاعدة البيانات كما ورد سابقًا، وترغب في إنشاء الجداول من الصفر أيضًا، فيمكن ذلك بإنشاء قاعدة بيانات جديدة كي لا تتعارض مع الجداول السابقة.

```
CREATE DATABASE hsoubguide2;
```

يمكننا إنشاء جدول جديد في قاعدة البيانات عن طريق التعليمة CREATE TABLE حيث نقوم بتحديد اسم الجدول، وتفصيل أسماء الأعمدة ونوع البيانات في كل منها، إلى جانب الصفات الخاصة بكل عمود، كما في الأمثلة التالية التي تشرح نفسها بنفسها:

```
CREATE TABLE products (
    id integer NOT NULL,
    title character varying(255),
    price numeric,
    created_at timestamp with time zone,
    deleted_at timestamp with time zone,
    tags character varying(255)[]
);
```

في التعليمة السابقة، أُنشئ جدول اسمه products وحُدّد فيه الأعمدة التالية:

- العمود id، يحتوي بيانات نوعها integer أي أعداد صحيحة، وله المواصفة NOT NULL، أي أنه يُمنَع من إنشاء سطر دون تحديد قيمة لهذا العمود.
- العمود title، يحتوي بيانات محرفية (نصية)، طولها متغير، يمكن أن يصل إلى 255 حرفًا، ولذلك نستخدم النوع character varying مع وضع الطول الأكبر بين قوسين.
- العمود price، يحوي بيانات عددية من النوع numeric وهو نوع عددي مميز، يمكن تخزين أعداد يصل عدد منازلها إلى 131072 منزلة قبل الفاصلة العشرية، و 16383 منزلة بعد الفاصلة! إلا أن العمليات الحسابية عليه تكون أبطأ من النوع integer والسعة التخزينية التي يحتاج إليها أكبر أيضًا.
- العمود created_at و العمود deleted_at يحتويان على بيانات التاريخ والوقت مع حفظ المنطقة الزمنية، فهما من النوع timestamp with time zone.
- العمود tags يحتوي بيانات محرفية من النوع character varying إلا أن وجود الرمز [] في تعريف هذا العمود، يعني أنه سيحتفظ بمصفوفة من العبارات المحرفية (أي أنه لن يحتوي عبارة واحدة، بل قائمة من العبارات)

يمكنك الاطلاع على المزيد من التفاصيل عن الأنواع العددية من توثيق Postgres الرسمي.

يمكننا رؤية تفاصيل الجدول الذي أُنشئ كما يلي:

```
hsoubguide2=# \d products
```

Column	Type	Collation	Nullable	Default
id	integer		not null	
title	character varying(255)			
price	numeric			
created_at	timestamp with time zone			
deleted_at	timestamp with time zone			
tags	character varying(255)[]			

بعد أن قُمنا بإنشاء الجدول وتحديد أنواع الأعمدة، سنقوم بتحديد مواصفة مهمة للعمود id هي `PRIMARY KEY` وتعني أن القيمة المخزنة ضمن هذا العمود ستكون فريدة `UNIQUE` لا تتكرر في أي سطر من الأسطر، ولا يمكن أن تكون فارغة `NULL`، وستكون القيمة المخزنة في هذا العمود هي التي تميّز الأسطر عن بعضها.

سنتعرف من خلال هذه الخطوة على التعليمة التي تقوم بتعديل تعريف الجداول `ALTER TABLE`، والتعديل الذي سنقوم به سيكون `ADD CONSTRAINT` كما يلي:

```
ALTER TABLE ONLY public.products
    ADD CONSTRAINT products_pkey PRIMARY KEY (id);
```

حيث قمنا بإضافة `CONSTRAINT` أسميناه `products_pkey` ينصّ على جعل العمود id بالمواصفة `PRIMARY KEY`، وذلك في الجدول products.

يمكننا استخدام التعليمة `ALTER` لتغيير أي شيء من خصائص الجدول، كتغيير اسمه، أو تغيير اسم أحد الأعمدة، أو إضافة أعمدة جديدة أو حذف أعمدة، ويمكنك الاطلاع على المزيد عن هذه التعلمية من توثيق Postgres الرسمي.

أصبح لديك الآن المعرفة الكافية لفهم كيفية إنشاء الجداول، ويمكنك قراءة التعليمات التالية وتنفيذها بوضوح لإنشاء الجداول التي نحتاجها في أمثلتنا.

```
CREATE TABLE purchases (
    id integer NOT NULL,
    created_at timestamp with time zone,
    name character varying(255),
    address character varying(255),
    state character varying(2),
    zipcode integer,
    user_id integer
);

CREATE TABLE purchase_items (
    id integer NOT NULL,
    purchase_id integer,
    product_id integer,
    price numeric,
    quantity integer,
    state character varying(255)
);

CREATE TABLE users (
    id integer NOT NULL PRIMARY KEY,
    email character varying(255),
    password character varying(255),
    details public.hstore,
    created_at timestamp with time zone,
    deleted_at timestamp with time zone
);
```

في الجدول الأخير، يوجد العمود details يحوي بيانات من النوع hstore، وسنتحدث عن هذا النوع الخاص في الفصل التالي، أنواع بيانات خاصة في قواعد بيانات Postgres.

والآن، إذا أردت حذف جدولٍ من الجداول، فاستخدم التعليمة `DROP TABLE` كما يلي:

```
DROP TABLE users
```

4. 2. تعديل الجداول

يمكننا التعديل على الجدول بعد إنشائه باستخدام التعليمة الأساسية ALTER، حيث يمكننا من خلالها القيام بتعديل جميع خصائص الجداول، نذكر منها أهم عمليات التعديل التالية:

- تعديل اسم الجدول:

```
ALTER TABLE users
     RENAME TO our_lovely_users;
```

- تعديل اسم عمود ما:

```
ALTER TABLE products
  RENAME COLUMN title TO product_name;
```

- إضافة عمود إلى الجدول:

```
ALTER TABLE users
  ADD COLUMN image_url character varying(1024);
```

- حذف عمود من الجدول:

```
ALTER TABLE users
  DROP COLUMN IF EXISTS image_url
```

- تعديل نوع البيانات في عمود ما من الجدول:

```
ALTER TABLE products
  ALTER COLUMN price SET DATA TYPE float;
```

يمكنك التوسع في إمكانية التعديل على الجداول بالاطلاع على توثيق Postgres الرسمي.

4. 3. إدخال البيانات

بعد أن قمنا بإنشاء الجداول، علينا إدخال البيانات فيها، ويتم ذلك عن طريق التعليمة INSERT INTO ، وسنقوم في المثال بإدخال سطر في الجدول products كما يلي:

```
INSERT INTO products(
id,
title,
price,
created_at,
tags
)
VALUES (
  1,
'Python Book',
29.99,
NOW(),
ARRAY['Book','Programming','Python']
);
```

توضّح التعليمة السابقة كيفية إدخال سطر في جدول ما، وذلك عن طريق ذكر اسم الجدول، ثم ترتيب الأعمدة التي سنحدد قيمتها بين قوسين، وبعد ذلك نكتب الكلمة المفتاحية VALUES ثم نذكر قيمة كل عمود في السطر الجديد بالترتيب نفسه.

لاحظ استخدامنا للتابع NOW الذي يعطي التاريخ والوقت في لحظة تنفيذ التعليمة. لاحظ أيضًا كيف قمنا بإدخال مصفوفة في العمود tags، وسنتحدث عن ذلك أكثر لاحقًا في هذا الكتاب.

للتأكد من أن السطر تم إدخاله، يمكنك استخدام التعليمة SELECT وهي تعليمة الاستعلام الأساسية، وسنستخدمها هنا دون أي شروط كما يلي:

```
SELECT * FROM products;
```

حيث تشير * إلى اختيار جميع الأعمدة لعرضها من الجدول products، وسنحصل على المخرجات التالية:

```
hsoubguide2=# SELECT * FROM products;
```

id	title	price	created_at	deleted_at	tags
1	Python Book	29.99	2020-06-22 12:22:02.281079+03		{Book,Programming,Python}

يمكننا أيضًا إدخال عدة أسطر في تعليمة INSERT واحدة، وذلك بوضع قوسين () حول البيانات الخاصة بكل سطر، وفصل الأسطر بفاصلة كما يلي:

```
INSERT INTO products(id,title,price,created_at,tags)
VALUES ( 2,'Book2',1.99,NOW(),ARRAY['a1','b','q'] ),
       ( 3,'Book3',2.99,NOW(),ARRAY['a2','c','w'] ),
       ( 4,'Book4',3.99,NOW(),ARRAY['a3','d','e'] ),
       ( 5,'Book5',4.99,NOW(),ARRAY['a4','e','r'] );
```

> سنتحدث عن كيفية التعديل على الأسطر المخزّنة أو حذف أسطر من الجدول في فقرات لاحقة.

كانت هذه بداية سريعة في كيفية إنشاء جدول وإدخال البيانات إليه، ثم الاستعلام عنها، وسنتوسع في الفقرات التالية في أنواع الاستعلامات وكيفية القيام بها، وذلك على اعتبار وجود الجداول والبيانات مُسبقًا، واعتبار أن إدخال البيانات يتم تلقائيًا عن طريق ربط قاعدة البيانات بموقع ويب، أو بواجهة مستخدم ما.

4.4. استعراض الجداول

نستعرض في المثال التالي كيفية عرض جميع الجداول المخزنة في قاعدة البيانات باستخدام التعليمة \dt كما يلي:

```
hsoubguide=# \dt
```

Schema	Name	Type	Owner
public	products	table	postgres
public	purchase_items	table	postgres

Schema	Name	Type	Owner
public	purchases	table	postgres
public	users	table	postgres

العبارة hsoubguide=# مولّدة تلقائيًا من psql وتحوي اسم قاعدة البيانات التي نحن بداخلها، لذلك لا تنسخها في حال كنت ترغب بنسخ التعليمات مباشرةً من هذا الكتاب.

4. 5. الاستعلام عن البيانات

كلمة "استعلام" تعني طلب الحصول على أسطر محددة من قاعدة البيانات، ولكننا قد نرغب في تحديد أعمدة محددة من جداول محددة، وقد نحتاج إلى تحديد شروط على الأسطر التي نرغب بالحصول عليها، كأن نطلب الحصول على المستخدمين الذين لم يقوموا بأي تفاعل منذ سنة، أو الحصول على بيانات عن المنتجات التي يتجاوز عدد مبيعاتها قيمة محددة. ولذلك فالخطوة الأولى في بناء الاستعلامات هي معرفة مصدر البيانات التي تتعامل معها، ثم معرفة البيانات التي يمكننا الحصول عليها، وهي الأعمدة التي يحتويها الجدول.

في مثالنا التالي، سنحاول معرفة بعض المعلومات عن المستخدمين المخزّنين في قاعدة البيانات.

سنستخدم التعليمة \d يليها اسم الجدول لاستعراض أسماء الأعمدة ضمن الجدول كما يلي:

```
hsoubguide=# \d users
```

Column	Type	Collation	Nullable	Default
id	integer		not null	nextval('users_id_seq'::regclass)
email	character varying(255)			
password	character varying(255)			

Column	Type	Collation	Nullable	Default
details	hstore			
created_at	timestamp with time zone			
deleted_at	timestamp with time zone			

```
Indexes:
    "users_pkey" PRIMARY KEY, btree (id)
Referenced by:
    TABLE "purchases" CONSTRAINT "purchases_user_id_fkey" FOREIGN KEY
(user_id) REFERENCES users(id)
```

ظهر لدينا الآن مجموعة متنوعة من البيانات، سنختار منها المعلومات الثلاث التالية الخاصة بالمستخدمين وهي `id` و `email` و `created_at`.

والآن أصبحنا نعرف المعلومات الأساسية لبناء الاستعلام وهي:

- الجدول الذي نريد استعلام البيانات منه
- البيانات التي نريدها من ذاك الجدول

يبين المثال التالي التركيب النحوي للقيام بالاستعلام المطلوب، وهو يحوي البيانات المُراد الحصول عليها، يليها اسم الجدول الحاوي عليها، ثم فاصلة منقوطة للدلالة على انتهاء الاستعلام:

```
hsoubguide=# SELECT id,email,created_at
hsoubguide-# FROM users;
```

id	email	created_at
1	Earlean.Bonacci@yahoo.com	2009-12-20 22:36:00+02
2	Evelyn.Patnode@gmail.com	2010-11-12 23:27:00+02
3	Derek.Crenshaw@gmail.com	2009-03-08 05:06:00+02
4	Shari.Julian@yahoo.com	2010-11-20 12:58:00+02
5	Zita.Breeding@gmail.com	2009-08-12 01:33:00+03
6	Samatha.Hedgpeth@yahoo.com	2010-07-18 13:40:00+03
7	Quinton.Gilpatrick@yahoo.com	2010-09-03 00:56:00+03
8	Vivian.Westmoreland@yahoo.com	2009-10-01 14:34:00+03

id	email	created_at
9	Danny.Crays@gmail.com	2009-04-22 10:30:00+03
10	Edmund.Roles@yahoo.com	2009-07-08 00:01:00+03
11	Shanell.Lichtenstein@aol.com	2009-05-22 03:18:00+03
12	Romaine.Birdsell@aol.com	2009-01-14 07:07:00+02
13	Zita.Luman@yahoo.com	2009-02-04 16:49:00+02
14	Claud.Cousineau@gmail.com	2009-08-17 21:48:00+03
15	Kali.Damore@yahoo.com	2010-07-07 13:28:00+03
16	Graciela.Kubala@yahoo.com	2010-08-19 08:42:00+03
17	Theresia.Edwin@yahoo.com	2010-08-11 11:21:00+03
18	Ozella.Yoshimura@gmail.com	2010-07-23 19:03:00+03
19	Wynona.Greening@aol.com	2009-05-24 17:25:00+03
20	Kimi.Mcqueeney@gmail.com	2010-06-22 18:16:00+03
21	Cherryl.Tarnowski@gmail.com	2009-01-26 11:56:00+02
22	Isabel.Breeding@gmail.com	2010-07-11 16:28:00+03
23	Ivana.Kurth@yahoo.com	2010-06-25 11:36:00+03
24	Humberto.Jonson@yahoo.com	2009-09-23 16:09:00+03
25	Ivana.Sosnowski@aol.com	2009-01-16 13:55:00+02
26	Cortney.Strayer@gmail.com	2009-07-19 09:08:00+03
27	Williams.Upson@gmail.com	2010-08-10 08:48:00+03
28	Jeremiah.Buonocore@yahoo.com	2009-03-19 09:49:00+02
29	Ozella.Roles@gmail.com	2009-10-09 12:44:00+03
30	Salvatore.Arends@aol.com	2009-09-05 04:55:00+03
31	Layne.Sarver@aol.com	2010-09-26 11:00:00+03
32	Takako.Gilpatrick@aol.com	2009-02-22 17:46:00+02
33	Russ.Mcclain@yahoo.com	2010-01-12 19:27:00+02
34	Claud.Westmoreland@aol.com	2010-06-11 20:21:00+03
35	Derek.Knittel@gmail.com	2010-08-17 00:09:00+03
36	Eleanor.Patnode@yahoo.com	2010-06-06 04:27:00+03
37	Carmel.Bulfer@aol.com	2009-06-06 23:13:00+03
38	Mauro.Pung@yahoo.com	2009-08-20 05:34:00+03

id	email	created_at
39	Sherilyn.Hamill@gmail.com	2010-04-02 02:39:00+03
40	Glen.Lanphear@yahoo.com	2010-08-06 18:14:00+03
41	Stacia.Schrack@aol.com	2010-06-14 22:28:00+03
42	Tonette.Alba@gmail.com	2009-12-28 12:21:00+02
43	Eve.Kump@yahoo.com	2009-08-20 12:45:00+03
44	Shanell.Maxson@gmail.com	2009-11-21 08:28:00+02
45	Gudrun.Arends@gmail.com	2010-06-30 15:30:00+03
46	Angel.Lessley@yahoo.com	2009-08-21 20:06:00+03
47	Harrison.Puett@yahoo.com	2009-07-21 18:20:00+03
48	Granville.Hedgpeth@gmail.com	2009-08-03 17:54:00+03
49	Samatha.Pellegrin@yahoo.com	2009-03-25 22:17:00+02
50	Wan.Dilks@gmail.com	2009-10-09 01:43:00+03

> يجب إنهاء تعليمات postgres بفاصلة منقوطة، وفي حال عدم القيام بذلك، فعند الضغط على enter لا تعتبر psql أن التعليمة قد انتهت، وتنتظر كتابة الفاصلة المنقوطة في السطر الجديد.

يسمح لنا الأمر السابق بعرض جميع بيانات الجدول، إلا أن هذا لا يناسب عرض الجداول الحاوية على كمية كبيرة من البيانات، ولذلك يتوجب علينا الحد من كمية البيانات وتصفيتها.

4. 5. 1. الحد من البيانات المعروضة باستخدام LIMIT

يمكننا إضافة التوجيه `LIMIT` يليه عدد الأسطر الأكبر الذي نسمح بعرضه عند الاستعلام، وذلك للحد من الأسطر المعروضة، كما يلي:

```
hsoubguide=# SELECT id,email,created_at
hsoubguide-# FROM users
hsoubguide-# LIMIT 5;
```

id	email	created_at
1	Earlean.Bonacci@yahoo.com	2009-12-20 22:36:00+02
2	Evelyn.Patnode@gmail.com	2010-11-12 23:27:00+02

id	email	created_at
3	Derek.Crenshaw@gmail.com	2009-03-08 05:06:00+02
4	Shari.Julian@yahoo.com	2010-11-20 12:58:00+02
5	Zita.Breeding@gmail.com	2009-08-12 01:33:00+03

4. 5. 2. ترتيب البيانات باستخدام ORDER BY

بعد أن تعرفنا على كيفية الحد من البيانات، قد نرغب بإظهار عدد محدود من البيانات ولكن وفق ترتيب معيّن، فمثلًا نريد الاستعلام عن 5 مستخدمين ولكن بعد ترتيبهم حسب البريد الإلكتروني الخاص بهم، وللقيام بذلك سنستخدم التوجيه ORDER BY كما يلي:

```
hsoubguide=# SELECT id,email,created_at
hsoubguide-# FROM users
hsoubguide-# ORDER BY email
hsoubguide-# LIMIT 5;
```

id	email	created_at
46	Angel.Lessley@yahoo.com	2009-08-21 20:06:00+03
37	Carmel.Bulfer@aol.com	2009-06-06 23:13:00+03
21	Cherryl.Tarnowski@gmail.com	2009-01-26 11:56:00+02
14	Claud.Cousineau@gmail.com	2009-08-17 21:48:00+03
34	Claud.Westmoreland@aol.com	2010-06-11 20:21:00+03

ماذا لو أردنا القيام بترتيب تنازلي؟ يمكننا استخدام DESC اختصارًا لكلمة "descending" كما يلي:

```
hsoubguide=# SELECT id,email,created_at
hsoubguide-# FROM users
hsoubguide-# ORDER BY email DESC
hsoubguide-# LIMIT 5;
```

id	email	created_at
13	Zita.Luman@yahoo.com	2009-02-04 16:49:00+02
5	Zita.Breeding@gmail.com	2009-08-12 01:33:00+03

id	email	created_at
19	Wynona.Greening@aol.com	2009-05-24 17:25:00+03
27	Williams.Upson@gmail.com	2010-08-10 08:48:00+03
50	Wan.Dilks@gmail.com	2009-10-09 01:43:00+03

فلنلقِ نظرةً على جدول المنتجات كي نحظى بفرصة الحصول على مثالٍ جديد:

```
hsoubguide=# SELECT title,price
hsoubguide-# FROM products
hsoubguide-# LIMIT 8;
```

title	price
Dictionary	9.99
Python Book	29.99
Ruby Book	27.99
Baby Book	7.99
Coloring Book	5.99
Desktop Computer	499.99
Laptop Computer	899.99
MP3 Player	108.00

سنرتب في المثال التالي الجدول السابق حسب السعر، ثمّ سنرتب المنتجات المتشابهة بالسعر بعكس الترتيب الأبجدي لأسماء المنتجات، وذلك كما يلي:

```
hsoubguide=# SELECT title,price
hsoubguide-# FROM products
hsoubguide-# ORDER BY price ASC, title DESC
hsoubguide-# LIMIT 10;
```

title	price
Coloring Book	5.99
Baby Book	7.99
Pop CD	9.99
Holiday CD	9.99

title	price
Electronic CD	9.99
Dictionary	9.99
Country CD	9.99
Classical CD	9.99
Romantic	14.99
Drama	14.99

4. 5. 3. عمليات التجميع (Aggregation Functions)

سنتعرف في هذه الفقرة على توابع يمكنها القيام بتجميع قيم الأعمدة في قيمة واحدة، فمثلًا يمكننا الحصول على متوسط أسعار المنتجات في جدول المنتجات، أو عدد المنتجات الموجودة، أو السعر الأكبر أو الأصغر، وذلك كما يلي:

```
hsoubguide=# SELECT MAX(price),MIN(price),AVG(price),SUM(price),COUNT
(price)
hsoubguide-# FROM products;
```

max	min	avg	sum	count
899.99	5.99	132.0390476190476190	2772.82	21

لاحظ كيف اختلفت المخرجات فلم يعد هناك أسماء الأعمدة المعتادة، بل استُبدلت بأسماء التوابع المستخدمة.

تقوم عملية التجميع COUNT بإرجاع عدد الأسطر التي تحتوي قيمة غير خالية، وسيوضّح المثال التالي ما نرمي إليه:

```
hsoubguide=# SELECT COUNT(*),
hsoubguide-# COUNT(email),
hsoubguide-# COUNT(details)
hsoubguide-# FROM users;
```

count	count	count
50	50	34

لاحظ كيف أرجع استدعاء العملية COUNT على العمود details القيمة 34، رغم أنها أرجعت القيمة 50 للعمود email، وقد استخدمنا كذلك COUNT(*) لإظهار عدد جميع الأسطر في الجدول بغض النظر عن قيم الأعمدة.

ولكن هل لاحظت أسماء الأعمدة في المخرجات؟ إنها جميعًا count، ولا يبدو ذلك مفيدًا لمن سيطّلع على هذه المخرجات، ولذلك سنستخدم التوجيه AS.

للمزيد عن عمليات التجميع، يمكنك الرجوع إلى توثيق Postgres الرسمي.

4. 5. 4. التوجيه AS

يسمح لنا هذا التوجيه بإعادة تسمية أعمدة جدول الخرج، وذلك ليتناسب مع الهدف الذي قُمنا بالاستعلام لأجله.

رأينا في المثال السابق كيف قمنا بثلاث عمليات COUNT مختلفة، ولكن الجدول في المخرجات كانت جميع أعمدته بالاسم count دون تمييز، ولذلك سنستخدم التوجيه AS لتغيير أسماء هذه الأعمدة كما يلي:

```
hsoubguide=# SELECT COUNT(*) AS TOTAL_COUNT,
hsoubguide-# COUNT(email) AS EMAIL_COUNT,
hsoubguide-# COUNT(details) AS DETAILS_COUNT
hsoubguide-# FROM users;
```

total_count	email_count	details_count
50	50	34

الآن أصبحت المخرجات مقروءة ومفهومة أكثر، ولكن تجدر بنا الإشارة إلى أن ذكر التوجيه AS هو أمر اختياريّ، فيمكننا تسمية جداول الخرج بدون الحاجة إلى كتابة التوجيه AS كما يلي:

```
hsoubguide=# SELECT COUNT(*) TOTAL_COUNT,
hsoubguide-# COUNT(email) EMAIL_COUNT,
hsoubguide-# COUNT(details) DETAILS_COUNT
hsoubguide-# FROM users;
```

total_count	email_count	details_count
50	50	34

حافظ دومًا على مقروئية الجدول في المخرجات، فقد يكون لديك في العمود عمليات حسابية أو عدة توابع في عبارة واحدة، وهنا سيكون لزامًا استخدام الأسماء المستعارة للأعمدة وإلا ستبدو الجداول في المخرجات غير مفهومة.

4. 5. 5. ترشيح البيانات (Filtering)

استعلمنا حتى الآن عن جميع البيانات في الجدول، وأكثر ما استطعنا تحديده هو تقليل عدد الأسطر في المخرجات، وترتيبها بشرط محدد، ولكننا ماذا لو أردنا استعراض جميع الأسطر التي تحقق شرطًا ما؟ كيف يمكننا استعراض المنتجات التي يتجاوز سعرها قيمة محددة؟ أو المستخدمين الذين أنشئت حساباتهم بعد تاريخ محدد؟

يمكننا ترشيح البيانات باستخدام الشرط WHERE، ويبين المثال التالي عملية ترشيح للحصول على المستخدمين الذين أُنشئ حسابهم في تاريخ 30/6/2010 فصاعدًا:

```
SELECT email, created_at
FROM users
WHERE created_at >= '2010-06-30';
```

كما يمكننا دمج شروط أخرى باستخدام AND أو OR، ويبين المثال التالي استخدام AND للاستعلام عن جميع المستخدمين الذين قاموا بإنشاء حسابات في شهر تموز عام 2010:

```
hsoubguide=# SELECT email, created_at
hsoubguide-# FROM users
hsoubguide-# WHERE created_at >= '2010-07-01'
hsoubguide-#   AND created_at < '2010-08-01';
```

email	created_at
Samatha.Hedgpeth@yahoo.com	2010-07-18 13:40:00+03
Kali.Damore@yahoo.com	2010-07-07 13:28:00+03
Ozella.Yoshimura@gmail.com	2010-07-23 19:03:00+03
Isabel.Breeding@gmail.com	2010-07-11 16:28:00+03

لاحظ كيف يتغيّر محثّ الأوامر من hsoubguide=# إلى hsoubguide-# عندما تتوزع التعليمة على عدة أسطر، وتكون الفاصلة المنقوطة هي نهاية التعليمة.

والآن ما رأيك باستخدام بعض أوامر التجميع مثل SUM و AVG وغيرها للحصول على بعض المعلومات المفيدة!

سنبدأ بالاطلاع على جدول المنتجات المشتراة، لنرى ماذا بإمكاننا أن نستخلص منه من المعلومات:

```
hsoubguide=# SELECT *
hsoubguide-# FROM purchase_items
hsoubguide-# LIMIT 20;
```

id	purchase_id	product_id	price	quantity	state
2	1	3	27.99	1	Delivered
3	1	8	108.00	1	Delivered
4	2	1	9.99	2	Delivered
5	3	12	9.99	1	Delivered
6	3	17	14.99	4	Delivered
7	3	11	9.99	1	Delivered
8	4	4	7.99	3	Delivered
9	5	18	14.99	1	Delivered
10	5	2	29.99	4	Delivered
11	6	5	5.99	1	Delivered
12	7	6	499.99	3	Returned
13	8	10	529.00	1	Delivered
14	8	7	899.99	1	Delivered
15	9	15	9.99	2	Delivered
16	10	2	29.99	1	Delivered
17	11	9	499.00	2	Delivered
18	12	14	9.99	5	Delivered
19	12	10	529.00	1	Delivered
20	13	8	108.00	1	Delivered
21	14	20	14.99	1	Delivered

هـذا الجـدول فيـه الكثيـر مـن المعلومـات المهمـة، وسـنعود إليـه فـي الفقـرة التاليـة، ولكـن الآن يمكننـا رؤيـة أن بعـض عمليـات الشـراء يتـم إيصالها "Delivered" وبعضها تُرجـع للمتجـر "Returned"، وسـنقوم فـي المثـال التالـي بإحصائها، وإيجـاد مجمـوع المشـتريات التـي تـم إيصالها بالفعـل.

```
hsoubguide=# SELECT COUNT(*)
hsoubguide-# FROM purchase_items
hsoubguide-# WHERE state='Delivered';
```

count
3888

```
hsoubguide=# SELECT COUNT(*)
hsoubguide-# FROM purchase_items
hsoubguide-# WHERE state='Returned';
```

count
246

```
hsoubguide=# SELECT COUNT(*)
hsoubguide-# FROM purchase_items;
```

count
4371

سنوجد مجموع المشتريات التي سُلِّمَت بالفعل، وتلك التي أُعيدَت:

```
hsoubguide=# SELECT SUM(price)
hsoubguide-# FROM purchase_items
hsoubguide-# WHERE state='Delivered';
```

sum
517787.85

```
hsoubguide=# SELECT SUM(price)
hsoubguide-# FROM purchase_items
hsoubguide-# WHERE state='Returned';
```

sum
36456.87

4. 5. 6. استخدام عبارة CASE الشرطية

تسمح لنا عبارة CASE بعمل بنية مشابهة للبنية البرمجية switch الموجودة في معظم لغات البرمجة، أو يمكن تشبيهها أكثر بعبارات if متتابعة، حيث يمكننا إنتاج قِيَم محددة بناءً على اختبارات على قيم أخرى، ولا بد أنه المثال سيوضّح الفصل.

ننشئ جدولًا بسيطًا فيه عمود واحد يحوي أعدادًا صحيحة.

```
hsoubguide=# CREATE TABLE case_example(
hsoubguide(#      number INT
hsoubguide(# );

CREATE TABLE
```

وسندخل 5 أعداد في هذا الجدول كما يلي:

```
hsoubguide=# INSERT INTO case_example VALUES(1),(2),(3),(4),(5),(6);
INSERT 0 6
hsoubguide=# SELECT * FROM case_example ;
```

number
1
2
3
4
5
6

والآن سنستعلم عن عناصر هذا العمود، ولكن بدلًا من طباعة العنصر كما هو، سنطبع عبارات حسوبية بناءً على قيمة هذا العمود.

```
hsoubguide=# SELECT (CASE
    WHEN number=1 THEN 'Hsoub'
    WHEN number=2 OR number=3 THEN 'Khamsat'
    WHEN number<5 THEN 'Mostaql'
    WHEN number>=5 AND number<=6 THEN 'ANA'
    WHEN number=7 THEN 'IO'
    ELSE '.com'
    END) FROM case_example ;
```

case
Hsoub
Khamsat
Khamsat
Mostaql
ANA
ANA

4. 6. التعديل UPDATE والحذف DELETE

بعد أن تعرَّفنا على كيفية ترشيح البيانات في الاستعلامات، سنستخدم تعليمة الترشيح نفسها WHERE لتحديد الأسطر التي نرغب بالتعديل على محتوياتها أو حذفها.

> نعدِّل في هذه الفقرة على محتويات الجدول من البيانات، أو حذف سطر ما من أسطر الجدول، أما لو أردت التعديل على الجدول نفسه بإضافة عمود أو حذف عمود، أو التعديل على اسم عمود ما أو نوع بياناته، فعليك استخدام التعليمة ALTER المذكورة في فقرة تعديل الجداول.

سننشئ أولًا جدولًا جديدًا، كيف لا تؤثر على جداولنا السابقة التي تحوي معلومات مهمة:

```
hsoubguide=# CREATE TABLE test_table(
hsoubguide(# id integer PRIMARY KEY,
hsoubguide(# number integer,
hsoubguide(# name character varying(5)
hsoubguide(# );

CREATE TABLE
hsoubguide=# INSERT INTO test_table VALUES
hsoubguide-# (1,10,'hello'),
hsoubguide-# (2,13,'mosta'),
hsoubguide-# (3,-5,'test2'),
hsoubguide-# (4,22,'hel..'),
hsoubguide-# (5,-9,'test1');

INSERT 0 5
```

وهذه هي محتويات الجدول الجديد:

```
hsoubguide=# SELECT * from test_table;
```

id	number	name
1	10	hello
2	13	mosta
3	-5	test2
4	22	hel..
5	-9	test1

4. 6. 1. التعديل على الأسطر

يمكننا باستخدام الأمر UPDATE مع إضافة التوجيه SET تعديل الأسطر التي تحقق الشرط المذكور في تعليمة الترشيح WHERE كما في المثال التالي:

```
hsoubguide=# UPDATE test_table SET name = 'O_O' WHERE id =1;
UPDATE 1

hsoubguide=# SELECT * FROM test_table ;
```

id	number	name
2	13	mosta
4	22	hel..
3	-50	test2
5	-90	test1
1	10	O_O

كما يمكن تعديل عدة أعمدة معًا حسب شرط محدد، كما يلي:

```
hsoubguide=# UPDATE test_table
hsoubguide=# SET name = ('WOW' || id ),
hsoubguide=# number = id*10
hsoubguide=# WHERE id > 3;

UPDATE 2

hsoubguide=# SELECT * from test_table ;
```

id	number	name
2	13	mosta
3	-50	test2
1	10	O_O
4	40	WOW4
5	50	WOW5

عدَّلنا في المثال السابق القيمة في العمود name لتحتوي كلمة WOW موصولة بالرقم المحتوى في العمود id للسطر نفسه. وعدَّلنا القيمة في العمود number لتكون ناتج ضرب الرقم الموجود في العمود id مضروبًا بالعدد 10. وهذه التعديلات تتم على الأسطر التي يتحقق فيها الشرط `id>3`.

في حال لم نضع أي شرط على القيام بالتعديل، فإن التعديل ينفّذ على جميع الأسطر، كما يلي:

```
hsoubguide=# UPDATE test_table
hsoubguide=# SET number = LENGTH(name);

UPDATE 5

hsoubguide=# SELECT * from test_table ;
```

id	number	name
2	5	mosta
3	5	test2
1	3	O_O
4	4	WOW4
5	4	WOW5

استخدمنا التابع LENGTH الذي يُرجع عدد الحروف للوسيط المُمرَّر له، ثم أسندناها للعمود number، وذلك دون استخدام أي شرط، لذلك تم تنفيذ التعديلات على جميع الأسطر.

4. 6. 2. إجراء عمليات حسابية على الأعمدة

يمكننا استخدام تعليمة التعديل UPDATE لتعديل قيم عمود ما عن طريق عملية حسابية أو أي عملية أخرى، كما في المثال التالي:

```
hsoubguide=# UPDATE test_table
hsoubguide-# SET number = number*10
hsoubguide-# WHERE number >3
hsoubguide-# ;

UPDATE 4
hsoubguide=# SELECT * from test_table
hsoubguide-# ;
```

id	number	name
1	3	O_O
2	50	mosta
3	50	test2
4	40	WOW4
5	40	WOW5

4. 6. 3. تعديل أسطر الجدول مع إظهار التعديلات

يمكن استخدام الكلمة المفتاحية RETURNING بشكل مشابه للاستعلام SELECT في نهاية عبارة التعديل، بحيث يتم التعديل ثم الاستعلام عن بعض الأعمدة في الجدول المُعدّل وعرضها، وذلك كما في المثال التالي:

```
hsoubguide=# UPDATE test_table
hsoubguide=# SET name=number
hsoubguide=# WHERE id<3
hsoubguide=# RETURNING id,number,name;
```

id	number	name
1	10	10
2	13	13

4. 6. 4. حذف الأسطر

يمكننا حذف سطر ما عن طريق تحديد شرط ما يشير إلى ذلك السطر بالذات دون سواه، ويُعد المعرّف الخاص بالسطر أفضل ما يمكن تحديده لحذف سطرٍ ما مع ضمان عدم حذف سواه، وذلك لأن له المواصفة PRIMARY KEY التي تمنع إنشاء سطرين بنفس القيمة في العمود id.

سنسترجع أولًا الجدول الخاص بنا قبل إجراء عمليات التعديل في الفقرة السابقة، وذلك عن طريق حذف جميع الأسطر، ثم إدخالها مرةً أخرى.

يمكنك حذف جميع محتويات الجدول كما يلي:

```
hsoubguide=# DELETE FROM test_table;

DELETE 5

hsoubguide=# SELECT * FROM test_table;
```

id	number	name

ثم سنعيد إدخال الأسطر كما سبق:

```
hsoubguide=# INSERT INTO test_table VALUES
hsoubguide-# (1,10,'hello'),
hsoubguide-# (2,13,'mosta'),
hsoubguide-# (3,-5,'test2'),
hsoubguide-# (4,22,'hel..'),
hsoubguide-# (5,-9,'test1');

INSERT 0 5
```

سنحذف الآن السطر ذا المُعرّف id بقيمة 5:

```
hsoubguide=# DELETE FROM test_table
hsoubguide-# WHERE id=5;

DELETE 1

hsoubguide=# SELECT * FROM test_table;
```

id	number	name
1	10	hello
2	13	mosta
3	-5	test2
4	22	hel..

> تُعد تعليمة DELETE من التعليمات الخطيرة، فالسطر المحذوف لا يمكن استعادته إلا في حال وجود نسخة احتياطية عن قاعدة البيانات، فاستخدم هذه التعليمة بحذر.

يمكننا طبعًا استخدام تعليمة الترشيح WHERE لتحديد أكثر من سطر، ومن ثمّ سيتم حذف جميع الأسطر التي تحقق الشرط المحدد، فمثلًا يمكننا حذف جميع الأسطر التي تكون قيمة العمود number فيها موجبة كما يلي:

```
hsoubguide=# DELETE FROM test_table
hsoubguide=# WHERE number>0;

DELETE 3

hsoubguide=# SELECT * from test_table;
```

id	number	name
3	-5	test2

4. 7. الربط Join

سنتعرف في هذه الفقرة على أسلوب أساسي في استخراج البيانات المرتبطة من عدة جداول، وطريقتها الرئيسية ببساطة، هي طلب البيانات من عدة جداول ثم تصفيتها، وسنبدأ بأبسط طرق الربط على الإطلاق، ونتدرج فيها إلى الوصول إلى أنواع الربط المتعددة.

4. 7. 1. الاستعلام من عدة جداول معًا

سننشئ جدولين للاختبار وإدخال بيانات إليهما، كي نتمكن من استعراض المثال الخاص بنا:

الجدول الأول اسمه names فيه عمودان، الأول رقم معرّف للشخص، والثاني فيه اسمه.

```
hsoubguide=# CREATE TABLE names(
hsoubguide(#     id integer PRIMARY KEY,
hsoubguide(#     name character varying(255)
hsoubguide(# );

CREATE TABLE
```

```
hsoubguide=# INSERT INTO names VALUES
hsoubguide-# (1,'mostafa'),
hsoubguide-# (2,'ali'),
hsoubguide-# (3,'fares');

INSERT 0 3

hsoubguide=# SELECT * FROM names;
```

id	name
1	mostafa
2	ali
3	fares

أمـا الجـدول الثانـي، ففيـه أرقـام هواتـف نفـس الأشـخاص، ولكـن لـن نضـع فيـه أسـماءهم، ففيـه الرقـم المعـرّف بالشـخص، ورقـم هاتفـه:

```
hsoubguide=# CREATE TABLE phones(
hsoubguide(#     id integer PRIMARY KEY,
hsoubguide(#     phone character varying(15)
hsoubguide(# );

CREATE TABLE

hsoubguide=# INSERT INTO phones VALUES
hsoubguide-# (1,'+966123456789'),
hsoubguide-# (2,'+961111111111'),
hsoubguide-# (3,'+962333333333');

INSERT 0 3

hsoubguide=# SELECT * FROM phones;
```

id	phone
1	+966123456789
2	+961111111111
3	+962333333333

فلنبدأ بالاستعلام عن البيانات من الجدولين معًا:

```
hsoubguide=# SELECT * FROM names, phones;
```

id	name	id	phone
1	mostafa	1	+966123456789
2	ali	1	+966123456789
3	fares	1	+966123456789
1	mostafa	2	+961111111111
2	ali	2	+961111111111
3	fares	2	+961111111111
1	mostafa	3	+962333333333
2	ali	3	+962333333333
3	fares	3	+962333333333

لاحظ كيف تم إظهار جميع أسطر الجدول الأول مرتبطة بسطر واحد من الجدول الثاني، ثم مرةً أخرى، جميع أسطر الجدول الأول مرتبطة بالسطر الثاني من الجدول الثاني وهكذا.

ماذا لو أردنا عرض اسم الشخص مع رقم هاتفه؟ يكفي أن نختار الأسطر التي يتوافق فيها العمود id من الجدول names مع العمود id من الجدول phones، ولكن علينا كتابة اسم الجدول تليه نقطة . ثم اسم العمود id كي نتمكن من تمييز العمود الموجود في الجدول الأول عن الآخر في الجدول الثاني، كما يلي:

```
hsoubguide=# SELECT * FROM phones, names
hsoubguide-# WHERE phones.id = names.id;
```

id	phone	id	name
1	+966123456789	1	mostafa
2	+961111111111	2	ali
3	+962333333333	3	fares

ربطنا لتوّنا بين جدولين، وسنحسّن المخرجات الآن قليلًا بعرض العمود id مرة واحدة:

```
hsoubguide=# SELECT phones.id,phone,name
hsoubguide-# FROM phones,names
hsoubguide-# WHERE phones.id = names.id;
```

id	phone	name
1	+966123456789	mostafa
2	+961111111111	ali
3	+962333333333	fares

والآن يمكننا أن نتّجه إلى تطبيق الربط على مثالنا الخاص بالمنتجات المشتراة، هل تتذكر جدول المنتجات المُشتراة:

```
hsoubguide=# SELECT *
hsoubguide-# FROM purchase_items
hsoubguide-# LIMIT 6;
```

id	purchase_id	product_id	price	quantity	state
2	1	3	27.99	1	Delivered
3	1	8	108.00	1	Delivered
4	2	1	9.99	2	Delivered
5	3	12	9.99	1	Delivered
6	3	17	14.99	4	Delivered

لاحظ الأعمدة الأولى، purchase_id يدل على المعرّف الخاص بعملية الشراء كاملة (التي يقوم بها زبون ما في وقت محدد)، والعمود product_id الذي يحدد المعرف الخاص بالمنتج الذي تم شراؤه.

هذه الأعمدة لا تحتوي بيانات عن المنتجات أو عن عملية الشراء، بل هي تشير إلى المعرّف الخاص بها في الجداول الخاصة بها، فهل يمكننا استخدام هذا المعرّف لجلب تلك البيانات من جداولها ثم دمج تلك البيانات مع بيانات الجدول الحالي؟

ما نحاول الوصول إليه في هذه الفقرة، هو ربط عدة جداول مع بعضها، للحصول على معلومات متكاملة في جدول الخرج، فبدلًا من عرض المعرّف الخاص بالمنتج، نريد عرض اسمه مثلًا، وبدلًا من عرض المعرّف الخاص بعملية الشراء نريد عرض المستخدم الذي قام بالشراء.

فلنلقِ نظرة على مخطط قاعدة البيانات الخاصة بنا:

```
hsoubguide=# \dt
```

Schema	Name	Type	Owner
public	products	table	postgres
public	purchase_items	table	postgres
public	purchases	table	postgres
public	users	table	postgres

وسنستعلم في مثالنا التالي عن المنتجات التي تم شراؤها مؤخرًا، ونحتاج للقيام بذلك إلى بيانات من جدولي المنتجات والمشتريات معًا، وسنبدأ بأخذ نظرة إلى الجدولين لمعرفة الأعمدة التي سنحتاج إليها:

يحوي الجدول products على تفاصيل المنتجات، كاسم المنتج وسعره:

```
hsoubguide=# SELECT * from products;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}
2	Python Book	29.99	2011-01-01 22:00:00+02		{Book,Programming,Python}
3	Ruby Book	27.99	2011-01-01 22:00:00+02		{Book,Programming,Ruby}
4	Baby Book	7.99	2011-01-01 22:00:00+02		{Book,Children,Baby}
5	Coloring Book	5.99	2011-01-01 22:00:00+02		{Book,Children}
6	Desktop Computer	499.99	2011-01-01 22:00:00+02		{Technology}
7	Laptop Computer	899.99	2011-01-01 22:00:00+02		{Technology}

id	title	price	created_at	deleted_at	tags
8	MP3 Player	108.00	2011-01-01 22:00:00+02		{Technology,Music}
9	42" LCD TV	499.00	2011-01-01 22:00:00+02		{Technology,TV}
10	42" Plasma TV	529.00	2011-01-01 22:00:00+02		{Technology,TV}
11	Classical CD	9.99	2011-01-01 22:00:00+02		{Music}
12	Holiday CD	9.99	2011-01-01 22:00:00+02		{Music}
13	Country CD	9.99	2011-01-01 22:00:00+02		{Music}
14	Pop CD	9.99	2011-01-01 22:00:00+02		{Music}
15	Electronic CD	9.99	2011-01-01 22:00:00+02		{Music}
16	Comedy Movie	14.99	2011-01-01 22:00:00+02		{Movie,Comedy}
17	Documentary	14.99	2011-01-01 22:00:00+02		{Movie}
18	Romantic	14.99	2011-01-01 22:00:00+02		{Movie}
19	Drama	14.99	2011-01-01 22:00:00+02		{Movie}
20	Action	14.99	2011-01-01 22:00:00+02		{Movie}

بينما يحوي الجدول purchase_items على تفاصيل عمليات الشراء، ففي كل عملية شراء، يمكن أن يكون هناك عدة منتجات مُشتراة، حيث يحوي السطر الواحد في هذا الجدول على عملية شراء لمنتج واحد، ويخزن العمود purchase_id رقمًا تسلسليًا لعملية الشراء الكاملة، ويشير العمود product_id إلى الرقم المعرّف للمنتج، وهو نفسه الموجود في الجدول السابق products وبفضل وجوده يمكننا الربط بين الجدولين.

```
hsoubguide=# SELECT * from purchase_items LIMIT 12;
```

id	purchase_id	product_id	price	quantity	state
2	1	3	27.99	1	Delivered
3	1	8	108.00	1	Delivered
4	2	1	9.99	2	Delivered
5	3	12	9.99	1	Delivered
6	3	17	14.99	4	Delivered
7	3	11	9.99	1	Delivered
8	4	4	7.99	3	Delivered
9	5	18	14.99	1	Delivered
10	5	2	29.99	4	Delivered
11	6	5	5.99	1	Delivered
12	7	6	499.99	3	Returned
13	8	10	529.00	1	Delivered

الطريقة الأساسية لربط الجداول هي المفاتيح (keys) وسنشرح عنها لاحقًا باستفاضة أكثر، أما ما يهمنا الآن هو معرفة أن العمود product_id في جدول المشتريات يشير إلى العمود id في جدول المنتجات.

والآن يمكننا إنشاء الاستعلام كما يلي:

```
SELECT
   products.title,
   purchase_items.quantity
FROM
   products,
   purchase_items
WHERE
   products.id = purchase_items.product_id
LIMIT 12;
```

لاحظ كيف يتم طلب عمودين من جدولين مختلفين عن طريق كتابة اسم الجدول تتبعه نقطة . ثم اسم العمود المطلوب.

وستكون المخرجات بالشكل التالي:

title	quantity
Ruby Book	1
MP3 Player	1
Dictionary	2
Holiday CD	1
Documentary	4
Classical CD	1
Baby Book	3
Romantic	1
Python Book	4
Coloring Book	1
Desktop Computer	3
42" Plasma TV	1

نلاحظ في المثال السابق، أنه ليس لدينا تفاصيل عن عملية الشراء، سوى عن الكمية المشتراة من كل منتج، وسنحاول في المثال التالي استعراض المنتج والكمية المشتراة، مع ذكر اسم المشتري وتاريخ الشراء، كي تكون المعلومات المعروضة مفيدة أكثر.

نستعرض أولًا الجدول purchases:

```
hsoubguide=# SELECT * FROM purchases LIMIT 7;
```

id	created_at	name	address	state	zipcode	user_id
1	2011-03-16 17:03:00+02	Harrison Jonson	6425 43rd St.	FL	50382	7
2	2011-09-14 08:00:00+03	Cortney Fontanilla	321 MLK Ave.	WA	43895	30
3	2011-09-11 08:54:00+03	Ruthie Vashon	2307 45th St.	GA	98937	18
4	2011-02-27 22:53:00+02	Isabel Wynn	7046 10th Ave.	NY	57243	11
5	2011-12-20 14:45:00+02	Shari Dutra	4046 8th Ave.	FL	61539	34

id	created_at	name	address	state	zipcode	user_id
6	2011-12-10 15:29:00+02	Kristofer Galvez	2545 8th Ave.	WA	83868	39
7	2011-06-19 06:42:00+03	Maudie Medlen	2049 44th Ave.	FL	52107	8

وجدنا الأعمدة التي كنا نبحث عنها في ما أردنا الوصول إليه قبل قليل، إنها الأعمدة name و created_at، ويمكننا الآن ربط هذين العمودين بالاستعلام السابق للحصول على مخرجات أكثر فائدة:

```
SELECT
   products.title,
   purchase_items.quantity,
   purchases.name,
   purchases.created_at
FROM
   products,
   purchase_items,
   purchases
WHERE
   products.id = purchase_items.product_id AND
   purchases.id = purchase_items.purchase_id
LIMIT 12;
```

لاحظ في المثال السابق عبارة WHERE المرشِّحة، حيث وضعنا فيها شرطين. وسنحصل على المخرجات التالية:

title	quantity	name	created_at
Ruby Book	1	Harrison Jonson	2011-03-16 17:03:00+02
MP3 Player	1	Harrison Jonson	2011-03-16 17:03:00+02
Dictionary	2	Cortney Fontanilla	2011-09-14 08:00:00+03
Holiday CD	1	Ruthie Vashon	2011-09-11 08:54:00+03
Documentary	4	Ruthie Vashon	2011-09-11 08:54:00+03
Classical CD	1	Ruthie Vashon	2011-09-11 08:54:00+03
Baby Book	3	Isabel Wynn	2011-02-27 22:53:00+02
Romantic	1	Shari Dutra	2011-12-20 14:45:00+02

title	quantity	name	created_at
Python Book	4	Shari Dutra	2011-12-20 14:45:00+02
Coloring Book	1	Kristofer Galvez	2011-12-10 15:29:00+02
Desktop Computer	3	Maudie Medlen	2011-06-19 06:42:00+03
42" Plasma TV	1	Isabel Crissman	2011-05-28 04:19:00+03

في المثال السابق، ربطنا ثلاثة جداول للحصول على معلومات مترابطة وإظهارها في جدول مخرجات واحد، ولكن ربما لاحظت كيف أصبح الاستعلام طويلًا، ومعقّدًا بعض الشيء، ولذلك وُجد العرض View كطريقة لحفظ الاستعلام كجدول منطقي جديد، لا يُخزّن في الذاكرة، ولكن يمكننا التعامل معه كأنه كذلك، وهو ما سنتحدث عنه بعد استعراض أنواع الربط المختلفة.

4. 7. 2. أنواع الربط

سنستعرض أنواعًا أخرى من الربط الذي قمنا به في الفقرات السابقة، وسنستخدم لذلك مثالًا جديدًا، ننشئ فيه جدولين، الأول لتخزين اسم الطالب وكنيته، والثاني نخزّن فيه اسم الطالب وعلامته في مادة الرياضيات.

> للاستفادة الكاملة من هذا الكتاب، نتمنّى أن تطبّق عزيزي القارئ الأمثلة بنفسك، كي يحصل على الخبرة العملية إلى جانب الفهم النظري للمحتوى.

```
hsoubguide=# CREATE TABLE student(
hsoubguide(#     id integer PRIMARY KEY,
hsoubguide(#     name character varying(255),
hsoubguide(#     last_name character varying (255)
hsoubguide(# );

CREATE TABLE

hsoubguide=# CREATE TABLE marks(
hsoubguide(#     id integer PRIMARY KEY,
hsoubguide(#     name character varying(255),
hsoubguide(#     marks integer
hsoubguide(# );

CREATE TABLE
```

```
hsoubguide=# INSERT INTO student VALUES
hsoubguide-# (1,'mostafa','ayesh'),
hsoubguide-# (2,'ali','badawi'),
hsoubguide-# (3,'samer','khateeb'),
hsoubguide-#  (4,'amer','masre');

INSERT 0 4

hsoubguide=# INSERT INTO marks VALUES
hsoubguide-# (1,'ali',14),
hsoubguide-# (2,'ramez',20),
hsoubguide-# (3,'amer',16),
hsoubguide-# (4,'fadi',18);

INSERT 0 4
```

لاحظ أن جدول العلامات لا يحتوي جميع الطلاب المذكورة أسمائهم في جدول الطلاب، وكذلك يحتوى طلابًا غير مذكورين في جدول الطلاب، ولاحظ كذلك أن المعرّف id لطالب ما، ليس من الضروري أن يطابق المعرّف الخاص به في جدول العلامات ! ولذلك فلن نتمكن من ربط الجدولين عن طريق المعرّف الخاص بالسطر.

أ. الربط الداخلي INNER JOIN

نستعرض فيه التقاطع بين جدولين دون أي معلومات إضافية:

```
hsoubguide=# SELECT student.name,marks.mark
hsoubguide-# FROM student
hsoubguide-# INNER JOIN marks
hsoubguide-# ON student.name = marks.name;
```

name	mark
ali	14
amer	16

يبدو ذلك مشابهًا لما رأيناه في الفقرة السابقة:

```
hsoubguide=# SELECT student.name,marks.mark
hsoubguide-# FROM student,marks
hsoubguide-# WHERE student.name = marks.name;
```

name	mark
ali	14
amer	16

ب. الربط اليساري LEFT JOIN

في هذا الربط، يتم عرض جميع أسطر الجدول اليساري (أي الجدول المرتبط بالتوجيه FROM) ويتم عرض الأسطر التي تقابلها أو NULL في حال عدم وجود ما يقابلها:

```
hsoubguide=# SELECT student.name,marks.mark
hsoubguide-# FROM student
hsoubguide-# LEFT JOIN marks
hsoubguide-# ON student.name = marks.name;
```

name	mark
ali	14
amer	16
samer	
mostafa	

ج. الربط اليساري الخرجي LEFT OUTER JOIN

نستخدمه للحصول على الأسطر التي ليس لها ما يقابلها في الجدول اليميني كما يلي:

```
hsoubguide=# SELECT student.name,marks.mark
hsoubguide-# FROM student
hsoubguide-# LEFT JOIN marks
hsoubguide-# ON student.name = marks.name
hsoubguide-# WHERE mark IS NULL;
```

name	mark
samer	
mostafa	

د. الربط اليميني RIGHT JOIN

كما في الربط اليساري، إلا أن الربط اليميني يظهر جميع أسطر الجدول المرتبط بالتعليمة JOIN (في مثالنا هو الجدول marks)، بينما يظهر الأسطر التي لها مقابل فقط من الجدول الأول (اليساري) أو NULL في حال عدم وجود تقابل، كما يلي:

```
hsoubguide=# SELECT student.name,marks.mark
hsoubguide-# FROM student
hsoubguide-# RIGHT JOIN marks
hsoubguide-# ON student.name = marks.name;
```

name	mark
ali	14
	20
amer	16
	18

من الواضح في المثال السابق أن العبارة SELECT student.name, marks.mark غير مناسبة لهذا النوع من الربط، وذلك لأن العلامات التي ليس لها مقابل لأسماء أصحابها في الجدول names ستظهر بلا أسماء، وهذا أمر غير منطقي في العرض، لذلك سنضيف إضافةً بسيطة لحل هذه المشكلة:

```
hsoubguide=# SELECT COALESCE(student.name,marks.name) AS student_
name, hsoubguide=# marks.mark
hsoubguide=# FROM student
hsoubguide=# RIGHT JOIN marks
hsoubguide=# ON student.name = marks.name;
```

student_name	mark
ali	14
ramez	20

student_name	mark
amer	16
fadi	18

استخدمنا التابع COALESCE الذي يؤدي إلى عرض قيمة السطر من الجدول الأول له إن لم يكن قيمته NULL، وإلا فإنه يعرض قيمة السطر من الجدول الثاني، كما استخدمنا التوجيه AS كي لا يظهر اسم التابع في جدول المخرجات، ويظهر بدلًا منه العبارة student_name.

ه. الربط اليميني الخارجي RIGHT OUTER JOIN

هنا يتم استثناء الأسطر التي لها مقابل في الجدول الأول، ونستعرض الأسطر التي ليس لها مقابل من الجدول الأول فقط.

```
hsoubguide=# SELECT COALESCE(student.name,marks.name) AS Student_
name, hsoubguide-# marks.mark
hsoubguide-# FROM student
hsoubguide-# RIGHT JOIN marks
hsoubguide-# ON student.name = marks.name
hsoubguide-# WHERE student.name IS NULL;
```

student_name	mark
ramez	20
fadi	18

و. الربط الخارجي الكامل FULL OUTER JOIN

يتم فيه إظهار جميع الأسطر من جميع الجداول، سواء كان لها مقابل (عندها تُعرَض قيمته) أو لم يكن لها مقابل (يُعرض NULL).

```
hsoubguide=# SELECT COALESCE(student.name,marks.name) AS Student_
name, hsoubguide-# marks.mark
hsoubguide-# FROM student
hsoubguide-# FULL OUTER JOIN marks
hsoubguide-# ON student.name = marks.name;
```

student_name	mark
ali	14
ramez	20
amer	16
fadi	18
samer	
mostafa	

ما رأيكم لو نضع علامة الصفر لكل من لم يقدم الامتحان؟ نترك لكم تنفيذ الاستعلام التالي:

```
SELECT COALESCE(student.name,marks.name) AS Student_name,
COALESCE(marks.mark,0) AS mark
FROM student
FULL OUTER JOIN marks
ON student.name = marks.name;
```

z. الربط الخارجي الكامل حصرًا FULL OUTER JOIN-only

نستخدم هذا الربط لاستعراض جميع الأسطر من الجدولين باستثناء الأسطر المرتبطة ببعضها من الجدولين معًا حسب شرط الربط، فهو بذلك يعطي نتائج الربط الخارجي الكامل، مستثنى منها نتائج الربط الداخلي:

```
hsoubguide=# SELECT COALESCE(student.name,marks.name) AS Student_
name, hsoubguide-# COALESCE(marks.mark,0)
hsoubguide-# FROM student
hsoubguide-# FULL OUTER JOIN marks
hsoubguide-# ON student.name = marks.name
hsoubguide-# WHERE student.name IS NULL OR marks.name IS NULL;
```

student_name	coalesce
ramez	20
fadi	18
samer	0
mostafa	0

8.4. خلاصة الفصل

تعرفنا في هذا الفصل على كيفية إنشاء وتعديل وحذف الجداول، وكذلك تعرفنا على كيفية إدخال البيانات فيها ثم الاستعلام عنها مع ترشيحها والاستفادة من العديد من الأوامر للحصول على المخرجات المطلوبة، كما تعرفنا على عمليات التجميع وعلى طرق ربط الجداول للحصول على النتائج المستخلصة من عدة مصادر.

ومع ازدياد خبرتك في التعامل مع الاستعلامات، وخصوصًا بعد تعرفك على طرق ربط الجداول في هذا الفصل، ربما تكون قد لاحظت أن بعض الاستعلامات قد أصبحت طويلة، وصعبة القراءة، وبعضها يتكرر كثيرًا مع بعض التعديلات، ولذا فقد آن الأوان أن نتعرف إلى العرض View وبعض المزايا المتقدمة في Postgres في الفصل التالي، ليسهل علينا استخدام الاستعلامات الطويلة.

5. مزايا متقدمة في Postgres

نتعرف في هذا الفصل إلى العرض View وتعابير الجداول الشائعة CTE التي تساهم في تحسين طريقة كتابة الاستعلامات، وذلك يجعلها أقصر وأسهل قراءةً، وسنتعلم استخدام دوال النوافذ Window Functions التي تسمح بإجراء عمليات تجميعية على مجموعة جزئية من الأسطر، كما سنتعرف على الفهارس وأهميتها في تسريع البحث ضمن البيانات.

5. 1. العرض View

هو جدول منطقي يقوم بوصل أجزاء البيانات الأساسية تلقائيًا، فهو لا يكرر البيانات ولا يحتفظ فيها، بل يعيد ترتيبها لعرضها فقط. وتكمن فوائد استخدام العرض في تبسيط نموذج البيانات عند الحاجة إلى تقديمه للآخرين للعمل عليه، كما يمكن استخدامه لتبسيط العمل بالبيانات الخاصة بك.

وإذا وجدت نفسك تدمج نفس المجموعة من البيانات بشكل متكرر فيمكن للعرض تسهيل هذه العملية بدلًا من تكرارها عدة مرات. وكذلك عند العمل مع أشخاص غير تقنيين ليسوا على دراية بلغة SQL يكون العرض هو الطريقة المُثلى لتقديم البيانات غير المنتظمة.

5. 1. 1. طريقة عمل العرض

فنلقِ نظرةً على الجداول التالية أولًا:

- جدول أقسام الشركة:

departments

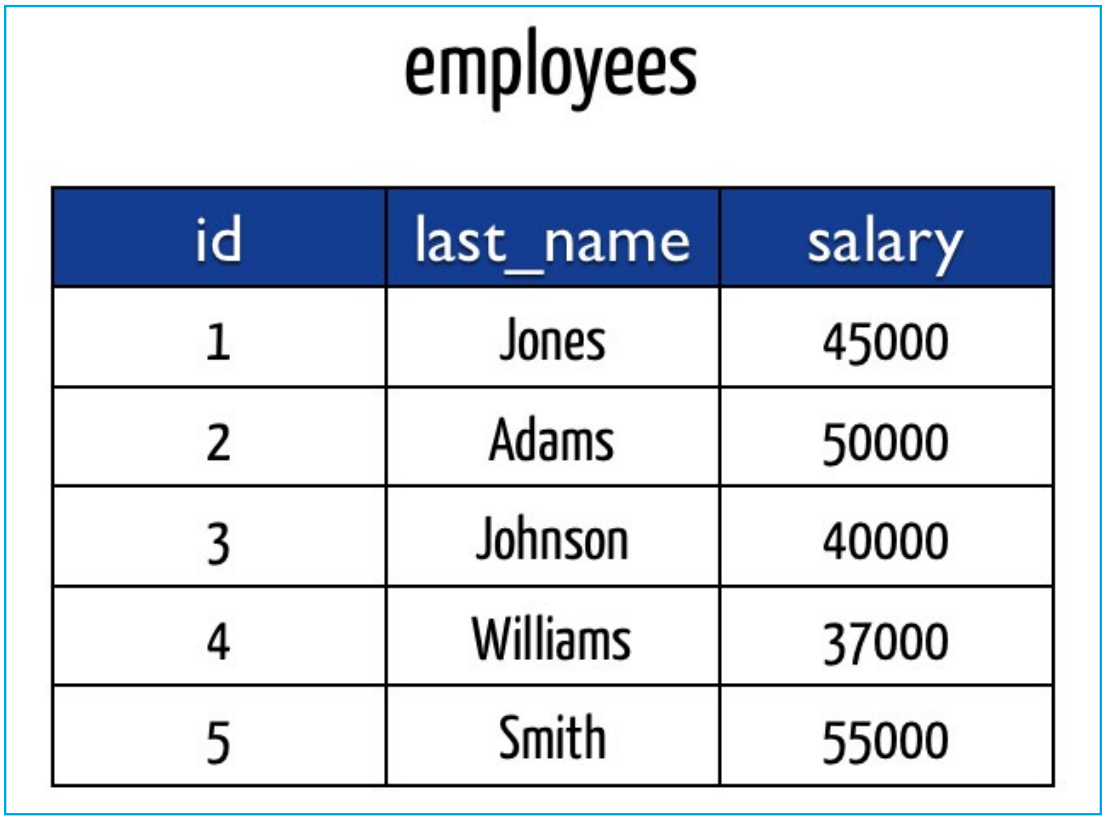
id	department
1	Accounting
2	Marketing
3	Sales

- الجدول الذي يحدد انتماء الموظف إلى قسم معين:

employee_departments

employee_id	department_id
1	1
2	3
3	2
4	1
5	3

- جدول الموظفين:



employees

id	last_name	salary
1	Jones	45000
2	Adams	50000
3	Johnson	40000
4	Williams	37000
5	Smith	55000

يمكنك نسخ التعليمات التالية لإنشاء الجداول وإدخال البيانات المذكورة في الصور السابقة.

```
CREATE TABLE departments(
    id serial PRIMARY KEY,
    department VARCHAR (50) UNIQUE NOT NULL
);

INSERT INTO departments (department) VALUES ('Accounting'),('Marketin
g'),('Sales');
CREATE TABLE employees(
id serial PRIMARY KEY,
last_name VARCHAR (50) NOT NULL,
salary int NOT NULL
);

INSERT INTO employees (last_name,salary) VALUES ('Jones',45000),('Ada
ms',50000),('Johnson',40000),('Williams',37000),('Smith',55000);
CREATE TABLE employee_departments(
employee_id int NOT NULL,
```

```
department_id int NOT NULL
);

INSERT INTO employee_departments VALUES (1,1),(2,3),(3,2),(4,1),(5,3);
```

وللتأكد من أن الجداول أُنشئت على الوجه الصحيح وأُدخلت البيانات كما هو مخطط، فسنستعرض الجداول كما يلي:

```
hsoubguide=# SELECT * FROM departments;
```

id	department
1	Accounting
2	Marketing
3	Sales

```
hsoubguide=# SELECT * FROM employees;
```

id	last_name	salary
1	Jones	45000
2	Adams	50000
3	Johnson	40000
4	Williams	37000
5	Smith	55000

```
hsoubguide=# SELECT * FROM employee_departments ;
```

employee_id	department_id
1	1
2	3
3	2
4	1
5	3

نهدف في المثال التالي إلى عرض أسماء الموظفين مع رواتبهم والقسم الذي ينتمون إليه، وللقيام بذلك سنكتب الاستعلام التالي:

```
SELECT
  employees.last_name,
  employees.salary,
  departments.department
FROM
  employees,
  employee_departments,
  departments
WHERE
  employees.id = employee_departments.employee_id
  AND
  departments.id = employee_departments.department_id;
```

وتكون المخرجات كما يلي:

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Johnson	40000	Marketing
Williams	37000	Accounting
Smith	55000	Sales

إذا أمعنت النظر في الجدول السابق في المخرجات، قد تتمنى لو أنه مُخزَّن في الذاكرة بالفعل، فبدلًا من القيام بالعديد من عمليات الربط من هذا الجدول أو ذاك، كان يمكن أن نتعامل مع هذا الجدول مباشرةً بدلًا من كتابة الاستعلام الطويل ذاك في كل مرة نريد استعراض الموظفين مع رواتبهم وأقسامهم، ولكن في الحقيقة، يمكننا إنشاء جدول وهمي عن طريق العرض View لتحقيق هذه الأمنية، كما يلي:

```
CREATE OR REPLACE VIEW employee_view AS
SELECT
  employees.last_name,
  employees.salary,
  departments.department
```

```
FROM
  employees,
  employee_departments,
  departments
WHERE
  employees.id = employee_departments.employee_id
  AND departments.id = employee_departments.department_id;
```

يمكن الآن القيام باستعلام على الجدول الافتراضي الجديد مباشرةً:

```
SELECT *
FROM employee_view;
```

وستظهر البيانات بشكل مشابه لعملية الربط التي قمنا بها سابقًا كما يلي:

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Johnson	40000	Marketing
Williams	37000	Accounting
Smith	55000	Sales

كما يمكننا التعامل مع هذا الجدول المنطقي وكأنه جدول حقيقي، فمثلًا يمكننا الاستعلام عن أسطر هذا الجدول مع ترشيحها أو القيام بأي عملية أخرى تتم على الجداول العادية المخزنة كما في المثال التالي:

```
hsoubguide=# SELECT * FROM employee_view WHERE salary > 40000;
```

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Smith	55000	Sales

والآن صار بالإمكان تبسيط الاستعلامات الطويلة المتكررة والمتداخلة عن طريق استخدام العرض.

5. 2. عبارات الجداول الشائعة (CTE)

تشبه عبارات الجداول الشائعة CTE العرض View في بعض النواحي، فهي نتيجة مؤقتة لاستعلام سابق، يمكننا القيام بعمليات استعلام عليها مرة أخرى دون أن يكون لها وجود دائم في الذاكرة. فهي تشكل جدولًا افتراضيًا جديدًا ناتجًا عن الاستعلام الأول، ويمكننا القيام بعمليات على هذا الجدول، دون أن يكون له تخزين دائم في الذاكرة.

الاستخدام الأساسي لها هو تبسيط الاستعلامات المعقدة، أو الاستعلامات التي تربط عدة جداول معًا، وذلك بعمل هذا الجدول الافتراضي الوسيط أولًا، ثم القيام بالعمليات عليه بعد ذلك أو عمل جدول وسيط آخر حتى الوصول إلى النتائج المطلوبة.

سنبدأ باستعلام بسيط من الجدول products عن المنتجات التي توجد ضمن تصنيف Book أو TV كما يلي:

```
hsoubguide=# SELECT title,price
hsoubguide-# FROM products
hsoubguide-# WHERE tags[1]='Book' OR tags[2]='TV';
```

title	price
Dictionary	9.99
Python Book	29.99
Ruby Book	27.99
Baby Book	7.99
Coloring Book	5.99
42" LCD TV	499.00
42" Plasma TV	529.00
Python Book	29.99

والآن، لنفترض أننا لما نظرنا إلى الجدول الناتج عن الاستعلام، شعرنا بالحاجة إلى المزيد من الترشيح، فأردنا مثلًا الاستعلام عن العناصر التي سعرها أكبر من 10 مع ترتيبها حسب الاسم، وإظهار 5 عناصر فقط.

عندها سنضيف العبارة التالية إلى الاستعلام السابق:

```
AND price>10 ORDER BY title LIMIT 5;
```

ليصبح الاستعلام الكامل:

```
hsoubguide=# SELECT title,price
hsoubguide=# FROM products
hsoubguide=# WHERE (tags[1]='Book' OR tags[2]='TV')
hsoubguide=# AND price>10
hsoubguide=# ORDER BY title
hsoubguide=# LIMIT 5;
```

title	price
42" LCD TV	499.00
42" Plasma TV	529.00
Python Book	29.99
Python Book	29.99
Ruby Book	27.99

يمكننا تحويل الاستعلام السابق إلى مرحلتين، الأولى هي عبارة جداول شائعة CTE، والثانية هي استعلام من الجدول الناتج عن عبارة الجداول الشائعة، وذلك كما يلي:

```
hsoubguide=# WITH small_cte AS(
hsoubguide-#     SELECT title,price
hsoubguide-#     FROM products
hsoubguide-#     WHERE (tags[1]='Book' OR tags[2]='TV')
hsoubguide-#     )
hsoubguide-#     SELECT * FROM small_cte
hsoubguide-#     WHERE price>10
hsoubguide-#     ORDER BY title
hsoubguide-#     LIMIT 5;
```

title	price
42" LCD TV	499.00
42" Plasma TV	529.00
Python Book	29.99

title	price
Python Book	29.99
Ruby Book	27.99

بعد أن فهمنا طريقة استخدام تعابير الجداول الشائعة، لا بد أن سؤالًا مهمًا يخطر لنا، وهو الفرق بينها وبين العرض View، والفرق بسيط، فتعابير الجداول الشائعة تُنشَأ أثناء الاستعلام، بينما تُنشَأ العروض مسبقًا، وتُخزّن ككائنات في قاعدة البيانات (دون تخزين الجدول الناتج عنها بالطبع كما ذكرنا من قبل) حيث يمكن استخدامها مرات عديدة، لذلك يُطلق على تعابير الجداول الشائعة أحيانًا العروض المؤقتة أو العروض الآنية.

5. 2. 1. استخدام عدة تعابير شائعة معًا

سننشئ جدولين بسيطين لتوضيح هذه الفكرة كما يلي:

```
hsoubguide=# CREATE TABLE test_1(
hsoubguide(#     id INT,
hsoubguide(#     value varchar(20)
hsoubguide(# );

CREATE TABLE
hsoubguide=# CREATE TABLE test_2(
hsoubguide(#     id INT,
hsoubguide(#     test_1_id INT,
hsoubguide(#     value varchar(20)
hsoubguide(# );

CREATE TABLE
```

في كل من الجدولين، عمود رقم معرف، وعمود للقيمة، وفي الجدول الثاني عمود فيه ربط مع أحد أسطر الجدول الأول من خلال الرقم المعرف الخاص به.

```
hsoubguide=# INSERT INTO test_1
hsoubguide-# VALUES(1,'aaa'),(2,'bbb'),(3,'ccc');

INSERT 0 3
```

```
hsoubguide=# INSERT INTO test_2
hsoubguide-# VALUES(1,3,'AAA'),(2,1,'BBB'),(3,3,'CCC');

INSERT 0 3

hsoubguide=# SELECT test_1.value as v1,test_2.value as v2
hsoubguide-# FROM test_1,test_2
hsoubguide-# WHERE test_1.id=test_2.test_1_id;
```

v1	v2
aaa	BBB
ccc	CCC
ccc	AAA

يمكن ربط التعابير الشائعة مع بعضها كما، بحيث يكون التعبير الثاني مستخدمًا للجدول الناتج عن التعبير الأول كما في المثال التالي:

```
hsoubguide=# WITH cte1 AS(
hsoubguide-#     SELECT test_1.value AS V1,test_2.value AS v2
hsoubguide-#     FROM test_1,test_2
hsoubguide-#     WHERE test_1.id=test_2.test_1_id)
hsoubguide-#     ,cte2 AS(
Hsoubguide-#     SELECT * FROM cte1
hsoubguide-#     WHERE v1='ccc')
hsoubguide-#     SELECT * FROM cte2 ORDER BY v2 DESC;
```

v1	v2
ccc	CCC
ccc	AAA

هناك الكثير مما يمكن الاستفادة منه باستخدام تعابير الجداول الشائعة عند استخدامها مع الربط أو دوال النوافذ التي سنتعرف عليها في الفقرة التالية.

5. 3. دوال النوافذ (Window Functions)

تقوم دوال النوافذ (Window Functions) بإجراء عملية حسابية على مجموعة من أسطر الجدول التي لها علاقة محددة مع السطر الحالي، وهذا يشابه إلى حد ما العمليات الحسابية التي قمنا بها باستخدام العمليات التجميعية `COUNT` و `SUM` وغيرها، إلا أن استخدام دوال النوافذ لا يسبب تجميع النتيجة في سطر واحد، بل تحافظ الأسطر على ذاتها، وخلف الكواليس تقوم دوال النوافذ بالوصول إلى الأسطر اللازمة للقيام بالعملية الحسابية.

لا بد أن الشرح السابق سيتضح أكثر مع الأمثلة على أي حال. فلنلقِ نظرة على الجدول التالي:

employees

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Johnson	40000	Marketing
Williams	37000	Accounting
Smith	55000	Sales

للحصول على جدول مماثل للجدول السابق، يمكنك تطبيق العرض الذي استخدمناه في الفقرة السابقة:

```
CREATE OR REPLACE VIEW employee_view AS
SELECT
  employees.last_name,
  employees.salary,
  departments.department
FROM
  employees,
  employee_departments,
```

```
  departments
WHERE
  employees.id = employee_departments.employee_id
  AND departments.id = employee_departments.department_id;
```

ثم الاستعلام عنه كما تعلمنا سابقًا:

```
hsoubguide=# SELECT *
hsoubguide-# FROM employee_view;
```

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Johnson	40000	Marketing
Williams	37000	Accounting
Smith	55000	Sales

في المثال التالي، سنحاول استخراج مجموع الرواتب التي تُصرف في كل قسم وذلك باستخدام دالة التجميع SUM والتوجيه GROUP BY:

```
SELECT
department
,sum(salary) AS department_salary_sum
FROM employee_view
GROUP BY department;
```

وستكون المخرجات كما يلي:

department	department_salary_sum
Accounting	82000
Sales	105000
Marketing	40000

ولكن ماذا لو أردنا استعراض الجدول الأساسي هذا :

last_name	salary	department
Jones	45000	Accounting
Adams	50000	Sales
Johnson	40000	Marketing
Williams	37000	Accounting
Smith	55000	Sales

مع إضافة عمودٍ جديد، فيه مجموع الرواتب في القسم الخاص بالموظف المذكور في ذلك السطر، هل يمكننا كتابة الاستعلام التالي:

```
SELECT last_name,salary,department,SUM(salary)
FROM employee_view
GROUP BY department;
```

للأسف، لا يمكننا القيام بذلك، وسيظهر الخطأ التالي:

```
ERROR:  column "employee_view.last_name" must appear in the GROUP BY
clause or be used in an aggregate function
LINE 1: SELECT last_name,salary,department,SUM(salary)
```

ينص الخطأ على أن أي عمود غير مذكور ضمن عبارة GROUP BY أو ضمن دالة التجميع فإنه لا يمكن عرضه، وهذا منطقي، لأن دالة التجميع والعبارة GROUP BY هدفها التجميع، أي اختصار الأسطر وإظهار نتائج الاختصار.

وهنا تأتي أهمية دوال النوافذ، فالآن سنستخدم التابع SUM ولكن لن نكتب GROUP BY، بل سنستبدلها بالعبارة OVER (PARTITION BY department) وتعني أن عملية الجمع ستتم على جزء من الجدول مقسّم حسب قيمة العمود department كما يلي :

```
hsoubguide2=# SELECT
hsoubguide2-# last_name,
hsoubguide2-# salary,
hsoubguide2-# department,
hsoubguide2-# SUM(salary)
```

```
hsoubguide2-# OVER (PARTITION BY department)
hsoubguide2-# FROM employee_view;
```

last_name	salary	department	sum
Jones	45000	Accounting	82000
Williams	37000	Accounting	82000
Johnson	40000	Marketing	40000
Adams	50000	Sales	105000
Smith	55000	Sales	105000

تُقسِّم العبارة PARTITION BY الجدول إلى مجموعات حسب العمود المذكور ضمن هذه العبارة، ثم تُنفَّذ الدالة المذكورة على هذه المجموعات، ويُحفَظ الناتج لعرضه في العمود الخاص بالدالة.

5. 3. 1. عدة دوال نوافذ في استعلام واحد

يمكننا استخدام العديد من توابع النوافذ، ولكن يجب إضافة عبارة OVER خاصة بكل استدعاء من الاستدعاءات، كما في المثال التالي:

```
hsoubguide2=# SELECT last_name,salary,department,
hsoubguide2=# AVG(salary) OVER (PARTITION BY department),
hsoubguide2=# SUM(salary) OVER (PARTITION BY salary/10000)
hsoubguide2=# FROM employee_view;
```

last_name	salary	department	avg	sum
Jones	45000	Accounting	41000.000000000000	82000
Williams	37000	Accounting	41000.000000000000	82000
Johnson	40000	Marketing	40000.000000000000	40000
Adams	50000	Sales	52500.000000000000	105000
Smith	55000	Sales	52500.000000000000	105000

ولكن في حال كنا نرغب في استخدام نفس النافذة لتنفيذ عدة دوال عليها، فيمكننا تعريف النافذة في نهاية الاستعلام لمرة واحدة، واستخدامها عن طريق اسمٍ مستعارٍ لها، كما في المثال التالي:

```
hsoubguide2=# SELECT last_name,salary,department,
hsoubguide2=# AVG(salary) OVER my_window,
hsoubguide2=# SUM(salary) OVER my_window
hsoubguide2=# FROM employee_view
hsoubguide2=# WINDOW my_window AS (PARTITION BY department);
```

last_name	salary	department	avg	sum
Jones	45000	Accounting	41000.000000000000	82000
Williams	37000	Accounting	41000.000000000000	82000
Johnson	40000	Marketing	40000.000000000000	40000
Adams	50000	Sales	52500.000000000000	105000
Smith	55000	Sales	52500.000000000000	105000

في حال أردنا إعطاء العمود الحاوي على نتائج دالة النافذة اسمًا مستعارًا باستخدام AS فيجب وضعها بعد عبارة OVER كما يلي:

```
hsoubguide2=# SELECT last_name,salary,department,
hsoubguide2=# AVG(salary) OVER (PARTITION BY department) AS avg_
department_salary,
hsoubguide2=# SUM(salary) OVER (PARTITION BY department) AS sum_
department_salary
hsoubguide2=# FROM employee_view;
```

last_name	salary	department	avg_department_salary	sum_department_salary
Jones	45000	Accounting	41000.000000000000	82000
Williams	37000	Accounting	41000.000000000000	82000
Johnson	40000	Marketing	40000.000000000000	40000
Adams	50000	Sales	52500.000000000000	105000
Smith	55000	Sales	52500.000000000000	105000

5. 3. 2. مثال ممتع باستخدام دوال النافذة

سنضيف في هذا المثال عمودًا إلى الجدول فيه مجموع رواتب الموظفين مجزّأة حسب الحرف الأول من أسمائهم الأخيرة. فإذا كان هناك موظفان اسماهما الأخيرة بالحرف J سيكونان في نفس النافذة، ويتم حساب مجموع رواتبهما وإخراجه إلى جانب اسميهما. وللقيام بذلك، سنستخدم التابع LEFT الذي يأخذ وسيطين، الأول هو اسم العمود والثاني هو عدد الحروف بدءًا من اليسار التي نريد اقتطاعها.

ستكون عبارة تجزئة النوافذ كما يلي:

```
OVER (PARTITION BY LEFT(last_name,1))
```

ولكننا نريد أيضًا أن تكون المخرجات ليست فقط مجموع رواتب الأشخاص المتشابهين في الحرف الأول من اسمهم الأخير، بل نريد إضافة هذا الحرف قبل مجموع الرواتب، كي تكون المخرجات واضحة، فبدلًا من أن يظهر لنا العدد 50000 مثلًا، نريد أن يخرج A:50000.

وللقيام بذلك، سنستخدم عملية وصل النصوص || وبدلًا من أن نطلب في الاستعلام إخراج ناتج العملية SUM(salary) سنطلب في الاستعلام العبارة التالية:

```
LEFT(last_name,1) || ':' || SUM(salary)
```

وبذلك يكون لدينا الاستعلام التالي:

```
hsoubguide2=#  SELECT last_name,salary,department,
hsoubguide2=#  LEFT(last_name,1) || ':' || SUM(salary) hsoubguide2=#
OVER (PARTITION BY LEFT(last_name,1))
hsoubguide2=#  AS Salary_sum_by_letter
hsoubguide2=#  FROM employee_view;
```

last_name	salary	department	salary_sum_by_letter
Adams	50000	Sales	A:50000
Jones	45000	Accounting	J:85000
Johnson	40000	Marketing	J:85000
Smith	55000	Sales	S:55000
Williams	37000	Accounting	W:37000

5. 3. 3. دالة النافذة rank

يمكننا استخدام دالة النافذة rank لإنتاج رقم ترتيبي للأسطر ضمن التقسيم الذي قمنا به، فلو أردنا مثلًا إنتاج رقم لترتيب الموظفين حسب رواتبهم تصاعديًّا، نكتب الاستعلام التالي:

```
hsoubguide2=# SELECT last_name,salary,department,
hsoubguide2=# rank() OVER (ORDER BY salary)
hsoubguide2=# FROM employee_view;
```

last_name	salary	department	rank
Williams	37000	Accounting	1
Johnson	40000	Marketing	2
Jones	45000	Accounting	3
Adams	50000	Sales	4
Smith	55000	Sales	5

ويمكننا إنتاج رقم ترتيبي للموظفين حسب رواتبهم، ولكن لكل قسم على حدة، وذلك باستخدام العبارة PARTITION BY كما يلي:

```
hsoubguide2=# SELECT last_name,salary,department,
hsoubguide2=# rank() OVER (PARTITION BY department ORDER BY salary)
hsoubguide2=# FROM employee_view;
```

last_name	salary	department	rank
Williams	37000	Accounting	1
Jones	45000	Accounting	2
Johnson	40000	Marketing	1
Adams	50000	Sales	1
Smith	55000	Sales	2

نريد الآن إيجاد الموظفين الأعلى راتبًا في كل قسم، ولذلك سننشئ عارضًا جديدًا، للحفاظ على الجدول الناتج عن الاستعلام السابق ولكن سنخزّنه بترتيب معكوس، بحيث يكون الموظف الأعلى راتبًا له قيمة rank تساوي 1 كما يلي:

```
hsoubguide2=# CREATE OR REPLACE VIEW employee_rank AS
hsoubguide2-# SELECT last_name,salary,department,
hsoubguide2-#  rank() OVER (PARTITION BY department ORDER BY salary
DESC)
hsoubguide2-#  FROM employee_view;
CREATE VIEW
```

أصبح بإمكاننا الآن تنفيذ استعلامات على الجدول الافتراضي الجديد employee_rank كما يلي:

```
hsoubguide2=# SELECT * FROM employee_rank WHERE rank=1;
```

last_name	salary	department	rank
Jones	45000	Accounting	1
Johnson	40000	Marketing	1
Smith	55000	Sales	1

يمكن استعراض توثيق Postgres للمزيد من التعمق في دوال النوافذ.

5. 4. الفهارس Indexes

الفهرس Index هو بنية محددة تنظّم مرجعًا للبيانات مما يجعلها أسهل في البحث، وبكون الفهرس في Postgres نسخةً من العنصر المُراد فهرسته مع مرجع إلى موقع البيانات الفعلي.

تستخدم Postgres عند استعلام البيانات الفهارس إن كانت متاحة أو ستجري مسحًا متتاليًا (sequential scan)، تبحث فيه عبر جميع البيانات المستعلم عنها قبل إرجاع نتيجة.

يحقق استخدام الفهارس سرعة كبيرة في البحث والاستعلام، فإضافة عمود فهرس إلى الجدول ستسرِّع غالبًا الوصول إلى البيانات، إلا أنّه بالمقابل سيؤدي إلى تبطيء إدخال البيانات، وذلك لأن عملية الإدخال ستتضمن كتابة بيانات في مكانين مختلفين مع المحافظة على ترتيب الفهرس ضمن البيانات السابقة.

يجب الانتباه إلى نوع البيانات المستخدمة كفهرس، فالفهرسة على أساس الأرقام أو الطابع الزمني فعالة، بينما يُعدّ استخدام النصوص كفهارس غير فعّال.

لنأخذ مثالا عمليًا بوضع فهرس للجدول التالي:

employees

id	last_name	salary
1	Jones	45000
2	Adams	50000
3	Johnson	40000
4	Williams	37000
5	Smith	55000

```
CREATE INDEX idx_salary ON employees(salary);
```

يُمكن استخدام عدة أعمدة كفهارس في الوقت نفسه، فإن كنت تقوم بترشيح نتائج الجداول باستخدام أعمدة محددة، فيمكنك إنشاء فهارس منها كما يلي:

```
CREATE INDEX idx_salary ON employees(last_name, salary);
```

5. 4. 1. نصائح عند استخدام الفهارس

أ. إنشاء الفهارس آنيًّا

عندما تنشئ Postgres الفهارس (كما في أنظمة قواعد البيانات الأخرى)، تقفا الجدول أثناء بناء الفهرس له، وهذا يمنع إضافة المزيد من البيانات إلى الجدول. فإن كانت كمية البيانات صغيرة

فإن العملية لن تستغرق الكثير من الوقت، ولكنّها ستسبب إعاقةً كبيرة للعمل على هذا الجدول إن كانت كمية البيانات كبيرة جدًا ومن ثمّ فستكون مدة قفل الجدول أطول.

ذلك يعني أنه للحصول على تحسين في كفاءة قاعدة البيانات لا بدّ من إيقاف العمل على قاعدة البيانات لمدة محددة، على الأقل للجدول الذي يتم إنشاء الفهرس له. إلا أن Postgres تتيح إمكانية إنشاء الفهرس دون قفل الجدول، وذلك باستخدام `CREATE INDEX CONCURRENTLY` كما في المثال التالي:

```
CREATE INDEX CONCURRENTLY idx_salary ON employees(last_name, salary);
```

ب. عندما يكون الفهرس أذكى منك

إن استخدام الفهرس ليس هو الحل الأسرع دومًا، فمثلًا إن كانت نتيجة الاستعلام ستعيد نسبة كبيرة من بيانات الجدول، فإن المرور على جميع بيانات الجدول أسرع في هذه الحال من استخدام الفهرس، لذا عليك الوثوق بأن Postgres ستقوم بالأمر الصحيح فيما يتعلق بسرعة الاستعلام سواء كان استخدام الفهرس أو غيره.

5. 4. 2. فهرسة النوع JSONB

إن طريقة الفهرسة الأكثر مرونة وقوة هي استخدام فهرس `GIN`، حيث يقوم `GIN` بفهرسة كل عمود ومفتاح ضمن مستند JSONB، ويمكن إضافته كما في المثال التالي:

```
CREATE INDEX idx_products_attributes ON products USING GIN
(attributes);
```

أ. المفاتيح الخارجية والفهارس

في بعض أنظمة الربط العلائقي للكائنات Object relational mapping، يؤدي إنشاء مفتاح خارجي (Foreign Key) إلى إنشاء فهرس (index) أيضًا، وننوه هنا إلى أن Postgres لا تنشئ تلقائيًا فهرسًا عند إنشاء مفتاح خارجي، فهي خطوة منفصلة عليك الانتباه إليها عندما لا تستخدم ORM.

5.5. خلاصة الفصل

نجحنا في هذا الفصل بجعل استعلاماتنا أقصر وأسهل قراءةً بفضل العرض View وتعابير الجداول الشائعة CTE، كما تعرفنا على دوال النوافذ Window Functions التي تسمح بالقيام بعمليات تجميعية تحتاج إلى لغة برمجة في العادة، كما تطرقنا إلى الفهارس وكيفية إنشائها.

6. أنواع بيانات خاصة في قواعد بيانات Postgres

تتميز Postgres بإضافة عدة أنواع بيانات مميزة خاصة بها، سنتحدث عنها في هذا الفصل، وهي النوع Arrays (المصفوفات) والنوع Hstore والنوع JSONB وهي تساهم بشكل أساسي بالسماح بتخزين هيكل بيانات أكبر من مجرد قيمة واحدة في العمود في الجدول، كما سنستخدم الأنواع التعدادية ENUM لتحديد قيم مخصصة في أعمدة جداولنا.

6. 1. المصفوفات (Arrays)

تتيح Postgres تخزين بيانات على شكل مصفوفات متغيرة الطول ضمن عمود واحد، حيث يمكن أن يكون نوع المصفوفات من الأنواع الأساسية، أو نوعًا جديدًا يحدده المستخدم أو من الأنواع التعدادية (enumerated).

لتحديد عمود ما لتخزين مصفوفة، نضع قوسين `[]` بعد اسم النوع كما يلي:

```
hsoubguide=# CREATE TABLE hsoub_team
hsoubguide-# (
hsoubguide(# team_name text,
hsoubguide(# team_members text[]
hsoubguide(# );

CREATE TABLE
```

يُنشئ الأمر السابق جدولًا اسمه hsoub_team له عمودان، أحدهما نص نخزّن فيه اسم الفريق، وعمودًا آخر team_members يخزن مصفوفة أحادية البعد لتخزين أسماء أعضاء الفريق.

6. 1. 1. إدخال قيم المصفوفات

```
hsoubguide=# INSERT INTO hsoub_team
hsoubguide-# VALUES
hsoubguide-# ('postgres_team',
hsoubguide(# '{"mostafa","jamil","abood"}'
hsoubguide(# );

INSERT 0 1
```

لاحظ أن السلاسل النصية بداخل المصفوفة تكون محصورة بعلامات تنصيص مزدوجة، وذلك لأن قيمة المصفوفة كاملة هي التي تُكتب بين علامات تنصيص مفردة.

سيُظهِر استعلام الجدول ما يلي:

```
hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
postgres_team	{mostafa,jamil,abood}

كما يمكن بناء المصفوفات بطريقة ثانية، عن طريق استخدام باني المصفوفات (Constructor) كما يلي:

```
hsoubguide=# INSERT INTO hsoub_team
hsoubguide-# VALUES
hsoubguide-# ('C++ team',
hsoubguide(# ARRAY['mostafa','yougharta']
hsoubguide(# );
INSERT 0 1
```

عند استخدام باني المصفوفات يتم حصر السلاسل النصية بعلامات تنصيص مفردة، ولا شك أن هذه الطريقة أكثر وضوحًا وأسهل مقروئيّة.

يُعطي استعلام الجدول الآن ما يلي:

```
hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
postgres_team	{mostafa,jamil,abood}
C++ team	{mostafa,yougharta}

ولكن لماذا نستخدم المصفوفات؟ ألم يكن بإمكاننا تخزين محتويات المصفوفة كلها كنص عادي؟ إن ميزة المصفوفات تأتي من القدرة على اختيار عنصر محدد من المصفوفة والوصول إليه، كما سنرى في الفقرة التالية.

6. 1. 2. الوصول إلى عناصر من المصفوفة

يمكن اختيار عناصر المصفوفة عن طريق رقم العنصر للوصول إلى عنصر مفرد، أو باستخدام المجال (من العنصر: إلى العنصر) كما يلي:

```
hsoubguide=# SELECT * FROM hsoub_team WHERE team_members[2] =
'jamil';
```

team_name	team_members
postgres_team	{mostafa,jamil,abood}

```
hsoubguide=# SELECT * FROM hsoub_team WHERE team_members[2:3] =
ARRAY['jamil','abood'];
```

team_name	team_members
postgres_team	{mostafa,jamil,abood}

```
hsoubguide=# SELECT team_members[2:3] FROM hsoub_team ;
```

team_members
{jamil,abood}
{yougharta}

> يجب الانتباه إلى أن ترقيم عناصر المصفوفة يبدأ من 1 وليس من 0 كما في بعض لغات البرمجة.

6. 1. 3. تعديل قيم عناصر المصفوفات

يمكن تعديل قيمة عنصر واحد في المصفوفة، كما يمكن تعديل المصفوفة كاملةً، أو مجالًا محدّدًا منها. فلتعديل عنصر واحد في المصفوفة، نستخدم الوصول إلى العنصر المُراد تعديله كما يلي:

```
hsoubguide=# UPDATE hsoub_team SET team_members[3]='new_member';

UPDATE 2

hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
postgres_team	{mostafa,jamil,new_member}
C++ team	{mostafa,yougharta,new_member}

يمكن الوصول إلى عنصر خارج حدود المصفوفة كما في المثال السابق، ولكن يمكن كذلك تجاوز هذا الحد لتعديل العنصر رقم 5 مثلًا في مصفوفة من 3 عناصر فقط، عندها يتم ملء العناصر الفارغة بقيمة NULL كما في المثال التالي:

```
hsoubguide=# UPDATE hsoub_team SET team_members[5]='new_member2';

UPDATE 2

hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
postgres_team	{mostafa,jamil,new_member,NULL,new_member2}
C++ team	{mostafa,yougharta,new_member,NULL,new_member2}

أما لتعديل المصفوفة كاملةً فلا نستخدم الوصول إلى عنصر مفرد، بل نقوم بتغيير قيمة العمود كاملةً:

```
hsoubguide=# UPDATE hsoub_team SET team_members = ARRAY ['a','b']
WHERE team_name ='C++ team';

UPDATE 1

hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
postgres_team	{mostafa,jamil,new_member,NULL,new_member}
C++ team	{a,b}

كما يمكن تغيير قيمة مجالٍ من المصفوفة كما يلي:

```
hsoubguide=# UPDATE hsoub_team SET team_members[2:3] = ARRAY['x','y']
WHERE team_name LIKE 'postgres%';

UPDATE 1

hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
C++ team	{a,b}
postgres_team	{mostafa,x,y,NULL,new_member}

> يجب أن يكون حجم المجال المُستبدل مساويًا أو أصغر من طول المصفوفة الجديدة، فلو حاولنا استبدال مجالٍ بمجالٍ أصغر منها سيظهر الخطأ التالي:

```
hsoubguide=# UPDATE hsoub_team SET team_members[2:4] = ARRAY['one_
member']  WHERE team_name LIKE 'postgres%';

ERROR:  source array too small
```

أما إن كان حجم المصفوفة الجديدة أكبر من الأصلية، فيتم أخذ عدد من العناصر مساوٍ للمجال الأصلي، كما يلي:

```
hsoubguide=# UPDATE hsoub_team SET team_members[2:4] = ARRAY['one_
member','a','b','c','d','e']  WHERE team_name LIKE 'postgres%';

UPDATE 1
hsoubguide=# SELECT * FROM hsoub_team ;
```

team_name	team_members
C++ team	{a,b}
postgres_team	{mostafa,one_member,a,b,new_member}

6. 1. 4. البحث ضمن المصفوفات

للبحث عن عنصر معين ضمن المصفوفة نستخدم الكلمة المفتاحية ANY كما يوضح المثال التالي:

```
hsoubguide=# SELECT * FROM hsoub_team WHERE 'mostafa' = ANY(team_
members);
```

team_name	team_members
postgres_team	{mostafa,one_member,a,b,new_member}

ويمكن البحث للتحقق من كون كل قيم المصفوفة تطابق قيمة معينة باستخدام الكلمة ALL.

```
hsoubguide=# INSERT INTO hsoub_team
hsoubguide-# VALUES
hsoubguide-# ('team1',
hsoubguide(# ARRAY['programmer1','programmer1','programmer1']
hsoubguide(# );

INSERT 0 1

hsoubguide=# SELECT * FROM hsoub_team WHERE 'programmer1' = ALL(team_
members);
```

team_name	team_members
team1	{programmer1,programmer1,programmer1}

كما يمكن استخدام ALL مع تحديد المجال ضمن المصفوفة كما يلي:

```
hsoubguide=# INSERT INTO hsoub_team
VALUES
('team7',
ARRAY['programmer1','programmer1','another_programmer']
);

INSERT 0 1

hsoubguide=# SELECT * FROM hsoub_team WHERE 'programmer1' = ALL(team_
members[1:2]);
```

team_name	team_members
team1	{programmer1,programmer1,programmer1}
team7	{programmer1,programmer1,another_programmer}

6. 2. أنواع البيانات التعدادية (Enumerated Data Types)

توفر Postgres نوع بيانات تعدادية enums تُستخدَم لحصر قيم عمود ما في مجموعة قيم محددة مسبقًا من القيم.

سنعمل في المثال التالي على حصر قيم العمود contact_method بمجموعة القيم Email و SMS و Phone، وذلك عن طريق تعريف التعداد كما يلي:

```
hsoubguide=# CREATE TYPE e_contact_method AS ENUM (
hsoubguide(# 'Email',
hsoubguide(# 'Sms',
hsoubguide(# 'Phone');

CREATE TYPE
```

ومن ثم نرفق نوع التعداد الجديد بالعمود الذي نريد حصر قيمه كما يلي:

```
hsoubguide=# CREATE TABLE contact_method_info (
hsoubguide(# contact_name text,
hsoubguide(# contact_method e_contact_method,
hsoubguide(# value text
hsoubguide(# );

CREATE TABLE
```

6. 2. 1. استخدام الأنواع التعدادية

سنحاول في هذا المثال إدخال قيم في العمود الذي يستخدم النوع التعدادي، لنرى ما يحصل عند الخروج عن القيم المحددة مسبقًا:

```
hsoubguide=# INSERT INTO contact_method_info
hsoubguide-# VALUES ('Jamil', 'Email', 'jamil@mail.com');
INSERT 0 1
hsoubguide=# SELECT * FROM contact_method_info ;
```

contact_name	contact_method	value
Jamil	Email	jamil@mail.com

لا يمكن إدراج قيمة للعمود contact_method غير موجودة سلفًا ضمن التعداد e_contact_method وسيظهر خطأ كما في المثال التالي:

```
hsoubguide=# INSERT INTO contact_method_info VALUES ('Jamil', 'Fax',
'4563456');

ERROR:  invalid input value for enum e_contact_method: "Fax"
LINE 1: INSERT INTO contact_method_info VALUES ('Jamil', 'Fax',
'456...
```

6. 3. عرض وتعديل قيم التعداد

يمكننا عرض قائمة القيم في التعداد بالاستعانة بالجداول pg_type و pg_enum التي تُخزّن إعدادات الأنواع والتعدادات، وذلك كما يلي:

```
hsoubguide=# SELECT pg_type.typname, pg_enum.enumlabel
hsoubguide-# FROM pg_type,pg_enum
hsoubguide-# WHERE pg_type.oid = pg_enum.enumtypid;
```

typname	enumlabel
e_contact_method	Email
e_contact_method	Sms
e_contact_method	Phone

كما يمكن إضافة قيم للتعدادات الموجودة مسبقًا كما يلي:

```
hsoubguide=# ALTER TYPE e_contact_method
hsoubguide-# ADD VALUE 'Facebook' AFTER 'Phone';

ALTER TYPE

hsoubguide=# SELECT pg_type.typname, pg_enum.enumlabel
hsoubguide-# FROM pg_type,pg_enum
hsoubguide-# WHERE pg_type.oid = pg_enum.enumtypid;
```

typname	enumlabel
e_contact_method	Email
e_contact_method	Sms
e_contact_method	Phone
e_contact_method	Facebook

يتم حفظ ترتيب القيم داخل التعدادات بنفس الترتيب الذي تم إدخال القيم به، ولكن يمكن إدخال قيم جديدة وتحديد مكانها قبل قيمة معينة أو بعدها، كما في المثال التالي:

```
hsoubguide=# ALTER TYPE e_contact_method
hsoubguide-# ADD VALUE 'Twitter' BEFORE 'Sms';

ALTER TYPE

hsoubguide=# SELECT pg_type.typname,pg_enum.enumlabel,pg_enum.
enumsortorder
hsoubguide-# FROM pg_type, pg_enum
hsoubguide-# WHERE pg_type.oid = pg_enum.enumtypid
hsoubguide-# ORDER BY pg_enum.enumsortorder;
```

typname	enumlabel	enumsortorder
e_contact_method	Email	1
e_contact_method	Twitter	1.5
e_contact_method	Sms	2
e_contact_method	Phone	3
e_contact_method	Facebook	4

لا تسمح Postgres بإزالة قيم من التعدادات ولا بتغيير ترتيبها، وللقيام بذلك علينا حذف التعداد عن طريق التعليمة DROP TYPE، ولكن انتبه إلى أنه لا يمكنك حذف التعداد إذا كان هناك أعمدة تستخدم هذا النوع، لكن يمكنك حذف التعداد مع جميع الأعمدة المرتبطة به باستخدام الكلمة المفتاحية CASCADE، ولتوضيح ذلك لدينا المثال التالي:

```
hsoubguide=# DROP TYPE e_contact_method CASCADE;

NOTICE:  drop cascades to column contact_method of table contact_
method_info
DROP TYPE

hsoubguide=# SELECT pg_type.typname, pg_enum.enumlabel
FROM pg_type,pg_enum
WHERE pg_type.oid = pg_enum.enumtypid;
```

typname	enumlabel

```
hsoubguide=# SELECT * FROM contact_method_info ;
```

contact_name	value
Jamil	jamil@mail.com

6. 4. نوع البيانات HStore

النوع HStore هو أسلوب تخزين (مفتاح، قيمة) ضمن Postgres يُستخدَم مثل القاموس (يشبه النوع Hash في روبي والنوع Object أو Map في جافاسكربت)، لكنه مخصص لعمود في سطر ما.

6. 4. 1. تفعيل HStore

لتفعيل HStore في قاعدة البيانات قم بتنفيذ الأمر التالي:

```
hsoubguide=# CREATE EXTENSION hstore;

CREATE EXTENSION
```

6. 4. 2. إنشاء عمود HStore

لإنشاء حقل في جدول ذو نوع بيانات HStore استخدم HStore كنوع للعمود ببساطة كما يلي:

```
hsoubguide=# CREATE TABLE students (
hsoubguide-# id serial PRIMARY KEY,
hsoubguide-# name varchar,
hsoubguide-# attributes hstore
hsoubguide-# );

CREATE TABLE
```

6. 4. 3. إدخال بيانات من نوع HStore

لإدخال البيانات عليك كتابتها ضمن علامات تنصيص مفردة. الفرق في HStore هو بنية إضافية لتوضيح كيفية إنشاء القاموس:

```
hsoubguide=# INSERT INTO students (name,attributes) VALUES(
hsoubguide(# 'mostafa',
hsoubguide(# 'nickname => mayesh,
hsoubguide'# grade => 12,
hsoubguide'# school => "Hsoub Academy",
hsoubguide'# weight => 82'
hsoubguide(# );

INSERT 0 1
```

إن استخدام النوع HStore كان من أوائل المحاولات في الخروج عن هيكلة قواعد البيانات، ومن ثم فهو من أوائل أنواع NoSQL التي ظهرت، فليس هناك محددات للعناصر التي يمكننا تخزينها في HStore.

6. 4. 4. الوصول إلى بيانات من نوع HStore

يمكننا استخدام العملية -> للوصول إلى عناصر في داخل العنصر من النوع Hstore، وذلك بتحديد اسم المفتاح كما يلي:

```
hsoubguide=# SELECT name,attributes->'school'
hsoubguide=# FROM students
hsoubguide=# WHERE attributes->'nickname' = 'mayesh';
```

name	?column?
mostafa	Hsoub Academy

لتغيير اسم العمود بدلًا من ?column? استخدم AS كما يلي:

```
hsoubguide=# SELECT name,attributes->'school' AS school
FROM students
WHERE attributes->'nickname' = 'mayesh';
```

name	school
mostafa	Hsoub Academy

بما أنه لا توجد قيود على المفاتيح المخزنة بداخل النوع Hstore فإنه من الممكن أن يوجد مفتاحُ ما في سطر ما، ولا يوجد في سطر آخر، وعندها يُتعامل معه على أنه موجود بقيمة خالية، كما في المثال التالي:

```
hsoubguide=# INSERT INTO students (name,attributes) VALUES(
hsoubguide(# 'Jamil',
hsoubguide(# 'grade => 13
hsoubguide'# ,
hsoubguide'# weight => 72');

INSERT 0 1

hsoubguide=# SELECT * FROM students;
```

id	name	attributes
1	mostafa	"grade"=>"12", "school"=>"Hsoub Academy", "weight"=>"82", "nickname"=>"mayesh"
2	Jamil	"grade"=>"13", "weight"=>"72"

```
hsoubguide=# SELECT name,attributes->'school' AS school FROM
students;
```

name	school
mostafa	Hsoub Academy
Jamil	

يتيح استخدام Hstore مرونة عالية في قاعدة البيانات، ولكنها أصبحت تقنية قديمة موازنةً بتقنية النوع JSONB التي سنتحدث عنها في الفقرة التالية.

6. 5. بيانات بصيغة JSON

ظهر استخدام JSON في postgres بدءًا من الإصدار 9.2، لكن الإصدار الحقيقي ظهر باسم JSONB في postgres 9.4.

JSNOB هي الصيغة الثنائية لتعابير JSON للتخزين الدائم، فهي أكثر كفاءة في التخزين والفهرسة.

6. 5. 1. إنشاء أعمدة JSONB

لإنشاء أعمدة من النوع JSONB حدد النوع JSONB ضمن تعليمة CREATE TABLE كما يلي:

```
hsoubguide=# CREATE TABLE school (
hsoubguide(#     id serial PRIMARY KEY,
hsoubguide(#     name varchar,
hsoubguide(#     attributes JSONB
hsoubguide(# );

CREATE TABLE
```

6. 5. 2. إدخال البيانات من النوع JSONB

يُفترض أن يكون إدخال عمود يحتوي على صيغة JSON سهلًا ومباشرًا، ونوضّحه بالمثال التالي:

```
hsoubguide=# INSERT INTO school (name,attributes) VALUES (
hsoubguide(#     'Hsoub', '{
hsoubguide'#     "manager" : "Agha",
hsoubguide'#     "classes" : 7,
hsoubguide'#     "teachers": 12}'
hsoubguide(# );

INSERT 0 1

hsoubguide=# SELECT * FROM school;
```

id	name	attributes
1	Hsoub	{"classes": 7, "manager": "Agha", "teachers": 12}

6. 5. 3. الوصول إلى قيم المفاتيح في JSONB

يمكننا استخدام العملية -> للوصول إلى القيم عن طريق أسماء مفاتيحها، كما يلي:

```
hsoubguide=# SELECT name,attributes->'manager' AS manager FROM
school;
```

name	manager
Hsoub	"Agha"

ويمكننا استخدامها داخل شروط الترشيح كذلك:

```
hsoubguide=# SELECT * FROM school WHERE attributes->'classes' = '7';
```

id	name	attributes
1	Hsoub	{"classes": 7, "manager": "Agha", "teachers": 12}

هناك العديد من الأمور المتقدمة التي يمكن القيام بها في JSONB، ولكن يجب أن نعرف أنها مخصصة لتخزين العناصر والوصول إليها، ولكنها لم تصمم لتعديل العناصر بعد تخزينها.

6. 6. التعامل مع التاريخ والوقت

يمكن تخزين التاريخ والوقت في Postgres باستخدام عدة أنواع، كما تتيح العديد من الطرق للتعامل المرن مع التواريخ والأوقات، النوع الأساسي لتخزين التاريخ هو DATE ولتخزين الوقت TIME، ويمكننا حفظ التاريخ مع الوقت باستخدام النوع TIMESTAMP.

سننشئ جدولًا جديدًا لتعلم التعامل مع هذه الأنواع الجديدة:

```
hsoubguide=# CREATE TABLE date_example(
hsoubguide(#     mydate DATE NOT NULL DEFAULT CURRENT_DATE,
hsoubguide(#     mytime TIME NOT NULL DEFAULT CURRENT_TIME,
hsoubguide(#     mytimestamp TIMESTAMP NOT NULL DEFAULT NOW()
hsoubguide(# );

CREATE TABLE

hsoubguide=# SELECT * FROM date_example;
```

mydate	mytime	mytimestamp

أنشأنا في الجدول السابق ثلاث أعمدة، لكل منها قمنا بمنع تخزين القيمة الخالية عن طريق NOT NULL، واستخدمنا القيم الافتراضية التالية:

- التاريخ الحالي، باستخدام CURRENT_DATE
- الوقت الحالي، باستخدام CURRENT_TIME
- التاريخ والوقت الحالي، باستخدام التابع NOW()

سندخل الآن سطرًا مميزًا، حيث لن نحدد فيه أي قيمة، بل سنستخدم التوجيه DEFAULT VALUES كي يتم إدخال جميع القيم الافتراضية كما يلي:

```
hsoubguide=# INSERT INTO date_example DEFAULT VALUES;

INSERT 0 1

hsoubguide=# SELECT * FROM date_example;
```

mydate	mytime	mytimestamp
2020-08-22	01:24:08.241482	2020-08-22 01:24:08.241482

كما يظهر استعلام الجدول السابق، يتم إظهار التاريخ بالتنسيق yyyy-mm-dd.

6. 6. 1. استخدام لتابع NOW()

يمكننا استخدام التابع NOW() للاستعلام عن التاريخ والوقت كما يلي:

```
hsoubguide=# SELECT NOW();
```

now
2020-08-22 01:26:48.875054+03

```
hsoubguide=# SELECT NOW()::DATE;
```

now
2020-08-22

```
hsoubguide=# SELECT NOW()::TIME;
```

now
01:26:55.072126

6. 6. 2. استخدام التابع TO_CHAR()

كما يمكننا تحديد النسق الذي نرغب بعرض التاريخ فيه من خلال التابع TO_CHAR() كما يلي:

```
hsoubguide=# SELECT TO_CHAR(NOW()::DATE,'dd ++ mm ++ yyyy');
```

to_char
22 ++ 08 ++ 2020

```
hsoubguide=# SELECT TO_CHAR(NOW()::DATE,'Month dd/mm/yyyy');
```

to_char
August 22/08/2020

يمكننا استخدام التابع AGE() لحساب فارق التاريخ وإظهاره بنفس تنسيق التاريخ كما يلي:

```
hsoubguide=# SELECT AGE(CURRENT_DATE,'25-07-1993');
```

age
27 years 28 days

كما يمكننا استخدام عملية الطرح للتواريخ - للحصول على الفرق بالأيام كما يلي:

```
hsoubguide=# SELECT CURRENT_DATE-'25-07-1993' AS days;
```

days
9890

للحصول على فارق الوقت كذلك علينا تحديد نوع البيانات التي نقوم بطرحها على أنها بيانات وقت:

```
hsoubguide=# SELECT time '10:57:18' - time '02:17:17' AS result;
```

result
08:40:01

> انتبه إلى إضافة النوع time قبل الوقت المطروح، وإلا سيظهر الخطأ التالي:

```
hsoubguide=# SELECT '10:57:18' - '02:17:17' AS result;

ERROR:  operator is not unique: unknown - unknown
LINE 1: SELECT '10:57:18' - '02:17:17' AS result;
                          ^
HINT:  Could not choose a best candidate operator. You might need to
add explicit type casts.
```

6. 6. 3. استخدام التابع EXTRACT()

لعل أحد أكثر التوابع فائدةً هو التابع EXTRACT() كما يمكننا استخلاص السنة والشهر واليوم من التاريخ كما يلي:

```
hsoubguide=# SELECT EXTRACT(YEAR FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
2016

```
hsoubguide=# SELECT EXTRACT(MONTH FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
12

```
hsoubguide=# SELECT EXTRACT(DAY FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
31

```
hsoubguide=# SELECT EXTRACT(HOUR FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
13

```
hsoubguide=# SELECT EXTRACT(MINUTES FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
30

```
hsoubguide=# SELECT EXTRACT(SECONDS FROM TIMESTAMP '2016-12-31
13:30:15');
```

date_part
15

6. 6. 4. تخزين المدد الزمنية INTERVAL

تتيح كذلك Postgres استخدام نوع مميز لتخزين المدة الزمنية INTERVAL، وسننشئ جدولًا صغيرًا لتعلّم كيفية استخدامه:

```
hsoubguide=# CREATE TABLE date_example2(
hsoubguide(#     myinterval INTERVAL
hsoubguide(# );

CREATE TABLE
```

يمكننا تخزين مدة زمنية كسنوات كما يلي:

```
hsoubguide=# INSERT INTO date_example2(myinterval) VALUES ('2
years');

INSERT 0 1
```

أو كسنوات وأشهر:

```
hsoubguide=# INSERT INTO date_example2(myinterval) VALUES ('2 years 3
months');

INSERT 0 1
```

كما يمكن تخزين مدة زمنية في الماضي باستخدام الكلمة ago:

```
hsoubguide=# INSERT INTO date_example2(myinterval) VALUES ('2 years 3
months ago');

INSERT 0 1
```

يمكننا كذلك تخزين التاريخ مع الوقت كمدة زمنية:

```
hsoubguide=# INSERT INTO date_example2(myinterval) VALUES ('2 years 3
months 5 days 33 minutes');

INSERT 0 1
```

```
hsoubguide=# INSERT INTO date_example2(myinterval) VALUES ('2 years 3
months 5 days 33 minutes ago');

INSERT 0 1
```

وعند الاستعلام عن كل هذه المدد الزمنية سنجد أنها مخزنة كما أدخلناها، باستثناء المدد التي أضفنا لها كلمة ago قد أضيف لها إشارة السالب:

```
hsoubguide=# SELECT * FROM date_example2;
```

myinterval
2 years
2 years 3 mons
-2 years -3 mons
2 years 3 mons 5 days 00:33:00
-2 years -3 mons -5 days -00:33:00

تكمن فائدة استخدام المدد الزمنية عند الحاجة إلى الرجوع بتاريخ معين إلى مدة زمنية محددة كما يلي:

```
hsoubguide=# SELECT NOW() - INTERVAL '1 year 2 months 3 days 4 hours
5 minutes 6 seconds' AS "1 year,2 months,3 days,04h:05m:06s ago";
```

1 year,2 months,3 days,04h:05m:06s ago
2019-06-18 22:36:02.945923+03

6. 7. خلاصة الفصل

تتميز Postgres عن الأصل SQL بوجود أنواع بيانات خاصة مثل Arrays و JSONB و HSTORE التي تسمح بتخزين مجموعات مرتبة من البيانات ضمن عمود واحد، وقد تعرفنا إلى كل منها في هذا الفصل، مما يعطينا القدرة على تخزين البيانات بطريقة أكثر احترافية باستخدام Postgres، وقد تعرفنا كذلك في هذا الفصل إلى النوع ENUM الذي يحدد القيمة المخزنة ضمن العمود بقيم معرفة مسبقًا.

7. إدارة النسخ الاحتياطي في قواعد بيانات Postgres

سنتعرف في هذا الفصل على كيفية أخذ نسخة احتياطية عن قاعدة البيانات الخاصة بنا، ثم استعادتها، كما سنتعرف على الأمر `\copy` وكيف نستخدمه لتحديد نمط البيانات في النسخة الاحتياطية.

7. 1. النسخ الاحتياطي والاستعادة

النسخ الاحتياطي هو أخذ نسخة كاملة عن مخططات الجداول وبيانات قاعدة البيانات، أما الاستعادة فهي القدرة على استخدام هذه البيانات التي تم نسخها احتياطيًا وتنزيلها إلى قاعدة البيانات الخاصة بك أو قاعدة بيانات أخرى.

> تتم عملية النسخ الاحتياطي والاستعادة على قاعدة بيانات بأكملها أو على جدول بأكمله وليس الهدف منها استخلاص أجزاء من البيانات فقط، ففي تلك الحالة نستخدم النسخ (copy) الذي سنتحدث عنه لاحقًا.

7. 1. 1. إجراء النسخ الاحتياطي

لأخذ نسخة احتياطية من قاعدة البيانات نستخدم الأداة pg_dump، وعلينا تحديد بعض الإعدادات لتحديد نتيجة عملية النسخ، ومنها:

- هل نريد أن تكون النتيجة على شكل نص عادي (قابل للقراءة ولكنه كبير الحجم) أو بصيغة ثنائية (غير قابلة للقراءة صغيرة الحجم) أو بصيغة مضغوطة tarball (مثالي للقيام بعملية الاستعادة).
- هل نرغب بنسخ كل قاعدة البيانات أم مخططات (schema) أو جداول معينة.

قبل البدء بالنسخ الاحتياطي قد ترغب باستعراض قواعد البيانات المخزنة لديك، باستخدام الأمر التالي:

```
bash-4.2$ psql -l
```

Name	Owner	Encoding	Collate	Ctype	Access privileges
hsoubguide	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	
hsoubguide2	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	
postgres	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	
template0	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +
					postgres=CTc/ postgres
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +
					postgres=CTc/ postgres

لاحظ أن العمل يتم ضمن صدفة bash في هذه الفقرة.

يمكنك القيام بعملية النسخ الاحتياطي باستخدام التعليمة التالية:

```
pg_dump database_name_here > database.sql
```

فمثلًا سنأخذ نسخة احتياطية من قاعدة البيانات hsoubguide التي كنا نعمل عليها خلال هذا الكتاب كما يلي:

```
bash-4.2$ pg_dump hsoubguide > /tmp/hsoubguide.sql
```

خزَّنَّا نسخة قاعدة البيانات في ملف اسمه hsoubguide.sql ضمن المجلد tmp/، يمكنك مراجعة دليل كيف تفهم هيكلية نظام الملفات في لينكس للتعرف أكثر على دور المجلد tmp وغيره في نظام لينكس.

تنتج التعليمة السابقة نسخة عن قاعدة البيانات على شكل نص عادي، سنستعرض أول 50 سطرًا منها من باب الاطلاع:

```
bash-4.2$ cat /tmp/hsoubguide.sql | head -50
--
-- PostgreSQL database dump
--

-- Dumped from database version 12.3
-- Dumped by pg_dump version 12.3

SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: hstore; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS hstore WITH SCHEMA public;
```

```
--
-- Name: EXTENSION hstore; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION hstore IS 'data type for storing sets of (key,
value) pairs';


SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: basket_a; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.basket_a (
    id integer NOT NULL,
    fruit character varying(100) NOT NULL
);


ALTER TABLE public.basket_a OWNER TO postgres;

--
-- Name: basket_b; Type: TABLE; Schema: public; Owner: postgres
```

ولكن بما أن النسخة الاحتياطية مكتوبة بشكلها المقروء، فإن حجمها ليس بالقليل:

```
bash-4.2$ du -h /tmp/hsoubguide.sql

212K    /tmp/hsoubguide.sql
```

بلغ حجم النسخة الاحتياطية 212 كيلو بايت تقريبًا.

لإنشاء نسخة احتياطية أكثر ملاءمة للتخزين الدائم، يمكنك استخدام الضغط لحفظ قاعدة البيانات بشكلها الثنائي:

```
bash-4.2$ pg_dump --format=c hsoubguide > /tmp/hsoubguide.bak

bash-4.2$ du -h /tmp/hsoubguide.bak

76K     /tmp/hsoubguide.bak
```

تمكّنا من ضغط قاعدة البيانات 2.8 مرات تقريبا، وذلك بسبب استخدام الراية `format` مع الحرف `c` الذي يدل على الكلمة custome.

للتوسع في استخدام التطبيق `pg_dump` يمكنك الرجوع إلى توثيق Postgres الرسمي.

7. 1. 2. استعادة البيانات من نسخة احتياطية

ما رأيك أن نبدأ هذه الفقرة بحركة خطيرة، يمكن أن تؤدي إلى طردك من العمل؟ سنقوم بحذف قاعدة البيانات!

```
bash-4.2$ dropdb hsoubguide
```

في حال لم يكن لديك نسخة احتياطية من قاعدة البيانات، فيمكن أن يكون الأمر السابق هو آخر ما تقوم به في عملك، ولكنك قد أخذت نسخة احتياطية بالفعل باستخدام `pg_dump` في الفقرة السابقة، فلنقم الآن بالاستعادة.

> يجدر بك تعلّم كيفية عمل نسخ احتياطي دوري تلقائي عن طريق مهام cron في لينكس، وقد يفيدك مقال 10 أمثلة لجدولة المهام باستخدام Cron في الأكاديمية.

هناك العديد من الخيارات عند استعادة قاعدة البيانات، ولكن عملية الاستعادة لا يمكنها العمل دون وجود قاعدة بيانات بالفعل ضمن العنقود الخاص بك، لذا عليك إنشاءها ثم الاستعادة إليها كما يلي:

```
bash-4.2$ createdb hsoubguide
bash-4.2$ pg_restore --format=c --dbname=hsoubguide /tmp/hsoubguide.
bak
```

للتوسع في استخدام التطبيق `pg_restore` يمكنك الرجوع إلى توثيق Postgres الرسمي.

7. 1. 3. النسخ (Copy)

يُرفق Postgres بالعديد من الأدوات المساعدة لنقل البيانات أشهرها `pg_dump` و `pg_restore` لأخذ نسخة احتياطية من قاعدة البيانات واستعادتها التي تعرفنا عليها في الفقرة السابقة. كما أن هناك أداة مشابهة بنفس القدر من الأهمية إلا أنها أقل شهرة هي أداة Postgres للنسخ التي تسمح بنسخ البيانات من الجداول في قاعدة البيانات وإليها، وتدعم هذه الأداة عدة أنماط، منها:

- النمط الثنائي
- نمط الجدولة باستخدام tab
- نمط csv، للجدولة باستخدام الفاصلة ,

قد تحتاج إلى هذه الأداة يومًا ما سواءً لتنزيل كتل من البيانات للتجربة، أو القيام ببعض من عمليات ETL، أو حتى لاستخراج البيانات لإرسالها إلى جهة ما.

أ. أمثلة عملية لاستخدام الأداة Copy

سنبين في هذه الفقرة بعض الأمثلة لشرح طريقة استخدام هذه الأداة، فمثلًا لاستخراج جميع الموظفين إلى ملف مُجدول بواسطة tab:

```
hsoubguide=# \copy (SELECT * FROM employees) TO '~/employees.tsv';

COPY 5

hsoubguide=# quit
```

```
bash-4.2$ ls -l

total 52
drwx------. 4 postgres postgres    51 Jun 15 02:22 12
-rw-rw-r--. 1 postgres postgres    75 Jun 30 06:12 employees.tsv
-rw-r--r--. 1 postgres postgres 46429 Jun  5 16:48 example.dump
bash-4.2$ cat employees.tsv

1  Jones 45000
2  Adams 50000
3  Johnson     40000
4  Williams    37000
5  Smith 55000
```

استخراج جميع الموظفين إلى ملف csv:

```
hsoubguide=# \copy (SELECT * FROM employees) TO '~/employees.csv'
WITH (FORMAT CSV);

COPY 5

hsoubguide=# quit

bash-4.2$ ls -l employees.csv

-rw-rw-r--. 1 postgres postgres 75 Jun 30 06:13 employees.csv
bash-4.2$ cat employees.csv

1,Jones,45000
2,Adams,50000
3,Johnson,40000
4,Williams,37000
5,Smith,55000
```

استخراج جميع الموظفين إلى ملف ثنائي (لاحظ علامات التنصيص المزدوجة حول الكلمة Binary):

```
hsoubguide=# \copy (SELECT * FROM employees) TO '~/employees.dat'
WITH (FORMAT "binary");

COPY 5

hsoubguide=# quit

bash-4.2$ ls -l employees.dat

-rw-rw-r--. 1 postgres postgres 161 Jun 30 06:16 employees.dat
bash-4.2$ hexdump employees.dat

0000000 4750 4f43 5950 ff0a 0a0d 0000 0000 0000
0000010 0000 0000 0003 0000 0004 0000 0001 0000
0000020 4a05 6e6f 7365 0000 0400 0000 c8af 0300
0000030 0000 0400 0000 0200 0000 0500 6441 6d61
0000040 0073 0000 0004 c300 0050 0003 0000 0004
0000050 0000 0003 0000 4a07 686f 736e 6e6f 0000
0000060 0400 0000 409c 0300 0000 0400 0000 0400
0000070 0000 0800 6957 6c6c 6169 736d 0000 0400
0000080 0000 8890 0300 0000 0400 0000 0500 0000
0000090 0500 6d53 7469 0068 0000 0004 d600 ffd8
00000a0 00ff
00000a1
```

ولتنزيل البيانات من الملفات إلى جدول، فالأسطر التالية تعاكس العمليات السابقة بالترتيب:

```
\copy employees FROM '~/employees.tsv';
\copy employees FROM '~/employees.csv' WITH CSV;
\copy employees FROM '~/employees.dat' WITH BINARY;
```

سنحذف محتويات الجدول واستعادتها لتجربة الأوامر السابقة كما يلي:

```
hsoubguide=# DELETE FROM employees;

DELETE 5

hsoubguide=# \copy employees FROM '~/employees.csv' WITH CSV;

COPY 5

hsoubguide=# SELECT * from employees;
```

id	last_name	salary
1	Jones	45000
2	Adams	50000
3	Johnson	40000
4	Williams	37000
5	Smith	55000

إن هذه التعليمة `\copy` مهمة خصوصًا عند الحاجة إلى إنشاء الجدول خارج صدفة `psql` عن طريق الكتابة على ملف ما ثم إدخال هذه البيانات مباشرةً إلى جدول في قاعدة البيانات.

ولكن تجدر الإشارة إلى أنه لا يمكن استعادة جدول إلا إذا كان الجدول معرّفًا من قبل في قاعدة البيانات، فكما نرى، فإن البيانات المخزنة في الملفات ليست سوى بيانات الجدول دون تعريف لمخطط الجدول.

7. 2. خلاصة الفصل

إدارة النسخ الاحتياطي لقاعدة البيانات أمرٌ شديد الأهمية، ولن يقدِّر أحد أهميته كما يقدرها من فقد بياناته دون أخذه مسبقًا لنسخة احتياطية، وقد تعرفنا في هذا الفصل على الأدوات المساعدة على ذلك، فاستخدمها بشكل دوري، ولا تعرض بياناتك لخطر الزوال.

8. أساسيات إدارة الذاكرة

سنتعرف في هذا الفصل على الملفات التي تُخزّن فيها قواعد البيانات في Postgres، وكيف نستعرض المساحات التخزينية التي تحجزها الجداول والفهارس.

8. 1. مسارات تخزين البيانات

لعله من البديهي أنه لا بد من وجود مكان لتخزين البيانات لأي قواعد بيانات، وهذا المكان هو ملفات موجودة في مكان ما من الخادوم الذي يشغّل عملية postgres.

لمعرفة مكان تخزين البيانات من داخل صدفة `psql` نستخدم التعليمة التالية:

```
hsoubguide=# SELECT name, setting FROM pg_settings WHERE category =
'File Locations';
```

name	setting
config_file	/var/lib/pgsql/12/data/postgresql.conf
data_directory	/var/lib/pgsql/12/data
external_pid_file	
hba_file	/var/lib/pgsql/12/data/pg_hba.conf
ident_file	/var/lib/pgsql/12/data/pg_ident.conf

تبيّـن مخرجـات التعليمـة السـابقة مسـارات ملفـات الإعـدادات، كمـا تشـير إلـى المسـار الخـاص بالبيانـات data_directory فـإذا انتقلنـا إلـى هـذا المسـار واسـتعرضنا محتوياتـه سـنجد مـا يلـي:

```
-bash-4.2$ cd /var/lib/pgsql/12/data
-bash-4.2$ ls -l

total 64
drwx------. 6 postgres postgres    54 Jun 30 00:16 base
-rw-------  1 postgres postgres    30 Aug 14 14:01 current_logfiles
drwx------. 2 postgres postgres  4096 Aug 14 15:23 global
drwx------. 2 postgres postgres   188 Jun 15 00:30 log
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_commit_ts
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_dynshmem
-rw-------. 1 postgres postgres  4269 Jun  5 16:10 pg_hba.conf
-rw-------. 1 postgres postgres  1636 Jun  5 16:10 pg_ident.conf
drwx------. 4 postgres postgres    68 Aug 14 14:06 pg_logical
drwx------. 4 postgres postgres    36 Jun  5 16:10 pg_multixact
drwx------. 2 postgres postgres    18 Aug 14 14:01 pg_notify
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_replslot
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_serial
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_snapshots
drwx------. 2 postgres postgres     6 Aug 14 14:01 pg_stat
drwx------. 2 postgres postgres    63 Aug 14 15:28 pg_stat_tmp
drwx------. 2 postgres postgres    18 Jun  5 16:10 pg_subtrans
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_tblspc
drwx------. 2 postgres postgres     6 Jun  5 16:10 pg_twophase
-rw-------. 1 postgres postgres     3 Jun  5 16:10 PG_VERSION
drwx------. 3 postgres postgres    92 Jun 29 23:21 pg_wal
drwx------. 2 postgres postgres    18 Jun  5 16:10 pg_xact
-rw-------. 1 postgres postgres    88 Jun  5 16:10 postgresql.auto.
conf
-rw-------. 1 postgres postgres 26632 Jun  5 16:10 postgresql.conf
-rw-------. 1 postgres postgres    58 Aug 14 14:01 postmaster.opts
-rw-------  1 postgres postgres   102 Aug 14 14:01 postmaster.pid
```

يمكنك اكتشاف الكثير عن آلية عمل postgres من الداخل في هذا المسار، ولكننا مهتمون بمعرفة مساحتها في الذاكرة، ولذا سنقوم بكتابة الأمر التالي:

```
-bash-4.2$ du -sh /var/lib/pgsql/12/data

66M /var/lib/pgsql/12/data
```

يُخبرنا ذلك بأن حجم جميع بيانات postgres هي 66 ميغا بايت.

ولكننا قد نحتاج إلى معرفة حجم قاعدة بيانات أو جدول أو فهرس على حدة، ولذلك توفر Postgres طريقة مناسبة لمعرفة هذه المعلومات عن طريق الاستعلام من داخل صدفة `psql` أو عن طريق تعليمات الصدفة `psql` أيضًا.

8.2. معرفة حجم قاعدة البيانات

يمكن معرفة حجم قاعدة البيانات بشكل سهل عن طريق التوجيه `\l+` الذي يعرض قائمة لقواعد البيانات مع أحجامها في العمود Size:

```
hsoubguide=# \l+
```

Name	Owner	Encoding	Collate	Ctype	Access privileges	Size	Tablespace	Description
hsoubguide	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8		9353 kB	pg_default	
postgres	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8		8201 kB	pg_default	default administrative connection datab
ase								
template0	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +	8049 kB	pg_default	unmodifiable empty database
					postgres=CTc/postgres			
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres +	8049 kB	pg_default	default template for new databases
					postgres=CTc/postgres			

كما يمكننا تنفيذ الاستعلام التالي للحصول على حجم قاعدة بيانات محددة، وواحدة هذا الحجم هي البايت:

```
hsoubguide=# SELECT pg_database_size('hsoubguide');
```

pg_database_size
9577327

كما يمكننا استخدام التابع `pg_size_pretty` لإظهار الحجم بواحدة مقروءة للمستخدم مثل kB أو MB كما يلي:

```
hsoubguide=# SELECT pg_size_pretty(pg_database_size('hsoubguide'));
```

pg_size_pretty
9353 kB

8. 3. معرفة حجم الجدول

يمكن استخدام التعليمة `\dt+` من صَدَفة `psql` لتظهر لك كل الجداول مع أحجامها:

```
hsoubguide=# \dt+
```

Schema	Name	Type	Owner	Size	Description
public	basket_a	table	postgres	8192 bytes	
public	basket_b	table	postgres	8192 bytes	
public	departments	table	postgres	8192 bytes	
public	employee_ departments	table	postgres	8192 bytes	
public	employees	table	postgres	8192 bytes	
public	marks	table	postgres	8192 bytes	
public	names	table	postgres	8192 bytes	
public	phones	table	postgres	8192 bytes	
public	products	table	postgres	16 kB	
public	purchase_items	table	postgres	328 kB	
public	purchases	table	postgres	120 kB	

Schema	Name	Type	Owner	Size	Description
public	student	table	postgres	16 kB	
public	table1	table	postgres	8192 bytes	
public	table2	table	postgres	8192 bytes	
public	test_table	table	postgres	8192 bytes	
public	users	table	postgres	16 kB	
public	users2	table	postgres	8192 bytes	

أو يمكنك استخدام الاستعلام التالي للحصول على حجم جدول محدد:

```
hsoubguide=# SELECT pg_size_pretty(pg_relation_size('users'));
```

pg_size_pretty
8192 bytes

8. 4. معرفة حجم الفهرس (index)

يمكن تطبيق التعليمة السابقة لمعرفة حجم الفهرس كما يلي:

```
hsoubguide=# SELECT pg_size_pretty(pg_relation_size('users_pkey'));
```

pg_size_pretty
16 kB

8. 5. قياس حجم الجدول مع الفهارس

تُخزّن الفهارس بمعزل عن الجداول، لكن يمكنك معرفة الحجم الكلي للجدول بالإضافة للفهارس بالتعليمة التالية:

```
hsoubguide=# SELECT pg_size_pretty(pg_total_relation_size('users'));
```

pg_size_pretty
32 kB

ماذا تعني هذه الأرقام؟

عندما تُخبرنا Postgres بأن حجم الجدول 32KB فإن هذا الرقم ليس هو بالفعل حجم البيانات المخزنة فيه، ولكنه الحجم الذي يحجزه الجدول في الذاكرة.

ولفهم ذلك، سننشئ جدولًا بسيطًا وسنراقب تغير حجمه مع زيادة البيانات فيه أو نقصها:

```
hsoubguide=# CREATE TABLE size_calc(msg VARCHAR(255));

CREATE TABLE
```

حجم الجدول الآن في الذاكرة، هو 0 بايت:

```
hsoubguide=# SELECT pg_size_pretty(pg_total_relation_size('size_
calc'));
```

pg_size_pretty
0 bytes

سنعمل على إضافة نص مكون من 64 حرفًا:

```
hsoubguide=# INSERT INTO size_calc(msg) VALUES ('0123456789abcdefghij
klmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ..');

INSERT 0 1
```

وسنجد أن حجم الجدول قفز إلى 8kB فورًا:

```
hsoubguide=# SELECT pg_size_pretty(pg_total_relation_size('size_
calc'));
```

pg_size_pretty
8192 bytes

ولكن ماذا لو حذفنا محتويات الجدول؟

```
hsoubguide=# SELECT * FROM size_calc;
```

msg
0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ..

```
hsoubguide=# DELETE FROM size_calc;

DELETE 1

hsoubguide=# SELECT * FROM size_calc;

 msg
-----
```

رأينا أن الجدول فارغٌ الآن من المحتوى، ولكن حجمه في الذاكرة لا زال 8kB كما نرى في الاستعلام التالي:

```
hsoubguide=# SELECT pg_size_pretty(pg_total_relation_size('size_
calc'));
```

pg_size_pretty
8192 bytes

إلا أنه يمكننا استخدام المكنسة الكهربائية لقواعد البيانات، أقصد التعليمة VACUUM FULL، التي تنظّف الذاكرة من الملفات الفارغة، أو تلك التي تمددت في وقت ما، ثم لم يعد هناك حاجة إلى حجمها الكبير، كما في المثال التالي:

```
hsoubguide=# VACUUM FULL;

VACUUM

hsoubguide=# SELECT pg_size_pretty(pg_total_relation_size('size_
calc'));
```

pg_size_pretty
0 bytes

8.6. خلاصة الفصل

لا شك أن هذا الفصل قد ساعدك في التعرف إلى كيفية تتبع استخدام الذاكرة في قاعدة بياناتك، وقد أصبح بإمكانك معرفة الحجم الذي تستهلكه الفهارس، والحجم الذي تستهلكه الجداول عمومًا، مما سيساعدك لاحقًا في إدارة ذاكرة التخزين لقاعدة البيانات ككل.

9. إدارة الأداء وذاكرة التخزين المؤقتة

سنتعرف في هذا الفصل إلى طريقة تتبع أداء الاستعلامات في Postgres، وذلك لمعرفة الزمن المتوقع والحقيقي للاستعلام، كما يذكر كيفية عمل قيود على استخدام الفهارس، ثم سنتطرق لفهم ذاكرة التخزين المؤقتة Cache في Postgres.

9. 1. خطة التنفيذ (Execution plan)

لدى Postgres قدرة كبيرة على إظهار كيفية تنفيذ الاستعلامات خلف الكواليس، وهذا ما يُسمى بخطة التنفيذ ونستخدم لإظهار ذلك التعليمة `explain`، وإن فهمك لهذه المعلومات يساعدك في تحسين قاعدة بياناتك باستخدام الفهارس لرفع الكفاءة.

سننشئ جدولًا صغيرًا لنقوم عليه ببعض التجارب:

```
hsoubguide=# CREATE TABLE test_explain(msg varchar(20));

CREATE TABLE
```

جميع الاستعلامات في Postgres يكون لها خطة تنفيذ عندما يتم تنفيذها، وهناك ثلاث أشكال لتنفيذ التعليمة explain كما يلي:

- الشكل العام :باستخدام EXPLAIN، يقوم بعرض توقّع لما سيحدث تقريبًا، دون أن يتم التنفيذ الفعلي للتعليمة:

```
hsoubguide=# EXPLAIN INSERT INTO test_explain(msg) VALUES('Hsoub');

QUERY PLAN
------------------------------------------------------------
 Insert on test_explain  (cost=0.00..0.01 rows=1 width=58)
   ->  Result  (cost=0.00..0.01 rows=1 width=58)


hsoubguide=# SELECT * FROM test_explain ;

 msg
-----
```

- الشكل التحليلي: باستخدام EXPLAIN ANALYZE، يقوم بتنفيذ الاستعلام ثم يعرض شرحًا لما حدث خلال التنفيذ:

```
hsoubguide=# EXPLAIN ANALYZE INSERT INTO test_explain(msg)
VALUES('Hsoub');

QUERY PLAN
------------------------------------------------------------------------
 Insert on test_explain  (cost=0.00..0.01 rows=1 width=58) (actual
time=0.898..0.898 rows=0 loops=1)
   ->  Result  (cost=0.00..0.01 rows=1 width=58) (actual
time=0.003..0.004 rows=1 loops=1)
 Planning Time: 0.067 ms
 Execution Time: 0.952 ms
hsoubguide=# SELECT * FROM test_explain ;
```

msg
Hsoub

- الشكل المستفيض أو التفصيلي (verbose)، يزيد عن الشكل التحليلي بالقليل من المعلومات، وهنا يمكن استخدام `EXPLAIN VERBOSE` للشرح دون التنفيذ:

```
hsoubguide=# EXPLAIN VERBOSE INSERT INTO test_explain(msg)
VALUES('Hsoub2');

QUERY PLAN
------------------------------------------------------------------
 Insert on public.test_explain  (cost=0.00..0.01 rows=1 width=58)
   ->  Result  (cost=0.00..0.01 rows=1 width=58)
         Output: 'Hsoub2'::character varying(20)

hsoubguide=# SELECT * FROM test_explain ;
```

msg
Hsoub

أو استخدام `EXPLAIN ANALYZE VERBOSE` للشرح المستفيض مع التنفيذ:

```
hsoubguide=# EXPLAIN ANALYZE VERBOSE INSERT INTO test_explain(msg)
VALUES('Hsoub2');

QUERY PLAN
------------------------------------------------------------------------
 Insert on public.test_explain  (cost=0.00..0.01 rows=1 width=58)
(actual time=0.044..0.045 rows=0 loops=1)
   ->  Result  (cost=0.00..0.01 rows=1 width=58) (actual
time=0.003..0.004 rows=1 loops=1)
         Output: 'Hsoub2'::character varying(20)
 Planning Time: 0.074 ms
 Execution Time: 0.086 ms

hsoubguide=# SELECT * FROM test_explain ;
```

msg
Hsoub
Hsoub2

غالبًا يتم استخدام التعليمة explain مع عبارات SELECT، إلا أنه يمكن استخدامها أيضًا مع التعليمات:

- INSERT
- UPDATE
- DELETE
- EXECUTE
- DECLARE

9. 1. 1. استخدام التعليمة Explain لشرح الاستعلامات

نستعلم في المثال التالي عن الأسماء الأخيرة للموظفين ذوي الرواتب التي تبدأ من 50000 فما فوق، كما يلي:

```
hsoubguide=# SELECT last_name FROM employees WHERE salary >= 50000;
```

last_name
Adams
Smith

يمكننا تفحّص كيفية قيام Postgres بتنفيذ الاستعلام السابق كما يلي:

```
hsoubguide=# EXPLAIN SELECT last_name FROM employees WHERE salary >=
50000;

QUERY PLAN
-------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..1.06 rows=2 width=128)
   Filter: (salary >= 50000)
```

كما يمكن فهم أداء تنفيذ الاستعلام الحقيقي عن طريق ANALYZE كما يلي:

```
hsoubguide=# EXPLAIN ANALYZE SELECT last_name FROM employees WHERE
salary >= 50000;
```

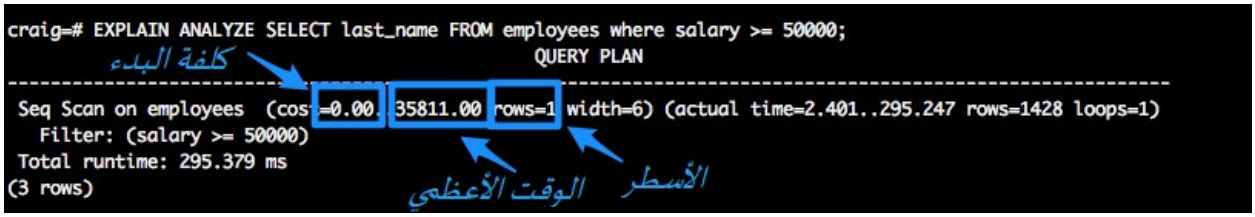
```
QUERY PLAN
------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..1.06 rows=2 width=128) (actual
time=0.032..0.036 rows=2 loops=1)
   Filter: (salary >= 50000)
   Rows Removed by Filter: 3
 Planning Time: 0.142 ms
 Execution Time: 0.084 ms
```

9. 1. 2. فهم خطط التنفيذ

إن نفّذنا التعليمة `EXPLAIN ANALYZE` على جدول مشابه للجدول السابق ولكن مع احتوائه على مليوني سطر، يمكن أن نحصل على المخرجات التالية:

```
craig=# EXPLAIN ANALYZE SELECT last_name FROM employees where salary >= 50000;
                                                  QUERY PLAN
-------------------------------------------------------------------------------------------------------------
 Seq Scan on employees  (cost=0.00..35811.00 rows=1 width=6) (actual time=2.401..295.247 rows=1428 loops=1)
   Filter: (salary >= 50000)
 Total runtime: 295.379 ms
(3 rows)
```

تَوضّح الصورة التالية معاني هذه الأرقام المكتوبة كما يلي:



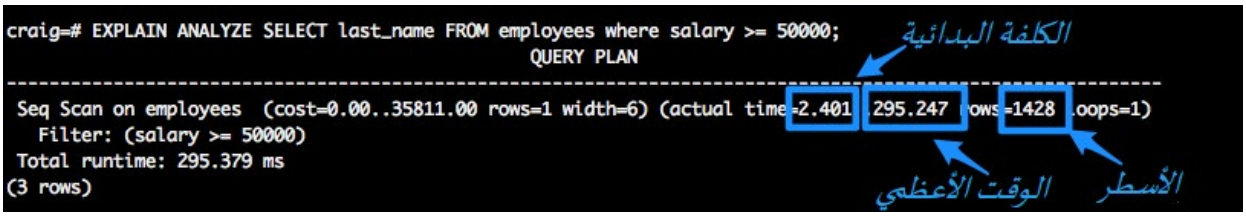
تشير الكلمة `Seq Scan` إلى أن عملية البحث التي تجري هي البحث التسلسلي.

أما العبارة التالية:

```
(cost=0.00..35811.00 rows=1 width=6)
```

فهي التقدير التقريبي (وليس الحقيقي) لما يُتوقّع أن يستغرقه تنفيذ الاستعلام، حيث يعبر الزمن 0.00 عن الزمن اللازم لبدء الاستعلام، والزمن 35811.00 هو الزمن المتوقع لإنهاء الاستعلام، أما القيمة rows=1 هي عدد الأسطر التي يُتوقّع أن تُعطى في المخرجات، والقيمة width=6 هي الحجم التقريبي لمحتوى الأسطر التي يُتوقع أن تُعطى في المخرجات.

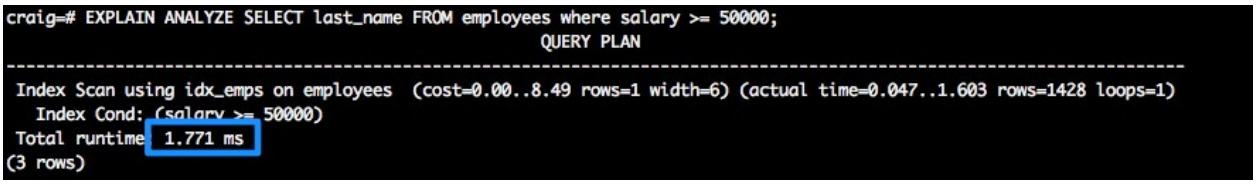
ولأننا قُمنا بتنفيذ التعليمة EXPLAIN ANALYZE فإننا لم نحصل فقط على التقدير المتوقع للتنفيذ، بل على الوقت الحقيقي المستغرق كذلك كما تبيّن الصورة التالية:



يُمكننا بذلك رؤية الوقت الكبير المستهلك في المسح المتتالي، وسنقارنه مع الوقت المستغرق عند إضافة فهرس واختبار النتائج:

```
CREATE INDEX idx_emps on employees (salary);
```

وبذلك خفضنا زمن الاستعلام من 295 ميللي ثانية إلى 1.7 ميللي ثانية فقط كما يوضح الشكل التالي:



كانت هذه مقدمة في استخدام التعليمة EXPLAIN ولا بد أن تجربها على استعلامات أكبر حجمًا لاستكشاف ما يمكنها إظهاره.

9. 2. قيود شرطية على إنشاء الفهارس

قد نحتاج أحيانًا إلى وضع بعض القيود على البيانات التي نرغب بالفعل في فهرستها، فمثلًا قد لا نرغب بحذف مستخدم ما من قاعدة البيانات ولكن نريد أن نظهره على أنه محذوف بحيث يمكن إرجاع بياناته لو أراد العودة للموقع بعد شهر مثلًا، ولكن في الوقت نفسه لا نريد أن يزداد حجم البيانات التي نقوم بفهرستها، ولذلك نستخدم الفهارس الجزئية.

سنستخدم الفهارس الجزئية في المثال التالي لوضع فهرس فريد فقط للمستخدمين غير المحذوفين:

```
CREATE UNIQUE INDEX user_email ON users (email) WHERE deleted_at IS
NULL;
```

9. 3. ذاكرة التخزين المؤقتة (Cache)

إن معظم التطبيقات تتعامل مع جزء صغير من البيانات بشكل متكرر، ومثل العديد من الأمور الأخرى فإن البيانات يمكن أن تتبع قاعدة 80/20 حيث أن 20% من البيانات هي التي يتم قراءتها في 80% من الحالات، وأحيانا يكون هذا الرقم أكبر من ذلك.

تعمل Postgres على تتبع أنماط استخدامك للبيانات وتحاول الإبقاء على البيانات الأكثر وصولًا في ذاكرة التخزين المؤقتة. عمومًا، سترغب في أن يكون نسبة نجاح الوصول إلى ذاكرة التخزين المؤقتة هي 99%، ويمكنك معرفة هذه النسبة باستخدام التعليمات التالية:

```
SELECT
       sum(heap_blks_read) as heap_read,
       sum(heap_blks_hit) as heap_hit,
       (sum(heap_blks_hit) - sum(heap_blks_read)) / sum(heap_blks_
hit) as ratio

FROM
     pg_statio_user_tables;
```

إن وجدت أن نسبة نجاح الوصول إلى التخزين المؤقتة عندك أخفض من 99% بشكل كبير، فربما يتوجب عليك زيادة حجمها المخصص لقاعدة البيانات.

9. 3. 1. فهم استخدام الفهارس

تُعد الفهارس الطريقة الأساسية الأخرى لزيادة كفاءة قاعدة البيانات، حيث تضيف العديد من بيئات العمل الفهارس إلى المفاتيح الرئيسية في الجداول، ولكن إن كنت تُجري عمليات البحث على حقول أخرى أو تقوم بالربط بين الجداول فربما عليك إضافة الفهارس يدويًا إلى هذه الأعمدة.

الفهارس هي الأهم في الجداول الكبيرة، فرغم أن الوصول إلى الجداول في ذاكرة التخزين المؤقتة أسرع من الوصول إليها على القرص الصلب، إلا أنه حتى البيانات فيها يمكن أن تكون بطيئة إن كان يتوجب على Postgres أن تحلل مئات آلاف الأسطر لمعرفة إن كانت تحقق شرطًا ما.

يمكنك استخدام التعليمة التالية لتوليد قائمة بالجداول مع النسبة المئوية لاستخدام الفهارس فيها:

```
SELECT
      relname, 100 * idx_scan / (seq_scan + idx_scan) percent_of_
times_index_used,
      n_live_tup rows_in_table
FROM
     pg_stat_user_tables
WHERE
     seq_scan + idx_scan \> 0
ORDER BY
     n_live_tup DESC;
```

إن لم تكن قريبًا من النسبة 99% في الجداول التي تحوي 10 آلاف سطرًا فأكثر، فربما يجب عليك إضافة المزيد من الفهارس، ولكن عليك معرفة العمود المناسب الذي يجب اعتباره فهرسًا، وذلك عن طريق معرفة نوع الاستعلامات التي تتم على الجداول.

عمومًا، قد يكون من المناسب وضع الأعمدة التي تحوي المعرّفات `ID` أو على الأعمدة تقوم بالترشيح على أساسها غالبًا مثل `created_at`.

> نصيحة احترافية: إن كنت تضيف فهرسًا في قاعدة بيانات قيد العمل استخدم CREATE INDEX CONCURRENTLY لتقوم ببناء الفهرس في الخلفية دون التسبب في قفل الجدول ومنع تنفيذ الاستعلامات عليه.

يمكن أن يستغرق الإنشاء الآني للفهارس 3-2 أضعاف الوقت المستغرق في العادة، ولا يمكن تنفيذها على دفعات، ولكن هذه المقايضة بين الوقت وتجربة المستخدم تستحق ذلك، فلا شك أنك لا تريد توقّف الموقع الخاص بك كلما قمت بإنشاء فهرس جديد في جدول كبير الحجم.

9. 3. 2. مثال باستخدام بيانات حقيقية

عند النظر إلى بيانات حقيقية من واجهة Heroku التي تم إطلاقها مؤخرًا، يمكن تنفيذ الاستعلام التالي ورؤية النتائج كما يلي:

```
SELECT relname,
       100 * idx_scan / (seq_scan + idx_scan) percent_of_times_index_
used,
       n_live_tup rows_in_table
FROM pg_stat_user_tables
ORDER BY n_live_tup DESC;
```

relname	percent_of_times_index_used	rows_in_table		
events	0	669917		
app_infos_user_info	0	198218		
app_infos	50	175640		
user_info	3	46718		
rollouts	0	34078 favorites	0	3059
schema_migrations	0	2		
authorizations	0	0		
delayed_jobs	23	0		

يمكننا أن نرى أن جدول events فيه 700 ألف سطرًا تقريبًا وليس فيه فهارس تم استخدامها، ومن هنا يمكننا التحقق ضمن تطبيقنا لنرى بعض الاستعلامات الشائعة التي تُستخدم، وأحد هذه الأمثلة هو جلب الأحداث لمدوّنة ما.

يمكنك أن ترى خطة التنفيذ باستخدام التعليمة EXPLAIN ANALYZE التي تعطيك فكرة أفضل عن الاستعلام:

```
EXPLAIN ANALYZE SELECT * FROM events WHERE app_info_id = 7559;

QUERY PLAN
 ------------------------------------------------------------------------
 Seq Scan on events (cost=0.00..63749.03 rows=38 width=688) (actual
 time=2.538..660.785 rows=89 loops=1) Filter: (app_info_id = 7559)
 Total runtime: 660.885 ms
```

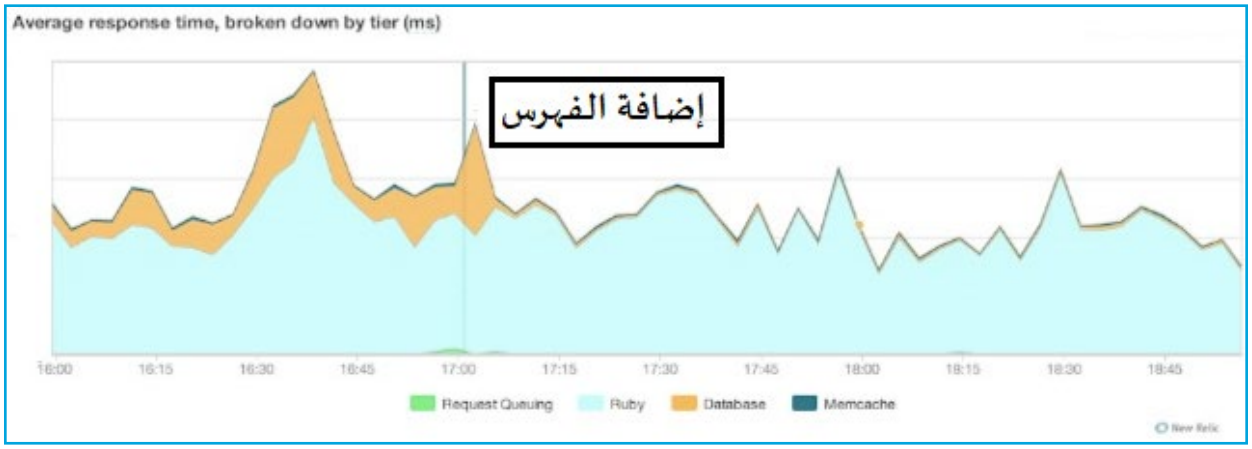
وبما أن طريقة البحث ضمن هذا الجدول هي المسح التسلسلي، فيمكننا تحسين ذلك باستخدام فهرس، وسنضيف الفهرس الخاص بنا آنيًّا لمنع قفل الجدول، ثم سنرى كيف تتغير كفاءة الجدول:

```
CREATE INDEX CONCURRENTLY idx_events_app_info_id ON
events(app_info_id);
EXPLAIN ANALYZE SELECT * FROM events WHERE app_info_id = 7559;

------------------------------------------------------------------------
Index Scan using idx_events_app_info_id on events (cost=0.00..23.40
rows=38 width=688) (actual time=0.021..0.115 rows=89 loops=1)
 :   Index Cond: (app_info_id = 7559)

 Total runtime: 0.200 ms
```

يتضح من التعليمة السابقة التحسن الذي أدى إليه استخدام الفهرس، ولكن يمكننا أيضًا أن نحلل النتيجة باستخدام الإضافة New Relic لنرى أننا خفّضنا بشكل كبير من الوقت المستغرق في المعالجة لقاعدة البيانات بإضافة هذا الفهرس وفهارس أخرى.



9. 3. 3. معدل نجاح الوصول إلى الفهارس في ذاكرة التخزين المؤقت

أخيرًا لدمج الأمرين معًا، فيمكنك استخدام التعليمة التالية في حال كنت ترغب بمعرفة معدل نجاح الوصول إلى الفهارس المخزنة ضمن ذاكرة التخزين المؤقت:

```
SELECT
   sum(idx_blks_read) as idx_read,
   sum(idx_blks_hit) as idx_hit,
   (sum(idx_blks_hit) - sum(idx_blks_read)) / sum(idx_blks_hit) as
ratio
FROM
     pg_statio_user_indexes;
```

عمومًا يمكنك أن تتوقع أن يكون هذا المعدل بقيمة 99% بشكل مشابه لمعدل نجاح الوصول إلى المخزن المؤقت المعتاد.

9. 4. خلاصة الفصل

تعرفنا في هذا الفصل إلى طريقة تتبع أداء الاستعلامات في Postgres عن طريق التعليمة EXPLAIN، كما عرفنا أهمية ذاكرة التخزين المؤقتة Cache في Postgres، ويُعد هذا الفصل خطوتك الأولى في إدارة الأداء عمومًا أثناء عملك مع قواعد بيانات Postgres.

10. أوامر متقدمة في صَدَفة psql

تعرفنا في بداية هذا الكتاب إلى بعض أوامر صدفة `psql` الأساسية، ولكننا يجب أن نتعرف إلى بقية هذه الأوامر، والتي يوجد العديد منها لمساعدتنا في أمور قد تبدو صعبة التنفيذ بطرق أخرى.

نذكر في هذا الفصل أوامر للاتصال بقاعدة بيانات، استعراض الفهارس، تنسيق المخرجات، كيفية استخدام الأوامر الشرطية وكيفية تنفيذ أوامر نظام التشغيل من داخل صدفة `psql` وغيرها من الأوامر التي تُعتبر الأكثر أهمية من بين الأوامر المتقدمة.

10. 1. أصناف أوامر psql

تنقسم أوامر `psql` إلى المجموعات التالية:

- أوامر مساعدة، لعرض معلومات مساعدة في كيفية استخدام أوامر أخرى.
- أوامر التحكم بمخزن الاستعلامات، هو ملف مؤقت يمكن كتابة الاستعلامات فيه عن طريق محرر النصوص، لتسهيل كتابة الاستعلامات الطويلة قبل تنفيذها، أو كتابة عدة أوامر واستعلامات وتحرير كل ذلك قبل التنفيذ.
- أوامر الإدخال والإخراج، تسمح بتنفيذ تعليمات مخزنة في ملفات خارجية، أو كتابة مخرجات التعليمات والاستعلامات إلى ملفات خارجية.
- الأوامر الشرطية، تسمح باستخدام `if` و `else` للتحكم بسير سلسلة من التعليمات.

- أوامر استعراض المعلومات، لعرض معلومات عن قاعدة البيانات، الجداول، المخططات، وغيرها.
- أوامر تنسيق البيانات، تسمح بتنسيق طريقة إظهار المخرجات.
- أوامر الاتصال، للاتصال بقاعدة بيانات أخرى.
- أوامر نظام التشغيل، تسمح بتنفيذ أوامر نظام التشغيل من داخل الصدفة.
- أوامر التحكم بالمتغيرات، لتحديد قيمة متغير أو إزالتها أو إدخالها من قبل المستخدم.
- أوامر الكائنات الكبيرة، وهي الكائنات التي لا تُخزَّن في قاعدة البيانات مباشرةً بل تُخزن في ملفات مستقلة، ويتم الإشارة إليها باستخدام `Object id`.

سنتعرف في هذا الفصل إلى الأوامر الأكثر أهمية فقط، إلا أن اطلاعك على باقي الأوامر لا شك سيتيح لك استخدامًا أكثر حرية واحترافية لصدفة `psql`، كما سيسمح لك في كتابة استعلامات معقدة حتى لو لم تكن محترفًا في SQL.

10. 2. أوامر الاتصال

تُعتبر أوامر الاتصال من أهم أوامر صدفة `psql`، فهي التي تسمح بالاتصال بقاعدة بيانات معينة، أو استعراض المعلومات الخاصة بها.

10. 2. 1. استعراض معلومات الاتصال

يمكن استخدام الأمر `\conninfo` لاستعراض المعلومات الرئيسية للاتصال الحالي (اسم قاعدة البيانات، اسم المستخدم، رقم المنفذ للعملية postgres):

```
hsoubguide=# \conninfo

You are connected to database "hsoubguide" as user "postgres" via
socket in "/var/run/postgresql" at port "5432".
```

10. 2. 2. الاتصال بقاعدة بيانات أخرى

إن لم تقم بالدخول إلى الصدفة محددًا اسم قاعدة البيانات التي ترغب بالاتصال بها، فيمكنك الاتصال من داخل الصدفة بالأمر `\connect`:

```
hsoubguide=# \connect postgres

You are now connected to database "postgres" as user "postgres".

postgres=# \connect hsoubguide

You are now connected to database "hsoubguide" as user "postgres".
```

10. 3. أوامر استعراض أخرى مهمة

سنذكر فيما يلي كيفية استعراض التوابع في صدفة `psql` وكيفية استعراض الفهارس المخزنة في قاعدة البيانات، وقد ذكرنا العديد من أوامر الاستعراض في الفصل الرابع من هذا الكتاب، استعرضنا فيها الجداول وقواعد البيانات، فيمكن الرجوع للفصل الرابع لأوامر الاستعراض الأساسية.

10. 3. 1. استعراض جميع التوابع المتاحة

يعرض الأمر `\df` قائمة بجميع التوابع التي يُمكن استخدامها في الصدفة، مع نوع القيمة المعادة من كل تابع، ونوع الوسطاء الممررة لكل منهم:

```
hsoubguide=# \df
```

Schema	Name	Result data type	Argument data types	Type
public	akeys	text[]	hstore	func
public	avals	text[]	hstore	func
public	defined	boolean	hstore, text	func
public	delete	hstore	hstore, hstore	func
public	delete	hstore	hstore, text	func
public	delete	hstore	hstore, text[]	func

Schema	Name	Result data type	Argument data types	Type
public	each	SETOF record	hs hstore, OUT key text, OUT value text	func
public	exist	boolean	hstore, text	func
public	exists_all	boolean	hstore, text[]	func
public	exists_any	boolean	hstore, text[]	func
public	fetchval	text	hstore, text	func
...				
...				

كما يمكن الحصول على المزيد من المعلومات التي تخص كل تابع بإضافة الرمز + في نهاية الأمر السابق.

10. 3. 2. استعراض الفهارس المخزنة في قاعدة البيانات

يعرض الأمر `\di` قائمة بأهم الفهارس (indexes) المخزنة في قاعدة البيانات، كما يمكن استخدام الرمز + لاستعراض معلومات إضافية، منها الحجم الذي تحجزه هذه الفهارس في الذاكرة.

```
hsoubguide=# \di+
```

Schema	Name	Type	Owner	Table	Size	Description
public	basket_a_pkey	index	postgres	basket_a	16 kB	
public	basket_b_pkey	index	postgres	basket_b	16 kB	
public	departments_department_key	index	postgres	departments	16 kB	
public	departments_pkey	index	postgres	departments	16 kB	
public	employees_pkey	index	postgres	employees	16 kB	
public	marks_pkey	index	postgres	marks	16 kB	
public	names_pkey	index	postgres	names	16 kB	
public	phones_pkey	index	postgres	phones	16 kB	

public	products_pkey	index	postgres	products	16 kB	
public	purchases_pkey	index	postgres	purchases	40 kB	
public	student_pkey	index	postgres	student	16 kB	
public	table1_pkey	index	postgres	table1	16 kB	
public	table2_pkey	index	postgres	table2	16 kB	
public	test_table_pkey	index	postgres	test_table	16 kB	
public	users2_pkey	index	postgres	users2	8192 bytes	
public	users_pkey	index	postgres	users	16 kB	

10.4. أوامر التنسيق

تسمح هذه الأوامر بتغيير التنسيق الافتراضي للجداول الناتجة من الاستعلامات، وتتيح ميزة كبيرة قد تختصر عليك الكثير من الوقت والجهد الذي يلزم لتحويل التنسيق الافتراضي يدويًا إلى شكل أكثر فائدة.

10.4.1. تنسيق المحاذاة

يمكن استخدام الأمر `\a` لتشغيل محاذاة محتوى العمود مع اسمه، أو لإيقاف ذلك.

```
hsoubguide=# \a

Output format is unaligned.

hsoubguide=# SELECT * FROM products LIMIT 4;

id|title|price|created_at|deleted_at|tags
1|Dictionary|9.99|2011-01-01 22:00:00+02||{Book}
2|Python Book|29.99|2011-01-01 22:00:00+02||{Book,Programming,Python}
3|Ruby Book|27.99|2011-01-01 22:00:00+02||{Book,Programming,Ruby}
4|Baby Book|7.99|2011-01-01 22:00:00+02||{Book,Children,Baby}

hsoubguide=# \a
```

```
Output format is aligned.
hsoubguide=# SELECT * FROM products LIMIT 4;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}
2	Python Book	29.99	2011-01-01 22:00:00+02		{Book,Programming,Python}
3	Ruby Book	27.99	2011-01-01 22:00:00+02		{Book,Programming,Ruby}
4	Baby Book	7.99	2011-01-01 22:00:00+02		{Book,Children,Baby}

كما يمكنك تغيير المحرف الفاصل بين الأعمدة باستخدام الأمر `\f` مع تمرير المحرف المُراد، ويمكنك كذلك منع طباعة أسماء الأعمدة والاكتفاء بالمحتويات باستخدام الأمر `\t`.

10. 4. 2. تنسيق HTML

يمكنك استخدام الأمر `\H` لتغيير حالة المخرجات من النمط المكتوب إلى نمط HTML وبالعكس:

```
hsoubguide=# SELECT * FROM products LIMIT 4;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}
2	Python Book	29.99	2011-01-01 22:00:00+02		{Book,Programming,Python}
3	Ruby Book	27.99	2011-01-01 22:00:00+02		{Book,Programming,Ruby}
4	Baby Book	7.99	2011-01-01 22:00:00+02		{Book,Children,Baby}

```
hsoubguide=# \H

Output format is html.

hsoubguide=# SELECT * FROM products LIMIT 4;
```

```
<table border="1">
  <tr>

    <th align="center">id</th>
    <th align="center">title</th>
    <th align="center">price</th>
    <th align="center">created_at</th>
    <th align="center">deleted_at</th>
    <th align="center">tags</th>
  </tr>
  <tr valign="top">
    <td align="right">1</td>
    <td align="left">Dictionary</td>
    <td align="right">9.99</td>
    <td align="left">2011-01-01 22:00:00+02</td>
    <td align="left">  </td>
    <td align="left">{Book}</td>
  </tr>
  <tr valign="top">
    <td align="right">2</td>
    <td align="left">Python Book</td>
    <td align="right">29.99</td>
    <td align="left">2011-01-01 22:00:00+02</td>
    <td align="left">  </td>
    <td align="left">{Book,Programming,Python}</td>
  </tr>
  <tr valign="top">
    <td align="right">3</td>
    <td align="left">Ruby Book</td>
    <td align="right">27.99</td>
    <td align="left">2011-01-01 22:00:00+02</td>
    <td align="left">  </td>
    <td align="left">{Book,Programming,Ruby}</td>
  </tr>
  <tr valign="top">
    <td align="right">4</td>
    <td align="left">Baby Book</td>
    <td align="right">7.99</td>
    <td align="left">2011-01-01 22:00:00+02</td>
    <td align="left">  </td>
```

```
    <td align="left">{Book,Children,Baby}</td>
  </tr>
</table>
<p><br />
</p>

hsoubguide=# \H

Output format is aligned.
```

10. 5. استعراض تاريخ الاستعلامات وحفظه

يمكن استعراض تاريخ الاستعلامات وأوامر الصدفة التي نُفِّذت من قبل باستخدام الأمر `\s`، كما يمكنك حفظ هذا السجل بإضافة اسم الملف الذي ترغب بحفظ السجل فيه.

```
hsoubguide=# \s /tmp/psql_history.txt

Wrote history to file "/tmp/psql_history.txt".
```

10. 6. أوامر التعامل مع المتغيرات

تسمح `psql` بتعريف متغيرات داخلها، واستخدامها لاحقًا ضمن التعليمات أو الاستعلامات اللاحقة.

يمكن تعريف متغير باستخدام الأمر `\set` كما يمكن حذفه باستخدام `\unset`، وفي حال أردنا استخدام قيمة هذا المتغير، علينا أن نضع قبل اسمه الرمز :.

```
hsoubguide=# \set NAME mostafa
hsoubguide=# \set rate 5.9

hsoubguide=# \echo :rate

5.9

hsoubguide=# \echo :NAME

mostafa
hsoubguide=# SELECT * FROM test_table WHERE number > :rate;
```

id	number	name
4	22	hel..
1	10	10
2	13	13

كما يمكن استخدام الأمر `\prompt` لطلب إدخال قيمة من المستخدم، وذلك بعد طباعة عبارة الطلب:

```
hsoubguide=# \prompt 'Please enter your name: ' NAME
Please enter your name: Mostafa

hsoubguide=# \echo :NAME

Mostafa
```

يمكنك استخدام قيم هذه المتغيرات ضمن الاستعلامات أو ضمن أوامر `psql` أخرى.

10. 7. الأوامر الشرطية

بعد أن تعرفنا على الأوامر الخاصة بالمتغيرات، يجدر بنا أن نعرف أن `psql` تتيح استخدام التعليمات الشرطية `\if` و `\elif` و `\else` للتحكم بكيفية سير التنفيذ، ويوضح المثال التالي المأخوذ من التوثيق كيفية استخدامها:

```
SELECT
    EXISTS(SELECT 1 FROM customer WHERE customer_id = 123) as is_
customer,
    EXISTS(SELECT 1 FROM employee WHERE employee_id = 456) as is_
employee
\gset
\if :is_customer
    SELECT * FROM customer WHERE customer_id = 123;
\elif :is_employee
    \echo 'is not a customer but is an employee'
    SELECT * FROM employee WHERE employee_id = 456;
\else
    \if yes
        \echo 'not a customer or employee'
```

```
    \else
        \echo 'this will never print'
    \endif
\endif
```

10.8. أوامر نظام التشغيل

تسمح لنا صدفة psql بتنفيذ أوامر من خارج إمكانيات الصدفة، وهذا يسمح لنا بالبقاء ضمنها، والتعامل مع نظام التشغيل من داخلها مباشرةً.

10.8.1. تنفيذ أوامر صدفة bash ضمن psql

تسمح psql بتنفيذ أوامر صدفة bash بكتابتها بعد الأمر \! كما يلي:

```
postgres=# \! echo Hello

Hello

postgres=# \! pwd

/usr/pgsql-12/bin
```

10.8.2. تشغيل توقيت الاستعلام

في الحالة الافتراضية لا يكون توقيت تنفيذ الاستعلام مُتاحًا للعرض، ولكن يمكننا تفعيله من خلال الأمر التالي:

```
hsoubguide=# SELECT * FROM products LIMIT 1;
```

id	title	price	created_at	deleted_at	tags
1	Dictionary	9.99	2011-01-01 22:00:00+02		{Book}

حيث سيتيح ذلك الأمر إظهار توقيت الاستعلام بالميلي ثانية.

```
Time: 0.723 ms
```

10. 9. الخروج من صَدَفة postgres

قد تقضي وقتًا طويلًا داخل صدفة psql، ليس حبًا بها، ولكن لعدم معرفة كيفية الخروج منها، لذلك لا تنسَ أن الأمر \q هو الذي يُخرجك من صدفة psql.

```
hsoubguide=# \q

bash-4.2$
```

10. 10. خلاصة الفصل

تعرفنا في هذا الفصل إلى جميع أنواع أوامر صدفة psql، وذكرنا أكثرها أهمية، بما يسمح لك بزيادة كفاءة تعاملك مع قواعد بيانات Postgres، وتذكر دومًا أنه يمكنك استعراض العديد من الأوامر الأخرى من خلال الأمر \?.

10. 11. خاتمة الكتاب

وصلنا إلى نهاية هذا الكتاب الشامل حول Postgres، والذي سعينا من خلاله إلى الحديث عن أهم العمليات والخصائص التي تحتويها قواعد بيانات Postgres. والآن يمكنك الانتقال إلى مرحلة أعلى لتطوير قدراتك في إدارة قواعد بيانات Postgres أو لتعميق خبراتك في استعلامات SQL، أو للتعرف إلى مزايا متقدمة في Postgres وربما لاحقًا المساهمة في تطوير جزء من الشيفرة البرمجية الخاصة بها.

نرجو لك التوفيق في هذه الرحلة!