Warum PostgreSQL momentan so erfolgreich ist

Daniel Westermann, dbi services sa

Obwohl PostgreSQL bereits seit mehr als zwanzig Jahren auf dem Markt ist, und zwar durchaus erfolgreich, hat es in den vergangenen Jahren, und speziell im Jahr 2018, enorm an Bedeutung gewonnen. Wird sich dieser Trend die kommenden Jahre fortsetzen oder werden sich die bisherigen Platzhirsche ihre Marktanteile wieder zurückholen? Um diese Frage beantworten zu können, muss man einerseits die Community hinter PostgreSQL verstehen, andererseits aber auch ergründen, warum denn gerade jetzt die Zeit für PostgreSQL gekommen scheint und wie das alles mit anderen Entwicklungen zusammenhängt.

PostgreSQL ist durch und durch Community-getrieben, doch was soll das heißen? Schaut man sich andere Open-Source-Datenbanken wie MariaDB und MySQL an, dann gibt es im Vergleich zu PostgreSQL einen riesigen Unterschied: Bei PostgreSQL gibt es keine Firma, die hinter dem Projekt steht. PostgreSQL kann man also nicht kaufen, weil es keine Firma „PostgreSQL" gibt. Es gibt allerdings etliche Firmen, die Mitarbeiter einstellen, um nur an PostgreSQL zu arbeiten, etwa EnterpriseDB oder Fujitsu. Ganz klar versuchen diese Firmen auch ihre eigenen Wünsche in PostgreSQL einzubringen, sie können das Projekt jedoch nicht kontrollieren.

Egal was in PostgreSQL Einzug halten soll, es ist immer die Community, die entscheidet, ob es schlussendlich integriert wird oder nicht. Man kann natürlich anmerken, dass eine Firma so viele Entwickler anstellen könnte, dass sie die Mehrheit der Community bilden und somit Einfluss nehmen können. In der realen Welt ist das allerdings schon dadurch ausgeschlossen, dass sich so viele Firmen auf dem ganzen Globus an PostgreSQL beteiligen, dass das sehr wahrscheinlich niemals passieren kann. Das hört sich erst einmal alles sehr theoretisch an, wird aber schnell klar, wenn man sich ansieht, welchen Weg ein Patch oder ein neues Feature in PostgreSQL zurücklegt, bis es dann wirklich in PostgreSQL landet.

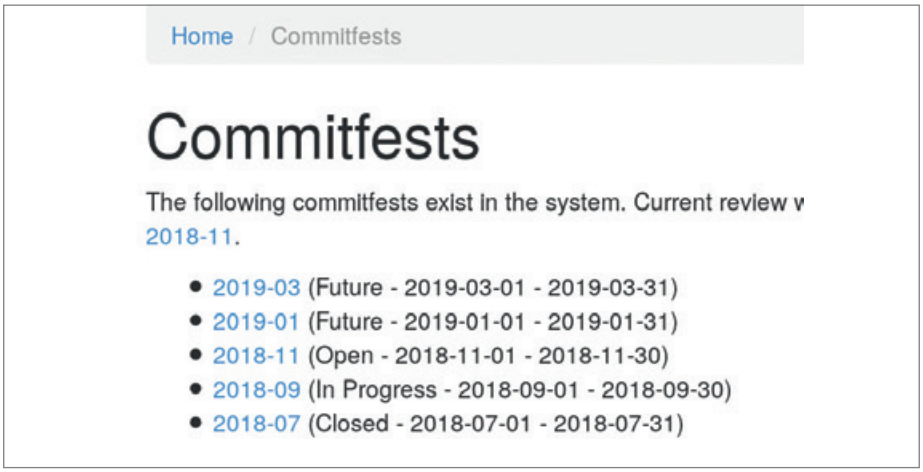

Abbildung 1: Daran arbeitet die PostgreSQL-Community aktuell

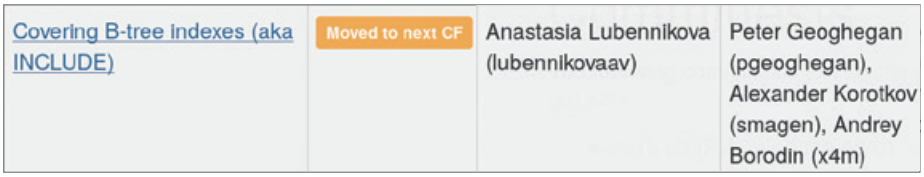

Abbildung 2: So arbeitet die PostgreSQL-Community

Patches/neue Features in PostgreSQL

Um sich zu informieren, woran die PostgreSQL-Community momentan arbeitet, gibt es mehrere Wege. Für Außenstehende führt der einfachste Weg über die sogenannten „Commitfests". Diese sind für alle unter *„https://commitfest.postgresql.org"* einsehbar *(siehe Abbildung 1)*.

Commitfests haben entweder den Status „Closed", „In Progress", „Open" oder „Future". Diejenigen, die mit „Future" gekennzeichnet sind, kann man getrost ignorieren, da man dort nichts finden wird. Commitfests im Status „Open" oder „In Progress" beinhalten das, was momentan entwickelt wird, und unter Status „Closed" befindet sich alles, was in der Vergangenheit war.

Möchte man sich also darüber informieren, was momentan geschieht, sind „2018-09" und „2018-11" von Interesse. Für das Beispiel ist allerdings „2018-01" das Commitfest der Wahl, da es einen Patch enthält, an dem man sehr schön sehen kann, wie die PostgreSQL-Community arbeitet *(siehe Abbildung 2)*.

Jeder Patch oder jedes neue Feature braucht einen Titel (Spalte 1). Allerdings gibt es immer auch einen Status (Spalte 2), eine oder mehrere Personen, die den Patch oder das Feature entwickeln (Spalte 3), und es braucht immer sogenannte „Reviewer", die sich ansehen, was gemacht wurde und ob es den PostgreSQL-Standards entspricht. Reviewer testen, geben Feedback, machen Verbesserungsvorschläge und können durchaus am Ende zu der Entscheidung kommen, dass ein Patch oder Feature nicht weiter verfolgt wird. Um nun die komplette Entwicklung eines Patches oder neuen Features zu verstehen, kann man sich zu diesem Details anzeigen lassen, indem man den Titel auswählt *(siehe Abbildung 3)*.

Anhand des „Status" kann man sehr schnell sehen, in welchem Zustand der Patch momentan ist. In diesem Fall wurde zweimal in ein folgendes Commitfest übertragen, um dann schließlich in „2018-03" in PostgreSQL zu landen (Status „Committed").

Jeder kann sich als „Reviewer" für einen Patch oder ein neues Feature registrieren. Alles was es dafür braucht, ist ein Community Account, den man sich unter *„https://www.postgresql.org/account"* erstellen kann. Schon das Testen, ob ein Patch sauber kompiliert, ist ein Beitrag zum Review-Prozess und kann ein erster Schritt sein, sich an PostgreSQL zu beteiligen.

Das Allerwichtigste an dieser Detail-Seite ist allerdings der Link zur ersten E-Mail an die Mailing-Liste, die zu diesem Patch oder neuem Feature geführt hat. In der PostgreSQL-Welt ist mehr oder weniger alles E-Mail-basiert und es gibt diverse Mailing-Listen, bei denen man sich registrieren kann, um an den Diskussionen teilzuhaben *(siehe „https://www.postgresql.org/list")*. Im Falle eines neuen Features oder eines Patches geht es um die „pgsql-hackers"-Mailing-Liste, und genau diese wird hier referenziert. Folgt man diesem Link, kommt man zum Start der Diskussion zu diesem neuen Feature *(siehe Abbildung 4)*.

Das mag nun nicht weiter bedeutend erscheinen, allerdings hat das einen riesigen Vorteil gegenüber Herstellern von proprietärer Software: Der komplette Entwicklungsprozess ist für alle jederzeit einsehbar und vor allem auch nachvollziehbar. Schaut man sich die Archive der Mailing-Listen genauer an (in diesem Fall die „pgsql-hackers"-List unter *„https://www.postgresql.org/list/pgsql-hackers"*),

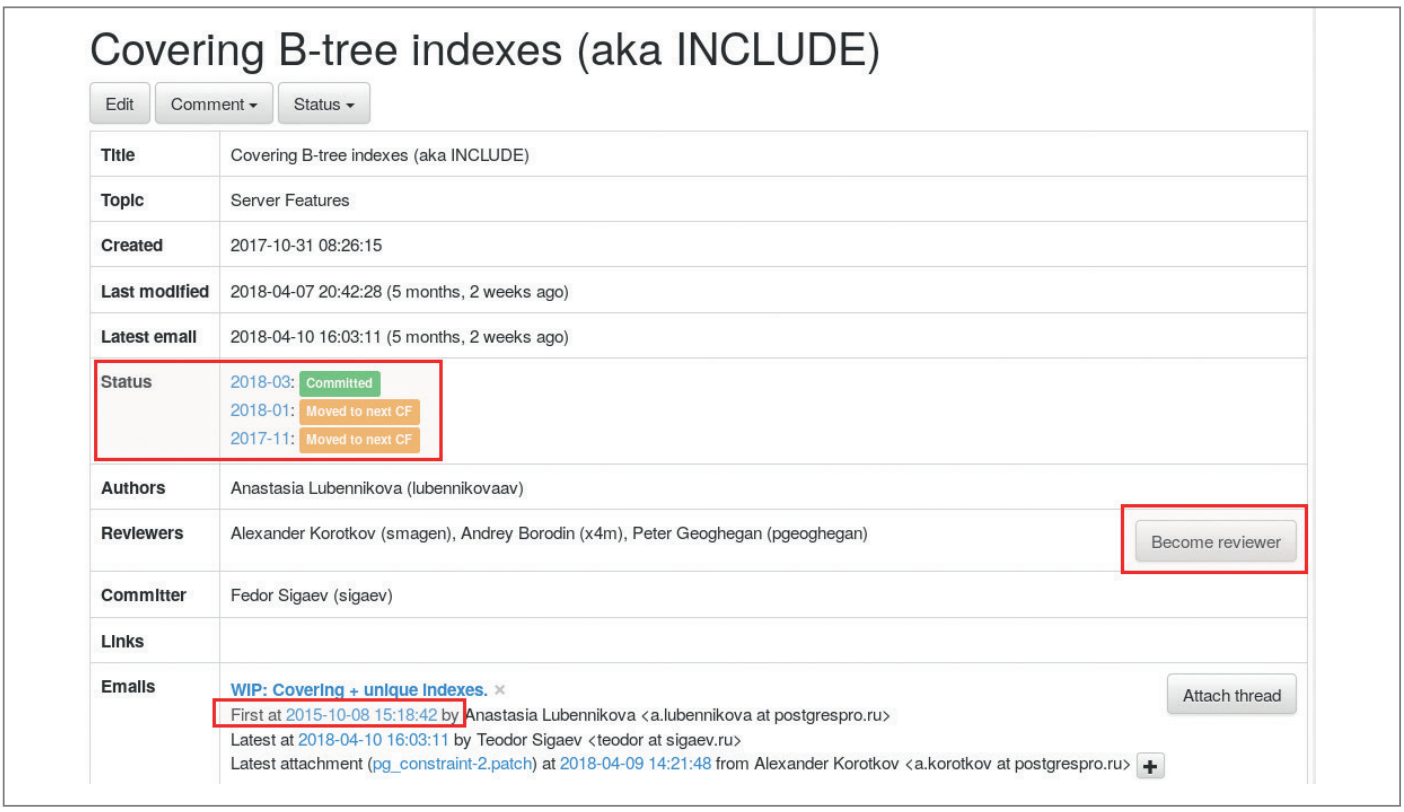

Abbildung 3: Der Verlauf eines Patches

wird man feststellen, dass alle Diskussionen bis zurück ins Jahr 1997 nachgelesen werden können – alle und öffentlich im Internet.

Die volle Transparenz trägt auch zum momentanen Erfolg von PostgreSQL bei, und sei es nur deswegen, weil es ein gutes Gefühl gibt, wenn man dabei sein oder zumindest jederzeit nachvollziehen kann, warum es zu gewissen Entscheidungen kam oder eben nicht. Zudem kann man auch immer abschätzen, was in der nächsten Version von PostgreSQL vorhanden sein wird und was nicht. Zu guter Letzt kann das auch jederzeit von jedem selbst getestet werden, denn auch die Patches an sich sind öffentlich.

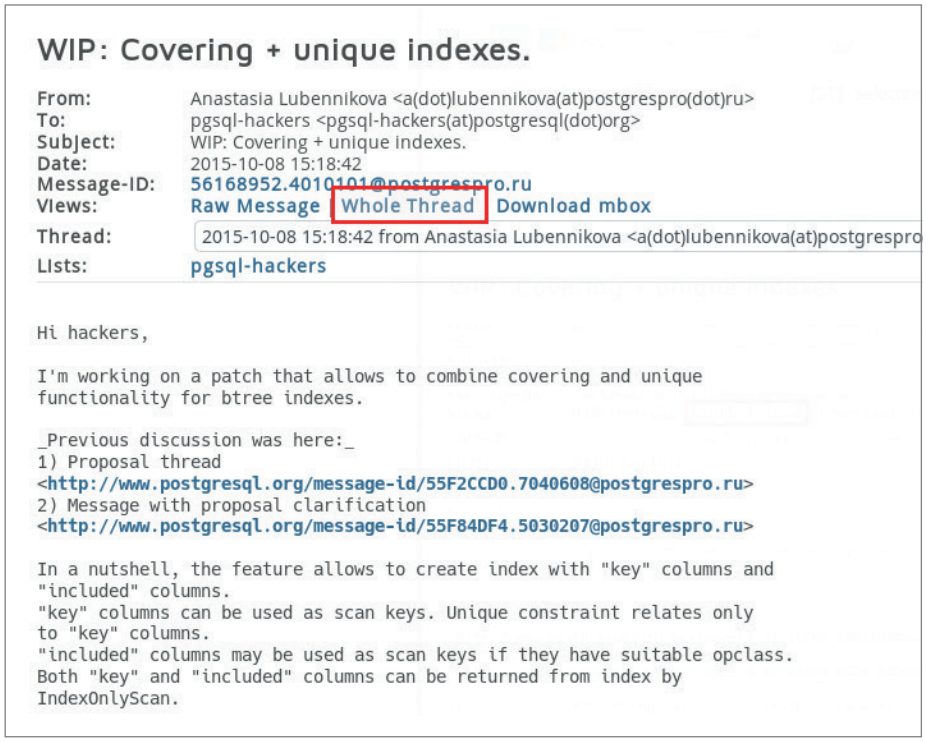

Abbildung 4: Neues Feature einrichten

Die PostgreSQL-Lizenz

Wenn nun aber alles öffentlich zugänglich ist, wie schützt dann die PostgreSQL-Community ihr Produkt? Die Antwort ist so einfach wie bestechend: Gar nicht. Kann nicht sein? Die PostgreSQL-Lizenz steht unter *„https://opensource.org/licenses/postgresql“*. Der wichtigste Auszug daraus ist: „Permission to use, copy, modify, and distribute this software and its documentation for any purpose, without fee, and without a written agreement is hereby granted, provided that the above copyright notice and this paragraph and the following two paragraphs appear in all copies.” Was immer man auch mit dem Quellcode machen möchte, man darf es machen. Das geht so weit, dass es auch explizit erlaubt ist, ein eigenes Produkt aus dem Quellcode zu erstellen und dieses dann zu vermarkten, auch wenn es kein Open Source mehr ist. Mehr Freiheit geht nicht.

```
pg_basebackup --checkpoint=fast --write-recovery-conf -D /u02/pgdata/10/DEMOR
pg_ctl -D /u02/pgdata/10/DEMOR start
```

Listing 1

Firmen setzen schon heute ganz bewusst auf Open-Source-Produkte, auch wenn es sich meistens noch um Produkte rund um Linux handelt. Firmen wie Red Hat, SUSE oder Ubuntu haben es geschafft, ein Business-Modell um Open-Source-Technologien zu erstellen und davon leben zu können. Im Datenbank-Umfeld passiert momentan das Gleiche wie mit Linux in den letzten Jahren: Firmen suchen Anbieter, die Services und Produkte um Open-Source-Projekte anbieten und diese dann auch unterstützen.

Heute will kaum einer mehr auf Nicht-Open-Source-Produkte bauen, da die Entscheidungsprozesse nicht nachvollziehbar sind und Lizenz-Modelle sich entweder sehr schnell ändern oder erst gar nicht zu verstehen sind. PostgreSQL ist aus den genannten Gründen perfekt dafür gemacht, weil es keine Einschränkungen gibt. Das würde zwar alles nichts helfen, wenn PostgreSQL keine Features hätte, die in der heutigen Zeit zwingend erforderlich sind. Hat es aber.

PostgreSQL-Features

PostgreSQL bietet nahezu alle Features, die im Enterprise-Umfeld erforderlich sind. Das sind nicht genau die gleichen Features wie bei den Datenbank-Systemen der großen Hersteller, das gilt jedoch auch andersherum. Für 90 Prozent der heutigen Anforderungen bietet PostgreSQL alles, was es braucht – seien es zum Lesen geöffnete Replika-Datenbanken, seien es Backup und Point-in-time-Recovery, parallele Ausführung von SQL über mehrere Cores, Partitionierung, logische Replikation oder die schiere Anzahl von Datentypen *(siehe https://www.postgresql.org/docs/current/static/datatype.html")*. Eine Übersicht aller Features steht unter *„https://www.postgresql.org/about/featurematrix"*. Wichtiger als die Anzahl der Features ist jedoch der generelle Ansatz der PostgreSQL-Entwickler: Alle Features sollen benutzerfreundlich und einfach anzuwenden sein.

```
create table demo ( a int, b text, c timestamp );
with generator as
( select a.*
       , md5(a::text)
       , now()
    from generate_series (1,1000000) a
)
insert into demo
select * from generator;
```

Listing 2

```
copy demo to '/var/tmp/demo.out';
\! vi /var/tmp/demo.out
```

Listing 3

```
create table demo2 (like demo);
\h copy
copy demo2 from '/var/tmp/demo.out';
```

Listing 4

Am deutlichsten wird dieser Ansatz, wenn man sich ansieht, wie viele Schritte es braucht, um eine zum Lesen geöffnete Replika-Datenbank zu erstellen. Davon ausgehend, dass das auf dem gleichen Knoten ausgeführt wird, auf dem die Master-Datenbank läuft, sind exakt drei Schritte notwendig *(siehe Listing 1)*.

Der erste Schritt erstellt ein physisches Abbild der Master-Datenbank in einem neuen Verzeichnis, der zweite Schritt startet die Instanz. Mehr ist nicht notwendig und die Replika ist fertig und fährt alle Änderungen der Master-Datenbank nach. Um die Replika auf einem anderen Server zu erstellen, braucht es nur einen einzigen Schritt mehr: die Konfiguration der Master-Datenbank so anzupassen, dass von einem anderen Server auch eine Verbindung aufgebaut werden kann.

Ein weiteres Beispiel zeigt, wie einfach es ist, Daten in PostgreSQL hinein- beziehungsweise herauszuladen. Man gehe von einer simplen Tabelle mit 1.000.000 Einträgen wie dieser aus *(siehe Listing 2)*. Um diese Daten aus PostgreSQL in eine Datei zu schreiben, ist ein einziger Schritt notwendig *(siehe Listing 3)*. Diese Daten in eine andere Tabelle wieder einzulesen, ist ebenso einfach *(siehe Listing 4)*.

Ein anderes Beispiel sind innovative Daten-Typen wie Range Types. Eine häufige Anforderung von Applikationen besteht darin, einen Gültigkeits-Bereich zu definieren und auch abzufragen. In den meisten Datenbanken wird das über zwei Felder („gültig von" und „gültig bis") und einen Trigger gelöst. Der Trigger vergleicht die Werte, abhängig davon wird der neue Eintrag entweder zurückgewiesen oder erlaubt. In PostgreSQL lässt sich das mit einem Range-Datentyp und sogenannten „Exclusion Constaints" sehr einfach realisieren *(siehe „https://www.postgresql.org/docs/current/static/ddl-constraints.html#DDL-CONSTRAINTS-EXCLUSION")*. *Listing 5* zeigt ein Beispiel für ein einfaches Reservierungs-System für Sitzungszimmer.

Die zweite Tabelle verwendet einen Datentyp „tsrange" (Range von Timestamps) und einen Exclusion Constraint. Diese Kombination erlaubt es nun, Datums-Perioden zu vergleichen, wie es die

```
create table meeting_rooms ( id int primary key
                           , mname varchar(20)
                           , location varchar(10)
                           );
create table meeting_rooms_booked ( mid int references meeting_rooms(id)
                                  , booking_range tsrange
                                  , exclude using gist (mid with =,booking_range with &&)
                                  );
insert into meeting_rooms ( id, mname, location)
       values ( 1, 'meetingsouth', 'south' )
            , ( 2, 'meetingnorth', 'north' )
            , ( 3, 'meetingwest', 'west' )
            , ( 4, 'meetingeast', 'east' );
insert into meeting_rooms_booked ( mid, booking_range )
       values ( 1, '[2018-01-01 15:00, 2018-01-01 18:30]' )
            , ( 1, '[2018-01-01 08:00, 2018-01-01 08:30]' )
            , ( 2, '[2018-03-01 17:00, 2018-03-01 18:30]' )
            , ( 1, '[2018-03-01 05:00, 2018-03-01 08:30]' )
            , ( 3, '[2018-02-01 15:00, 2018-02-01 18:30]' )
            , ( 4, '[2018-02-01 19:00, 2018-02-01 20:30]' )
            , ( 4, '[2018-03-01 15:00, 2018-03-01 18:30]' );
select booking_range && '[2018-02-01 16:00,2018-02-01 16:30)'::tsrange
  from meeting_rooms_booked where mid = 3;
select booking_range && '[2018-02-01 18:45,2018-02-01 19:15)'::tsrange
  from meeting_rooms_booked where mid = 3;
```

Listing 5

```
du -sh $PGHOME
28M    /u01/app/postgres/product/10/db_3
```

Listing 6

beiden Statements am Ende bespielhaft demonstrieren. Das sind nur drei Beispiele und es gäbe noch etliche mehr, doch sie verdeutlichen sehr gut, wie PostgreSQL entwickelt wird: Egal, welches Feature aufgenommen wird, es soll einfach sein, es anzuwenden.

Klein, aber fein

Installiert man PostgreSQL vom Source Code, benötigt man auf der Platte schlussendlich zwischen 25 und 30 MB, je nach PostgreSQL-Version und je nachdem, was alles hineinkompiliert wurde. Das Beispiel in *Listing 6* ist ohne Dokumentation für PostgreSQL 10.3.

Das mag für DBAs, die mit Datenbank-Systemen anderer Hersteller arbeiten, lächerlich aussehen. Fakt ist aber, dass es besonders im Container-Umfeld eine große Rolle spielt. Es macht einen Unterschied, ob Container im Megabyte- oder im Gigabyte-Bereich verteilt werden. Container sind heute überall und es ist keine Überraschung, dass PostgreSQL in diesem Umfeld besonders häufig zum Einsatz kommt. Die Installation ist klein, Lizenz-Probleme gibt es nicht, es ist durch Erweiterungen extrem einfach auszubauen und die Features für die meisten Anwendungsgebiete sind alle vorhanden.

Fazit

Die momentane Richtung ist klar: Open-Source-Produkte kombiniert mit professionellem Support sind das, wo die meisten Firmen heute hinwollen. Das hat allerdings nicht nur Kostengründe, sondern auch viel mit der Qualität der Produkte zu tun. Zumindest bei PostgreSQL wird extrem viel Wert auf sauberen, wiederverwendbaren und vor allem wartbaren Code gelegt. Die Größe der Installation ist ein eindeutiger Beleg dafür. Diese Stabilität und Schlankheit wird heute mehr und mehr geschätzt und bei Container-Deployments auch in Zukunft eine große Rolle spielen.

Selbst, wenn man auf professionellen Support verzichten möchte (aus welchen Gründen auch immer), bietet die PostgreSQL-Community mit ihren Mailing-Listen sehr guten Support *(siehe „https://www.postgresql.org/list“)*. Wer zweifelt, möge sich bitte registrieren und eine Frage stellen.

Die Antwort auf die am Anfang des Artikels gestellte Frage „… und wird das auch so bleiben” lautet eindeutig „ja“. Das gilt nicht nur für PostgreSQL, sondern für alle Community-Projekte, die sich so entwickeln können, wie es die PostgreSQL-Community über die letzten zwanzig Jahre getan hat: Teilhabe einfach gestalten, jährliche Releases, eine aktive Community, immer hilfsbereit, aufgeschlossen und brandaktuell sowie mit einem Code of Conduct *(siehe „https://www.postgresql.org/about/policies/coc“)*. Dieses Projekt wird auch in Zukunft Erfolg haben.

Daniel Westermann
daniel.westermann@dbi-services.com