

マルチテナントアプリケーション

この経験では、Azure Database for PostgreSQL のハイパースケール(Citus)上でマルチテナントアプリケーションを作成するプロセスを説明します。

サービスとしてのソフトウェア (SaaS) アプリケーションを構築する場合は、データモデルにテナントの概念が既に組み込まれている可能性があります。通常、ほとんどの情報はテナント/顧客/アカウントに関連し、データベーステーブルはこの自然な関係をキャプチャします。

SaaS アプリケーションの場合、各テナントデータは1つのデータベースインスタンスにまとめて格納され、他のテナントから分離され、非表示に保つことができます。これは3つの点で効率的です。まず、アプリケーションの改善はすべてのクライアントに適用されます。次に、テナント間でデータベースを共有する場合、ハードウェアを効率的に使用します。最後に、すべてのテナントに対して、テナントごとに異なるデータベースサーバーよりも、すべてのテナントに対して1つのデータベースを管理する方がはるかに簡単です。

ただし、単一のリレーショナルデータベースインスタンスでは、従来、大規模なマルチテナントアプリケーションに必要なデータの量にスケーリングするのが困難でした。開発者は、データが単一のデータベースノードの容量を超えた場合に、リレーショナルモデルの利点を放棄することを余儀なくされました。

ハイパースケール(Citus)を使用すると、データベースが実際には水平方向にスケーラブルなマシクラスタであるのに、ユーザーは単一の PostgreSQL データベースに接続しているかのようにマルチテナントアプリケーションを作成できます。クライアントコードは最小限の変更のみを必要とし、完全な SQL 機能を引き続き使用できます。

このガイドでは、マルチテナントアプリケーションのサンプルを取り上げ、ハイパースケール (Citus) を使用してスケーラビリティを考慮してモデル化する方法について説明します。その過程で、ノイズの多い隣のテナントからテナントを分離する、より多くのデータに対応するハードウェアのスケーリング、テナント間で異なるデータの格納など、マルチテナントアプリケーションの一般的な課題について検討します。Azure Database for PostgreSQL のハイパースケール (Citus)は、これらの課題を処理するために必要なすべてのツールを提供します。では作成してみましょう。

1. このウインドウの右下にある **Next** をクリックします。

ハイパースケール(Citus)を使ってマルチテナントアプリケーションを作成する

最初に提供されたクレデンシャルを用いて Azure ポータルにログインします。

Azure ポータルにサイン・インする

□1. 既に Azure ポータルにログインしている場合、次のページに進みます。このウインドウの右下にある **Next** をクリックします。

□2. ブラウザで <https://portal.azure.com> を開き、ブラウザのウインドウを最大化します。

□3. **Pick an account** というダイアログが表示されたら、**+ Use another account** を選択します。

□4. **Sign in** ダイアログの、**Email, phone or Skype** フィールドに `xxxx@cloudplatimmersionlabs.onmicrosoft.com` を入力し **Next** をクリックします。

□5. **Password** フィールドに `xxxxxxx` を入力します。

□6. **Sign in** をクリックします。

□7. **Stay signed in?** とタイトルがついた **No** と **Yes** ボタンがあるポップアップが表示されるかもしれません。**No** を選択します。


□8. **Welcome to Microsoft Azure** とタイトルがついた **Start Tour** と **Maybe Later** ボタンがあるポップアップが表示されるかもしれません。**Maybe Later** を選択します。

□9. このウインドウの右下にある **Next** をクリックします。

ハイパースケール(Citus)の利用を始める

Azure Portal クラウド シェルを使用するには、ストレージ アカウントを作成する必要があります。ストレージ アカウントを使用すると、クラウド シェルに関連付けられたファイルを保存できるため、スクリプトを実行して Azure リソースを管理するさまざまな Azure ポータルアクティビティで使用できます。

Cloud Shell を作成する

- 1. ポータルのバナーで Cloud Shell のアイコンをクリックします。 
- 2. Welcome to Azure Cloud Shell で **Bash** をクリックします。
- 3. You have no storage mounted の画面で、**Show advanced settings** をクリックします。
- 4. サブスクリプションとリージョンのデフォルト値を使います。
- 5. リソースグループは既存の **rg000000** を使うようにしてください。
- 6. ストレージアカウントには、Create new を選択し、**sg000000shell** をペーストします。
- 7. ファイルシェアには、Create new を選択し、**sg000000shell** を入力してください。
- 8. Create Storage をクリックします。

注: Cloud Shell を作成・開始するのに 1 分程度を要します。

□9. 次の手順でファイアウォールを構成するには、Cloud Shell のクライアント IP アドレスが必要です。コマンド プロンプトで次のコマンドを入力し、return キーを押してから、クラウド シェルの IP アドレスをコピーまたはメモします。

```
curl -s https://ifconfig.co
```

注: bash コンソールでペーストするには右クリック後に paste を選択します。

- 11. このウィンドウの右下にある **Next** をクリックします。

PostgreSQL 用のハイパースケール(Citus)拡張の利用を始める

Azure Database for PostgreSQL のハイパースケール (Citus) サービスは、サーバーレベルでファイアウォールを使用します。既定では、ファイアウォールはすべての外部アプリケーションとツールがコーディネータノードおよび内部のデータベースに接続するのを防ぎます。特定の IP アドレス範囲のファイアウォールを開くルールを追加する必要があります。

このラボでは、4 つの vCore と 16 GB の RAM を備えた 1 つのコーディネータと 2 つのワーカーを備えた基本的な本番グレードのハイパースケール (Citus) クラスターを事前にプロビジョニングしました。

サーバーレベルのファイアウォールのルールを設定する

- ☐1. Azure ポータルの左上にある **Home** をクリックします。
- ☐2. Azure サービスの下にある **Azure Database for PostgreSQL servers** をクリックします。
- ☐3. **sgxxxxxx** をクリックします。
- ☐4. Security の下の Overview ペインの左のナビゲーションで **Firewall** をクリックします。
- ☐5. Cloud Shell で確認した IP アドレスを **START IP** と **END IP** に入力します。
- ☐6. **FIREWALL RULE NAME** に **CloudShell** と入力します。
- ☐7. ペインの左上にある **Save** をクリックします。

注: ハイパースケール(Citus)サーバはポート 5432 を介して通信します。企業ネットワーク内から接続しようとしている場合、ポート 5432 を超える送信トラフィックは、ネットワークのファイアウォールで許可されない場合があります。その場合は、IT 部門がポート 5432 を開かない限り、ハイパースケール (Citus) サーバーに接続できません。

- ☐8. このウインドウの右下にある **Next** をクリックします。

Azure Database for PostgreSQL のハイパースケール (Citus)に接続する

ハイパースケール (Citus) を作成すると、**citus** という名前の既定のデータベースが作成されます。データベースサーバーに接続するには、接続文字列と管理者パスワードが必要です。最初の接続には最大 2 分かかる場合があります。何らかの理由でシェルがタイムアウトして再起動した場合は、`curl -s https://ifconfig.co` コマンドをもう一度実行し、ファイアウォールが新しい IP アドレスで更新されていることを確認する必要があります。

Psql でデータベースに接続する

□1. Cloud Shell の右上にある**最大化**ボックスをクリックして全画面にします。

□2. Bash プロンプトで、Psql ユーティリティを用いて Azure Database for PostgreSQL に接続します。最初の接続には最大 2 分かかる場合があります。以下のコマンドをコピー＆ペーストして[enter]を押します。

```
psql "host=sg000000-c.postgres.database.azure.com port=5432 dbname=citus user=citus
password='xxxxxxxx' sslmode=require"
```

□3. このウィンドウの右下にある **Next** をクリックします。

広告業向けのマルチテナントアプリケーションを作成する

顧客がオンライン広告のパフォーマンスを追跡できるようにするマルチテナント SaaS アプリケーションのバックエンドの例を構築します。ユーザーはある瞬間に自らの会社（自分の会社）に関連するデータをリクエストするので、マルチテナントアプリケーションが向いているのは当然のことです。

このマルチテナント SaaS アプリケーションの簡略化されたスキーマを検討することから始めましょう。アプリケーションは、広告キャンペーンを実行する複数の企業を追跡する必要があります。キャンペーンには多数の広告があり、各広告にはクリック数とインプレッションの記録が関連付けられています。

以下はスキーマの例です。

□1. Psql コンソールに以下の CREATE TABLE コマンドをコピー & ペーストして会社（テナント）とそのキャンペーンのテーブルを作成します。

```
CREATE TABLE companies (  
  id bigserial PRIMARY KEY,  
  name text NOT NULL,  
  image_url text,  
  created_at timestamp without time zone NOT NULL,  
  updated_at timestamp without time zone NOT NULL  
);  
  
CREATE TABLE campaigns (  
  id bigserial,  
  company_id bigint REFERENCES companies (id),  
  name text NOT NULL,  
  cost_model text NOT NULL,  
  state text NOT NULL,  
  monthly_budget bigint,  
  blacklisted_site_urls text[],  
  created_at timestamp without time zone NOT NULL,  
  updated_at timestamp without time zone NOT NULL,  
  PRIMARY KEY (company_id, id)  
);
```

□2. Psql コンソールに以下の CREATE TABLE コマンドをコピー & ペーストして会社の広告のテーブルを作成します。

```
CREATE TABLE ads (  
  id bigserial,  
  company_id bigint,  
  campaign_id bigint,  
  name text NOT NULL,  
  image_url text,  
  target_url text,  
  impressions_count bigint DEFAULT 0,  
  clicks_count bigint DEFAULT 0, created_at timestamp without time zone NOT NULL,  
  updated_at timestamp without time zone NOT NULL,  
  PRIMARY KEY (company_id, id),  
  FOREIGN KEY (company_id, campaign_id)  
    REFERENCES campaigns (company_id, id)  
);
```

□3. Psql コンソールに以下の CREATE TABLE コマンドをコピー & ペーストして各広告のクリック数とインプレッション数の状態を追跡します。

```
CREATE TABLE clicks (  
  id bigserial,  
  company_id bigint,  
  ad_id bigint,  
  clicked_at timestamp without time zone NOT NULL,  
  site_url text NOT NULL,  
  cost_per_click_usd numeric(20,10),  
  user_ip inet NOT NULL,  
  user_data jsonb NOT NULL,  
  PRIMARY KEY (company_id, id),  
  FOREIGN KEY (company_id, ad_id)  
    REFERENCES ads (company_id, id)  
);  
  
CREATE TABLE impressions (  
  id bigserial,  
  company_id bigint,  
  ad_id bigint,  
  seen_at timestamp without time zone NOT NULL,  
  site_url text NOT NULL,  
  cost_per_impression_usd numeric(20,10),  
  user_ip inet NOT NULL,  
  user_data jsonb NOT NULL,  
  PRIMARY KEY (company_id, id),  
  FOREIGN KEY (company_id, ad_id)  
    REFERENCES ads (company_id, id)  
);
```

□4. Psql コンソールに以下の CREATE TABLE コマンドをコピー & ペーストして今作成したテーブルを確認します。

```
¥dt
```

マルチテナントアプリケーションはテナントごとにのみ一意性を適用できるため、すべての主キーと外部キーに会社 ID が含まれます。この要件により、分散環境ではこれらの制約を強制的に適用する必要が生じます。

□5. このウィンドウの右下にある **Next** をクリックします。

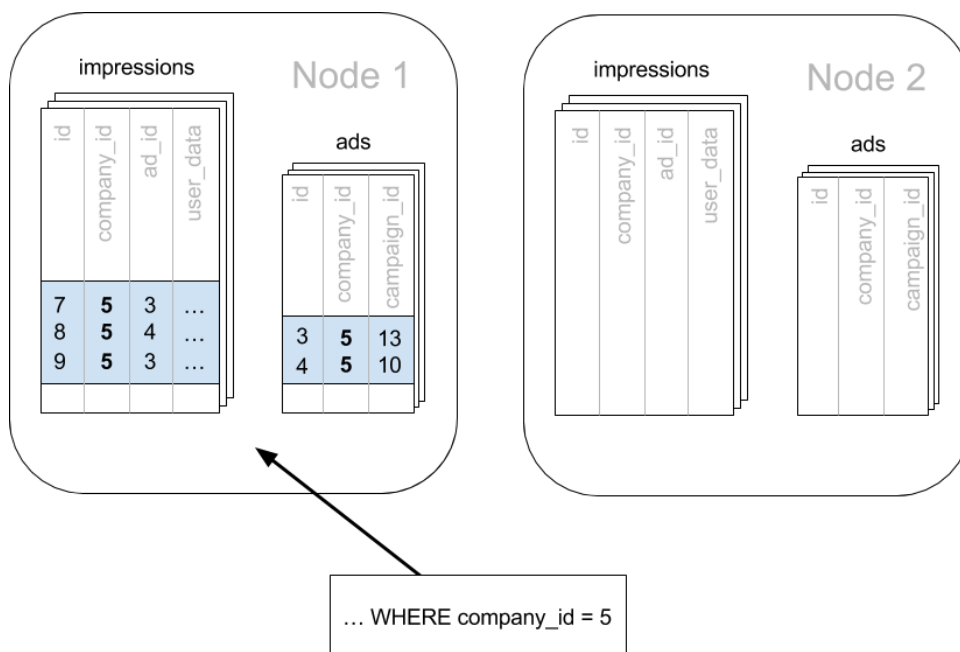
リレーショナルデータモデルをスケールする

リレーショナル データ モデルは、アプリケーションに最適です。データの整合性を保護し、柔軟なクエリを可能にし、変化するデータに対応します。従来の唯一の問題は、リレーショナル データベースが、大規模な SaaS アプリケーションに必要なワークロードにスケーリングできないと考えられていたことです。開発者は NoSQL データベースを受け入れるか、そのサイズに到達するまでバックエンド サービスの塊を受け入れるしかありませんでした。

ハイパースケール(Citus)を使用すると、データモデルを維持し、スケールすることができます。ハイパースケール (Citus) はアプリケーションに単一の PostgreSQL データベースとして表示されますが、内部的には、要求を並列処理できる調整可能な数の物理サーバー (ノード) にクエリをルーティングします。

マルチテナント アプリケーションには、通常、クエリはテナントの組み合わせではなく、一度に 1 つのテナントに対して情報を要求するという、優れたプロパティを持っています。たとえば、営業担当者が CRM で見込顧客情報を検索する場合、検索結果は雇用主に固有であって、その他の見込み案件情報およびメモは含まれません。

アプリケーション クエリは店舗や会社などの単一のテナントに制限されているため、マルチテナント アプリケーション クエリを高速化する方法の 1 つは、特定のテナントのすべてのデータを同じノードに格納することです。これにより、ノード間のネットワークオーバーヘッドが最小限に抑えられ、ハイパースケール (Citus) がすべてのアプリケーションの結合、キー制約、およびトランザクションを効率的にサポートできるようになります。これにより、アプリケーションを完全に書き直したり再設計したりしなくても、複数のノードにまたがってスケーリングできます。



ハイパースケール (Citus) では、テナントに関連するすべてのテーブルに、どのテナントがどの行を所有しているかを明確にマークする列があることを確認します。広告運用分析アプリケーションでは、テナントは会社なので、会社に関連するすべてのテーブルに `company_id` 列が含まれるようにする必要があります。これらのテーブルは分散テーブルと呼ばれています。たとえば:キャンペーンは企業向けであるため、キャンペーンテーブルには `company_id` 列が必要です。広告、クリック、インプレッションについても同じです。

したがって、この例では `company_id` は “distribution column” (シャーディング キーまたは分散キーとも呼ばれます) になります。つまり、`company_id` を使用して、ワーカー ノード間のすべてのテーブルをシャード / 分散します。これにより、すべてのテーブルが結び付きます。つまり、すべてのテーブルの 1 つの会社に関連するすべてのデータが同じワーカー ノード上にあります。このようにして、ハイパースケール (Citus) に対して、同じ会社の行がマークされている場合に、この列を使用して同じノードに行を読み取りおよび書き込むように伝えることができます。たとえば、上記の図では、`company_id` 5 のすべてのテーブルのすべての行が同じワーカー ノード上にあります。

この時点で、SQL をダウンロード・実行してスキーマを作成することにより、あなた専用のハイパースケール (Citus) クラスタを目で追ってみましょう。スキーマの準備ができたなら、ハイパースケール (Citus) にワーカーノードにシャードを作成するように伝えることができます。

ノード間にテーブルをシャードする

`create_distributed_table` 関数は、テーブルをノード間で分散する必要があり、それらのテーブルへの将来の着信クエリを分散実行用に計画する必要があることを Hyperscale (Citus) に通知します。この関数は、ワーカー ノード上のテーブルのシャードも作成します。

この具体的な例では、`company_id` に基づいてデータベース全体でテーブルをシャーディングして作成したアプリケーションをスケーリングします。

□1. Psql コンソールに以下をコピー＆ペーストして分散キー (シャード) を作成します。

```
SELECT create_distributed_table('companies',
SELECT create_distributed_table('campaigns',
SELECT create_distributed_table('ads',
SELECT create_distributed_table('clicks',
SELECT create_distributed_table('impressions', 'company_id');
```

□2. このウィンドウの右下にある **Next** をクリックします。

データを取得する

前のセクションでは、マルチテナント アプリケーションの正しい分散列を特定しました：company id。単一コンピューター データベースでも、行レベルのセキュリティや追加のインデックス作成のいずれであっても、company id を追加してテーブルを非正規化すると便利です。他の利点は、追加の列を含めると、マルチマシンのスケーリングにも役立ちます。

次の手順では、サンプル データをコマンド ラインからクラスターに読み込みます。

シェルでデータをダウンロードし取得する

□1. Psql コンソールに以下をコピー & ペーストしてサンプルデータをダウンロードします。

```
¥! curl -O https://examples.citusdata.com/mt_ref_arch/companies.csv
¥! curl -O https://examples.citusdata.com/mt_ref_arch/campaigns.csv
¥! curl -O https://examples.citusdata.com/mt_ref_arch/ads.csv
¥! curl -O https://examples.citusdata.com/mt_ref_arch/clicks.csv
¥! curl -O https://examples.citusdata.com/mt_ref_arch/impressions.csv
¥! curl -O https://examples.citusdata.com/mt_ref_arch/geo_ips.csv
```

PostgreSQL の拡張機能であるハイパースケール(Citus)は COPY コマンドを使用した一括読み込みをサポートします。ダウンロードしたデータを取り出し、ファイルを他の場所にダウンロードした場合は、正しいファイル パスを指定してください。

□2. Psql コンソールに以下をコピー & ペーストしてテーブルをロードします。

```
¥copy companies from 'companies.csv' with csv
¥copy campaigns from 'campaigns.csv' with csv
¥copy ads from 'ads.csv' with csv
¥copy clicks from 'clicks.csv' with csv
¥copy impressions from 'impressions.csv' with csv
```

□3. このウィンドウの右下にある **Next** をクリックします。

テナントデータをクエリする

`company_id` のフィルターを含むアプリケーション クエリまたは更新ステートメントは、引き続きそのまま動作します。前述のように、この種のフィルタはマルチテナント アプリで一般的です。オブジェクト リレーショナル マッパー (ORM) を使用する場合、これらのクエリは、場所やフィルタなどの方法によって認識できます。

基本的に、データベースで実行される結果の SQL に、すべてのテーブル (JOIN クエリ内のテーブルを含む) に **WHERE `company_id = value`** 句が含まれている場合、Hyperscale (Citus) はクエリを単一のノードにルーティングし、その状態で実行する必要があることを認識します。これにより、すべての SQL 機能を使用できます。結局のところ、各ノードは通常の PostgreSQL サーバです。

`company_id = 5` の単一のテナントのクエリ

アプリケーションが単一のテナントのデータを要求すると、データベースは単一のワーカー ノードでクエリを実行できます。シングル テナント クエリは、単一のテナント ID でフィルター処理します。たとえば、次のクエリは、広告とインプレッションに対して `company_id = 5` をフィルター処理します。

□1. Psq| コンソールに以下をコピー & ペーストして単一のテナントのクエリと更新を実行します。

```
-- campaigns with highest budget

SELECT name, cost_model, state, monthly_budget
FROM campaigns
WHERE company_id = 5
ORDER BY monthly_budget DESC
LIMIT 10;

-- double the budgets

UPDATE campaigns
SET monthly_budget = monthly_budget*2
WHERE company_id = 5;
```

NoSQL データベースを使用してアプリケーションをスケーリングするユーザーの一般的な問題点は、トランザクションと結合の欠如です。ただし、トランザクションはハイパースケール (Citus) で期待どおりに機能します。

□2. Psql コンソールに以下をコピー & ペーストしてトランザクションである更新を実行します。

```
-- transactionally reallocate campaign budget money
BEGIN;

UPDATE campaigns
SET monthly_budget = monthly_budget + 1000
WHERE company_id = 5
AND id = 40;

UPDATE campaigns
SET monthly_budget = monthly_budget - 1000
WHERE company_id = 5
AND id = 41;

COMMIT;
```

SQL サポートの最後のデモとして、集計関数とウィンドウ関数を含むクエリがあり、PostgreSQL の場合と同じように Hyperscale (Citus) で動作します。クエリは、各キャンペーンの広告をインプレッション数でランク付けします。

□3. Psql コンソールに以下をコピー & ペーストしてデータベースをまたぐ集計クエリを実行します。

```
SELECT a.campaign_id,
       Rank()
FROM
  OVER (
    partition BY a.campaign_id
    ORDER BY a.campaign_id, Count(*) DESC ),
Count(*) AS n_impressions, a.id
  ads AS a
JOIN impressions AS i ON i.company_id = a.company_id
  AND i.ad_id = a.id
WHERE a.company_id = 5
GROUP BY a.campaign_id, a.id
ORDER BY a.campaign_id, n_impressions DESC
Limit 20;
```

注: 結果ビューでスタックした場合は、「q」と入力し、「Enter」を押してビューモードを終了します。

要するに、クエリがテナントにスコープを設定すると、挿入、更新、削除、複雑な SQL、およびトランザクションがすべて期待どおりに動作します。

□4. このウィンドウの右下にある **Next** をクリックします。

参照テーブルでテナント間でデータを共有する

これまで、すべてのテーブルは `company_id` によって分散されていましたが、すべてのテナントで共有できるデータがあり、特にどのテナントにも“所属”が存在しない場合があります。たとえば、この例の広告プラットフォームを使用するすべての企業は、IP アドレスに基づいてオーディエンスの地理情報を取得する必要があります。単一のマシン データベースでは、次のような `geo-ip` のルックアップ テーブルによってこれを実現できます。(実際のテーブルはおそらく PostGIS を使用しますが、簡略化された例に準拠しています)。

共有の地理情報を保持するテーブルを作成します。

□1. Psq| コンソールに以下をコピー＆ペーストして `geo_ips` テーブルを作成します。

```
CREATE TABLE geo_ips (  
  addrs cidr NOT NULL PRIMARY KEY,  
  latlon point NOT NULL  
    CHECK (-90 <= latlon[0] AND latlon[0] <= 90 AND  
           -180 <= latlon[1] AND latlon[1] <= 180)  
);  
CREATE INDEX ON geo_ips USING gist (addrs inet_ops);
```

分散セットアップでこのテーブルを効率的に使用するには、`geo_ips` テーブルをクリック数、それも 1 つだけではなく、すべての会社の、と共に再配置する方法を見つける必要があります。この方法では、クエリ時にネットワーク トラフィックは発生しません。これをハイパースケール (Citus)で行うには、`geo_ips` をすべてのワーカー ノードにテーブルのコピーを格納する参照テーブルとして指定します。

□2. Psq| コンソールに以下をコピー＆ペーストして `geo_ips` テーブルを作成します。

```
-- Make synchronized copies of geo_ips on all workers  
  
SELECT create_reference_table('geo_ips');
```

参照テーブルはすべてのワーカー ノードにレプリケートされ、ハイパースケール (Citus) は変更時に自動的に同期を維持します。`create_distributed_table` ではなく、`create_reference_table` と呼ばれることに注意してください。

□3. Psq| コンソールに以下をコピー＆ペーストして `geo_ips` にデータをロードします。

```
¥copy geo_ips from 'geo_ips.csv' with csv
```

これで、このテーブルをクリック数を結合すると、すべてのノードで効率的に実行されます。広告 290 をクリックしたすべてのユーザーの場所を尋ねることができます。

□4. Psql コンソールに以下をコピー & ペーストして SELECT を実行します。

```
SELECT c.id, clicked_at, latlon  
FROM geo_ips, clicks c  
WHERE addr >> c.user_ip  
AND c.company_id = 5  
AND c.ad_id = 290;
```

□5. このウインドウの右下にある **Next** をクリックします。

スキーマのオンライン変更

マルチテナント システムのもう 1 つの課題は、すべてのテナントのスキーマの同期を維持することです。スキーマの変更は、すべてのテナントに一貫して反映する必要があります。ハイパースケール (Citus) では、標準の PostgreSQL DDL (データ定義言語) コマンドを使用してテーブルのスキーマを変更するだけで、ハイパースケール (Citus) は 2 フェーズ コミット (2PC) プロトコルを使用してコーディネータ ノードからワーカーに伝播します。

たとえば、このアプリケーションの提供情報は、キャプション テキスト列を使用できます。コーディネータで標準 SQL を発行することで、テーブルに列を追加できます。

□1. Psql コンソールに以下をコピー & ペーストして新しい列を追加します。

```
ALTER TABLE ads  
ADD COLUMN caption text;
```

これにより、すべてのワーカーノードも更新されます。このコマンドが完了すると、ハイパースケール (Citus) クラスタは、新しいキャプション列のデータの読み取りまたは書き込みを行うクエリを受け入れます。

□2. このウィンドウの右下にある **Next** をクリックします。

テナント間でデータが異なる時

各テナントは、他のテナントが必要としない一意の情報を格納する必要がある場合があります。たとえば、広告データベースを使用するテナント アプリケーションの 1 つでクリック トラッキング情報を保存する場合がありますが、別のテナントがブラウザ エージェントを必要とする場合があります。ただし、すべてのテナントは、同じデータベース スキーマを持つ共通のインフラストラクチャを共有します。

従来、マルチテナントに共有スキーマアプローチを使用するデータベースは、事前に割り当てられた “custom” 列の固定数を作成するか、外部の “extension” テーブルを持つことに頼っていました。非構造化列の種類、特に JSONB です。JSONB (B はバイナリ用) は、NoSQL データベースの利点の一部を提供し、より高速なクエリに対してインデックスを作成することもできます。

この例では、テナント (会社 5 など) は、JSONB 列を使用して、ユーザーがモバイル デバイス上にあるかどうかを追跡します。

□1. Psql コンソールに以下をコピー & ペーストしてモバイルデバイスである会社 5 のユーザを検索します。

```
SELECT
user_data->>'is_mobile' AS is_mobile,
count(*) AS count
FROM clicks
WHERE company_id = 5
GROUP BY user_data->>'is_mobile'
ORDER BY count DESC;
```

□2. Psql コンソールに以下をコピー & ペーストして**部分インデックス**を作成することによりクエリーを最適化します。

```
CREATE INDEX click_user_data_is_mobile
ON clicks ((user_data->>'is_mobile'))
WHERE company_id = 5;
```

PostgreSQL は JSONB 列の **GIN** インデックスをサポートしています。JSONB 列に GIN インデックスを作成すると、その JSON ドキュメント内のすべてのキーと値にインデックスが作成されます。これにより、?.?|、?&などのいくつかの JSONB 演算子が高速化されます。

□3. Psql コンソールに以下をコピー & ペーストして GIN インデックスを作成します。

```
CREATE INDEX click_user_data
ON clicks USING gin (user_data);

-- this speeds up queries like, "which clicks have
-- the is_mobile key present in user_data?"

SELECT id
FROM clicks
WHERE user_data ? 'is_mobile'
AND company_id = 5
Limit 5;
```

□4. このウィンドウの右下にある **Next** をクリックします。

結論

これで、どのように Azure Database for PostgreSQL のハイパースケール(Citus)で、マルチテナントアプリケーションがスケールするようにする方法が分かりました。

このチュートリアルでは、Azure Database for PostgreSQL のハイパースケール(Citus)を使い以下の方法を学びました。

- ハイパースケール(Citus)サーバグループの作成
- Psql ユーティリティを使ったスキーマの作成
- ノード間でのテーブルのシャード
- サンプルデータの取得
- テナントデータのクエリー
- テナント間でのデータの共有
- テナント毎のスキーマのカスタマイズ

追加のドキュメントにも興味があるかもしれません。

- Migration Guides
 - Ruby on Rails (https://docs.citusdata.com/en/v8.1/develop/migration_mt_ror.html#rails-migration)
 - Django (https://docs.citusdata.com/en/v8.1/develop/migration_mt_django.html#django-migration)
- マルチテナントフィルタを全てのクエリに自動的に追加するライブラリ
 - Rails 用の `activerecord-multi-tenant` (<https://github.com/citusdata/activerecord-multi-tenant>) ライブラリ
 - Django 用の `django-multitenant` (<https://github.com/citusdata/django-multitenant>) ライブラリ

1. この経験がどのくらい素晴らしかったか **Feedback** をクリックして教えてください。