ORACLE DATABASE 12c

Oracle Multitenant：新機能

Oracle Database 12c Release 2（12.2）

Oracle ホワイト・ペーパー | 2017 年 3 月

ORACLE

免責事項

下記事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。マテリアルやコード、機能の提供をコミットメント（確約）するものではなく、購買を決定する際の判断材料になさらないで下さい。オラクルの製品に関して記載されている機能の開発、リリース、および時期については、弊社の裁量により決定されます。

Oracle Database 12c Release 2が利用可能に。

世界でもっとも多用されているデータベースの最新世代である Oracle Database 12c Release 2（12.2）が Oracle Cloud で利用可能になり、Oracle Technology Network（OTN）からダウンロードできるようになりました。

目次

規模の経済性（スケールメリット）を伴う独立性と機敏性

重要なのは規模だけではありません。規模そのものはマイナスとして捉えられることもあります。1 つのオフィスでは不足するほど組織の規模が大きくなると、コミュニケーションが困難になるうえ管理階層を増やすことも必要になります。石油タンカーが方向転換するところを思い浮かべるとよくわかりますが、機敏に動くことが困難になるのです。こうした問題からわかるのは、規模が大きくなるとコストが指数関数的に増加しかねないことです。

第 1 世代のクラウドに関連するテクノロジーは自動化が進んでいるため、このコスト計算式は一次関数程度まで改善できます。開発環境を例にして考えてみましょう。10 人の開発者をサポートするには 10 台の仮想マシン（VM）が、20 人の開発者をサポートするには 20 台の VM が必要になります。それほどのワークロードがかかっていなくても、本質的に VM ごとにコストが発生します。そのため、コストと規模の間には線形の関係があります。とはいえ、これは正しい方向に進むうえで重要なステップでした。

Oracle Multitenant は次世代のデータベース・クラウドのためのアーキテクチャです。Oracle Multitenant は現状を一変させ、真のスケールメリットを実現します。プラガブル・データベース（PDB）がデータベースを搭載した VM というコストのかさむモデルに取って代わります。PDB 固有のコストはほとんどないため、各開発者の PDB のコストは実際の作業量の分だけで済みます。すべての開発者の PDB を 1 つのマルチテナント・コンテナ・データベース（CDB）に統合することができ、その CDB を実行するコストはすべての開発者で共有できます。コンピューティング・リソースの観点から見ると、バックグラウンド・プロセスが 1 セット、単一の共有メモリ領域（SGA）が 1 つだからです。管理の意味では、バックアップ、高可用性の構成、パッチの適用などが必要な CDB は 1 つだからです。

独立性と機敏性を犠牲にした統合であれば、その統合によるスケールメリットは限定的です。Oracle Database 12c Release 2（12.2）のマルチテナント・アーキテクチャは既存の強力な分離機能をベースに構築された包括的なモデルですが、簡単な構成だけで特定のユースケースに適した独立性のレベルを正確に実現できます。優れた独立性があり、統合される個々のワークロードが干渉し合うことがないため、大規模な統合が現実的になります。

機敏性とは、タンカーの方向転換のたとえに見られる統合のやっかいな影響を、良い意味で避けることと言えます。規模が大きければ強大な購買力を活かして単価を削減できますが、機敏性が損なわれるのであれば、そうして得られたメリットは相殺されます。データベース用語でいう機敏性は、データベースを作成、クローニング（さらには破棄）する速度と効率性を指します。つまり、サーバー間、データセンター間、データセンターとクラウド間でのデータベースの移動に関する話であり、クラウド内での移動に関する話です。移動する理由としては、設備を廃棄してクラウドの機能に置き換える、ロードバランシングのため、サービス・レベルを変更するため、ピーク時の負荷に対処するためなど、あらゆることが考えられます。現代の企業はこうしたことを必ずできる必要がありますが、Oracle Multitenant の高度な機能を使用すれば、可能な限り影響を抑えてこれらを実行できます。

スケールメリットを考える上で重要なのは、スケールデメリットよりも削減効果が上回って初めて真の投資効果がある点です。Oracle Multitenant が優れている点は、スケールメリットが非常に大きいため他のモデルよりもはるかに容易に投資効果が得られることです。Oracle Multitenant は真のスケールメリットをもたらし、2 つの側面を両立させます。

Oracle Multitenant は***スケールメリットを伴う独立性と機敏性を実現します***。

12.1のOracle Multitenant

Oracle Multitenant を使用すると、旧リリースの完全に機能する製品を手にしながら新リリースに搭載されている膨大な新機能を同時に手にするという矛盾した状況に陥ります。新リリースの新機能について詳しく説明する前に、Oracle Database 12c Release 1（12.1）の既存のマルチテナント・アーキテクチャとそのおもな利点について簡単に説明します。

Oracle Multitenant を使用すると、複数のプラガブル・データベース（PDB）を 1 つのマルチテナント・コンテナ・データベース（CDB）に統合できます。プラガブル・データベースは完全に機能する自己完結型の Oracle Database です。アプリケーションの観点からは何も変わっていませんが、これが非常に重要なところです。このアーキテクチャを採用するときにアプリケーションを変更する必要がないからです。したがって、アプリケーションの観点からすると PDB がデータベースです。ところが、運用の観点からデータベースと呼べるのは CDB です。CDB は統合された 1 つの運用環境と言えます。そこには 1 セットのバックグラウンド・プロセスと 1 つの共有メモリ領域（SGA）があり、これらは CDB 内のすべての PDB によって共有されます。このアーキテクチャを採用するとオーバーヘッドが繰り返し発生することがなくなるため、使用可能なリソースをもっとも効率的に使用できます。つまり、1 台のサーバーに統合できるアプリケーションの数が増えるため、設備投資（CapEx）を最小化できます。運用の観点からすると、統合されたすべての PDB をまとめて管理できるため、運用コスト（OpEx）を大幅に削減できることになります。これが該当するのは、バックアップ、高可用性の構成、パッチの適用、アップグレードなどです。このように CapEx と OpEx が削減されることは、Oracle Multitenant を使用することでもたらされるクラウド・コンピューティングのメリットの一部です。

このアーキテクチャは、これから紹介する 3 つの主要なユースケースに特に適しています。

統合

数年前の重大な関心事は、ワークロードを管理するためにサーバー（およびそこで実行されるソフトウェア）を“スケールアップ”できるかどうかでしたが、Oracle Exadata Database Machine などの最新の統合インフラストラクチャの処理能力は相当なものであり、実際、クラウド環境はそのようなマシンで構成されていることから、“スケールダウン”という言葉が聞かれるようになりました。つまり、ほとんどの場合、大規模なアプリケーションでもいくつかを 1 つのサーバーに統合することが十分に可能です。スケールメリットを得るうえで難しいのは、多様なワークロードを効率的にサポートするために、過剰にプロビジョニングすることなくサーバーを“スケールダウン”しつつ、スタンドアロンで実行するそもそもの理由であった独立性を維持できるようにすることです。PDB という本質的に独立したものを使用する Oracle Multitenant は、このユースケースに最適です。Resource Manager は Oracle Multitenant と完全に一体化するため、シンプルながらも強力なポリシーを定義して PDB 間のリソース配分を管理できます。12.1 では、最大 252 個のプラガブル・データベースを 1 つのマルチテナント・コンテナ・データベースに統合できます。1 つの CDB に PDB が 252 個とは、驚くべき数です。

開発およびテスト

開発やテストを担当する現代の組織に必要とされる機敏性は、クラウド・コンピューティングのもう 1 つの重要な特徴です。多数のデータベースを一括管理しつつも、必要に応じてきめ細かく制御できるため、機敏性を強化できます。PDB のプロビジョニングは極めて高速かつシンプルであるため、Database as a Service（DBaaS）に不可欠なセルフサービス・プロビジョニング・インタフェースに適しています。PDB のプロビジョニングは、クローニングというプロセスによって達成されます。クローニングには、完全クローン、サブセット・クローン、そしてスナップショット・クローンのシン・プロビジョニング（Copy-on-Write テクノロジーを利用）など、多くの種類があります。プラガブル・データベースは移植可能なデータベースです。切断/接続と呼ばれるプロセスを使ったサーバー間の PDB の移動は、非常に簡単で迅速です。ロードバランシングやクラウド間の移行などにはこれを使用します。

Oracle Real Application Clusters（Oracle RAC）とシームレスに統合でき、PDB をクラスタ内の特定のノードに紐付けることや複数のノードで一様に実行することができるため、ワークロードの変化に柔軟に適合させられます。前述したシンプルなセルフサービス・プロビジョニングと同様、これもデータベース・クラウドの重要な特徴です。

Software as a Service

Software as a Service（SaaS）はクラウド・コンピューティングの代表的な例であるため、Oracle Multitenant の代表的なユースケースでもあります。“士気をくじかれた 20 世紀のアプリケーション・ベンダー”とでも呼べるようなベンダーにとって、Oracle Multitenant は即席の SaaS アーキテクチャになります。旧式のアプリケーションは一般的に機能が非常に豊富ですが、時代遅れでスタンドアロンのオンプレミス・デプロイメント・モデル向けに構築されているため、機能的に優れていたとしても最新の SaaS アーキテクチャには太刀打ちできません。ですが、Oracle Multitenant によって状況はがらりと変わります。テナントごとに PDB に隔離しながら 1 つの CDB に複数のテナントを統合できるため、旧式のアプリケーションを SaaS 構成内で簡単に展開しなおすことができます。変更は一切不要です。

12.2の新機能

Oracle Database 12c の新しいリリースである Release 2（12.2）で提供している新機能はすべて、この極めて強力な基盤上で開発されました。これら 3 つの領域のそれぞれで、大幅に機能が強化されています。ここから先は、強化されたさまざまな機能について詳しく説明します。

12.1 の Oracle Multitenant について詳しくは、12.1 の Oracle Multitenant について書かれているホワイト・ペーパー（2013 年発行）を参照してください。

プロビジョニングの機能強化

データベース管理者（DBA）は従来からデータベースのプロビジョニングに頭を悩ませてきました。現代の企業は絶えずイノベーションを迫られているため、開発チームはモバイル機能、ソーシャル・メディアとのインタラクション機能、いつでもどこでも使えるクラウドなど、現代のユーザーが求める機能を搭載した新しいシステムを継続的に要求されます。至る所でインターネットがハッキングされる現代の環境では、そうしないと存続し続けていくことができません。しかも、規制要件や厳しいセキュリティを遵守することも必要です。これに加えてさらに求められるのが、スケーラビリティ、可用性、パフォーマンス、管理性、生産性といった基本的な技術です。しかしこれら

は、従来から Oracle Database に備わっている圧倒的な強みです。

最近はやりの"DevOps"では、絶えず開発し、絶えず統合し、絶えずリリースするという手法を取りますが、これがこのプレッシャーをさらに深刻にしています。前述した特質をすべて備えた開発環境の基本層にあるのは Oracle Database だからです。すでに説明したように、12.1 で Oracle Multitenant が登場したことでこの状況はがらりと変わりました。Oracle Multitenant を使用すると、PDB をクローニングするだけで新しいデータベースをプロビジョニングできます。これを最新の DBaaS モデルに組み込めば、データベースをプロビジョニングするプロセスは一変し、複数チームの担当者を巻き込んで数日または数週間（場合によっては数か月）をかけて実施されたかもしれないプロセスが、簡単なセルフサービス・インタフェースを使って数分あるいは数秒で完了します。

12.2 では、こうした基本的なプロビジョニング操作を大幅に改良するために、オンラインで実行できるようにしました。

PDBのクローニング

12.1 で PDB のクローンを作成する場合は、クローニング操作が終わるまでソース PDB を静止させておく必要があります。（トランザクションの観点からすると）ソース・データベースの停止が伴うため、この操作は"コールド・クローン"操作と呼ばれます。"ホット・クローン"の方が望ましいと言うのは簡単ですが、言うは易く行うは難しです。ところが、12.2 ではホット・クローンがサポートされています。このオンライン・クローニング機能は、Oracle Database が対応するすべてのストレージ上で使用できます。

ホット・クローンは、ソース PDB を読取り/書込みで開いたままで実行できます。つまり、ソース PDB 側の操作を中断せずに実行が可能です。ホット・クローンを実行するためにアプリケーションを停止する必要はありません。

ここからは、ホット・クローンの操作中に"背後で"実行されている技術的な処理を簡単に説明します。これは、好奇心が強い方のためですので、細かいことに関心がない方は、次の「***リフレッシュ可能 PDB***」の項まで読み飛ばしていただいて構いません。

ホット・クローンで実行している技術的操作は、ソース PDB のデータファイルに含まれる全ブロックの"ファジー読取り"と呼ばれます。つまり、クローン操作が時刻 ***t0*** に開始された場合、ソース PDB の最後のブロックが読み取られてターゲットにコピーされた頃（時刻 ***t1***）には、すでにコピーされたブロックの一部に何らかの変更が加えられている可能性があるという意味です。したがって、この段階では、クローンとソースには***物理的一貫性がない***可能性があります。

次に、時刻 ***t0*** から時刻 ***t1*** までの間にそのソース PDB に蓄積された REDO をコピーし、これをターゲットに転送します。この REDO をターゲット PDB に適用します。この段階で、ターゲット PDB は時刻 ***t1*** 時点のソース PDB の正確な物理コピーとなります。ただしこれには、コミットされたトランザクションとコミットされていないトランザクションの両方が含まれるため、***トランザクションとしての一貫性がない***可能性があると考える必要があります。

ここで必要になるのが、コミットされていないトランザクションのロールバックです。そのため、コミットされていないすべてのトランザクションに UNDO を適用します。これがクローン操作の最後の段階です。結果として得られたクローンは、時刻 ***t1*** 時点のソース PDB の、トランザクションとして一貫性のあるコピーとなっています。つまり、時刻 ***t1*** 時点のソース PDB に存在するコミットされたトランザクションはすべてクローンに存在し、コミットされていないトランザクションはすべてロールバックされていることになります。ホット・クローンを可能にしている重要な機能はローカル UNDO ということになるでしょう。この Oracle Multitenant の新機能については、後ほど

改めて説明します。ホット・クローンを実行するには、アーカイブ・ログも有効にする必要があります。

この一連の処理は、次に示すおなじみのクローン操作で***アトミック***に実行されます。

```
create pluggable database target from source;
```

ホット・クローンとコールド・クローンを区別する必要があるのは、12.1 の機能をよく知っているユーザーのみです。12.2 では、単にクローンと呼びます（そして、クローン操作はすべてホット・クローンです）。

リフレッシュ可能PDB

リフレッシュ可能 PDB はホット・クローンを土台にした機能です。ここでの代表的なユースケースは、本番 PDB のコピーを必要とする開発とテストです。本番データベースはおうおうにして大規模です。データベースが大きいほどコピーの取得に時間がかかります。数十テラバイト規模の本番データベースをコピーするには数日程度かかるでしょう。このような処理を頻繁に行うのは望ましくないため、この種の本番データベースの開発用コピーは時間の経過とともに古くなりがちです。データベースは、古くなればなるほど、デバッグ用などの開発用途での有意性が低下します。この問題はリフレッシュ可能 PDB で解消されます。

まず、ソース・データベースの完全クローンを取得します。これには、数時間か数日かわかりませんが、とにかく必要なだけ時間をかけます。ただし、現在はホット・クローニング機能があり、ソース・データベースを停止する必要がないため、このクローン取得の処理にどれほど時間がかかるかはほとんど問題になりません。

最初のクローンが完了したら、データベースは使用できる状態になります。リフレッシュ可能 PDB は、“ゴールデン・マスター”データベースとしてのユースケースを想定したものです。つまりこれは、個々の開発者がクローンを取得する際の元となるデータベースです。このときよく使われるのは、もっとも効率的なスナップショット・クローンで、非常に大規模なデータベースでも数秒または数分で作成できます。

スナップショット・クローンについては、オラクルから発行されている他のホワイト・ペーパーを参照してください。このリフレッシュ可能 PDB は“ゴールデン・マスター”データベースであるため、リフレッシュが終わってから次のリフレッシュまでの間、読取り専用でオープンしたままにしておく必要があります。リフレッシュ操作を実行している間は PDB をクローズしておく必要があります。

古くなったクローンはリフレッシュできますが、そのために、最後にリフレッシュした後に蓄積されたすべての REDO をクローンに適用します。ソース・データベースが巨大であっても、増分 REDO ははるかに小さいのが一般的です。したがって、最初のホット・クローンよりも極めて短時間のプロセスになります。そのため、本番からコピーされた最新のデータで本番クローンをリフレッシュし続けるのがはるかに容易になります。

リフレッシュ可能 PDB は、（特定のスケジュールに従って）自動的にリフレッシュされるように定義することも、（オンデマンドで）手動でリフレッシュするように定義することもできます。また、自動的にリフレッシュされるように定義した PDB をオンデマンドでリフレッシュすることも可能です。自動的にリフレッシュされるように定義した PDB を手動リフレッシュに移行すること、およびその逆も可能です。

自動的にリフレッシュされる PDB を作成する場合の構文は次のとおりです。

```
create pluggable database Golden_Master_PDB from Prod_PDB@DBLink
```

refresh mode every 360; -- (360 minutes)
作成したリフレッシュ可能 PDB は、次のようにして読取り専用でオープンする必要があります。

alter pluggable database Golden_Master_PDB read only;
手動でリフレッシュする PDB を作成する構文は次のとおりです。

create pluggable database Golden_Master_PDB from Prod_PDB@DBLink

refresh mode manual;
すでに述べたように、リフレッシュ可能 PDB は、最初のクローン操作が完了した時点で読取り専用で開くことができます。その後のリフレッシュ操作を実行するには PDB をクローズしておく必要がありますが、次の文で PDB をクローズできます。

alter pluggable database Golden_Master_PDB refresh;
注：12.1 の Oracle Multitenant のホワイト・ペーパー（2013 年発行）で説明されているように、作成後に初めて PDB が読取り/書込みでオープンされたときに、重要な手順が実行されます。したがって、総合するとこの最初の読取り/書込み操作は PDB の作成を完了するための重要な手順ととらえることができます。ただし、これはリフレッシュ可能 PDB には該当しませんので注意してください。というよりも、その後のリフレッシュができなくなりますので、リフレッシュ可能 PDB は読取り/書込みでオープンしないことが重要です。

PDB再配置

プラガブル・データベースは完全に機能する自己完結型の Oracle データベースです。プラガブル・データベースは移植可能なデータベースであるため、"プラガブル"と呼ばれます。この移植性が実現したのは 12.1 で、その手法は、ある CDB から PDB を切断して別の CDB に接続するというものです。これは非常に強力な機能です。PDB がクラウドに最適なのはそのためです。データセンターにあるサーバーに含まれる CDB から PDB を切断して Oracle Database Cloud Service に接続する、あるいは同じデータセンターにある別のサーバーに接続するだけで移植が完了します。切断/接続はロードバランシングを実行するための簡単な方法でもあり、その場合は、負荷の高いサーバーから空き容量が多い別のサーバーに PDB を移動する手法をとるのが一般的です。

ただし、切断/接続を使用して CDB 間で PDB を移動する手法には、切断してから接続するまで停止時間が発生します。ストレージを共有するサーバー間の移動はロードバランシングを目的として PDB を移動する場合にクラウド環境でよく見られるケースですが、この場合の移動操作はメタデータの操作に過ぎず、データファイル自体を物理的に移動する必要がないため、この停止時間はおそらく非常に短いでしょう。ところが、データセンター間や"地上からクラウド"（またはその逆）のデータベース移動には物理の法則が適用されます。つまり、すべてのデータを物理的に移動する必要があります。大規模なデータベースの場合は、使用するネットワークの容量によってはこのプロセスにかなりの時間がかかる場合があり、それに伴う停止時間が問題になる恐れがあります。

品質保証契約（SLA）では一般的に、2 つの対立する側面（パフォーマンスと可用性）が扱われます。データベースがパフォーマンス関連の SLA に違反する危険がある場合は、使用可能な処理能力が多いサーバーにデータベースを移動するのが一般的な解決方法です。しかしながら、これを実行するにはアプリケーションの停止時間が伴うのが常であり、可用性を規定した SLA の条項に違反する可能性が高くなります。

PDB 再配置は、CDB 間で PDB を移動する新しい手法です。これが 12.2 に導入されたことで、Oracle Database Cloud Service では、顧客との間で締結した SLA に含まれるパフォーマンス関連条項と可用性関連条項の両方を満たすことができます。Oracle Database Cloud Service は、オラクルのユーザーが使用しているものとまったく同じデータベース・テクノロジーを基盤としているため、この機能は一般にユーザーも使用できます。PDB 再配置の目標は停止時間を完全になくすことです。

ここからは PDB 再配置についてさらに詳しく説明します。詳細について知る必要がない場合は、次の項まで読み飛ばしていただいて構いません。

技術的な話をすると、PDB 再配置はリフレッシュ可能 PDB を"土台"にしており、すでに説明したように、リフレッシュ可能 PDB はホット・クローンを土台にしています。独立したリスナーが関係している場合は、透過的接続転送も伴う可能性があります。

運用の観点から言うと、PDB 再配置に必要な手順は次の 2 つです。

1. 新しいサーバーに PDB を再配置する
2. 新しい場所で PDB をオープンする

手順 1 には、元の場所から目的とする新しい場所に PDB をクローニングする操作が含まれます。言い換えると、この手順が完了した瞬間には、トランザクションとして一貫性があるこの PDB のコピーが 2 つ（ソース・サーバーに 1 つ、ターゲット・サーバーに 1 つ）存在することになります。操作の間、元の場所にあるデータベース上の処理は中断されることなく続行します。アプリケーション・ユーザーや、このデータベースに接続しているアプリケーションには、再配置が進行していることは認識されません。

技術的観点からすると、これは前述したリフレッシュ可能 PDB の作成と本質的に同じです。既存のすべてのアプリケーション接続と、この手順の間に作成される新しい接続は、引き続き元の場所にある PDB へと向かいます。すでに説明したように、手順 1 が時刻 t0 に開始されて時刻 t1 に完了した場合、時刻 t1 の時点では 2 つの PDB の内容にトランザクション上の一貫性があります。処理が続行しているため、t1 以降、接続が新しい場所に切り替わるまでの間に、元の場所にある PDB のデータが変更される可能性があることを考慮することが重要です。

手順 2 で再配置が完了します。操作の最後に PDB への接続が元の場所から新しい場所へと切り替わりますが、この手順もアプリケーション・ユーザーにはほとんど認識されません。この手順で重要なのは、ソースの場所にある PDB でコミットされたすべてのトランザクションをターゲットの場所に確実に保存することです。

すでに説明したように、厳密には、ソースの場所にある PDB に対して時刻 t1 以降、新しい場所で処理を再開するまでの間にコミットされたすべてのトランザクションを対象にする必要があります。おおまかに言うと、ソースの場所にある PDB で先にすべてのトランザクション処理を停止する必要があります（この時点を時刻 t2 とします）。ソースの場所にある PDB に蓄積されたコミット済みのすべてのトランザクションをターゲットの場所に適用し直さなければ、ターゲットの場所でトランザクションの処理を再開することはできません。これらの要件を踏まえると、このオープンの段階には次のサブ手順があります。

2a：読取り専用でオープンする。新しい場所にある PDB には時刻 t1 時点でのトランザクション上の一貫性があることはわかっているため、これを読取り専用でオープンしても問題はありません。そのため、問合せ用としてはすぐに使用できますが、DML を試行する接続は“混乱”するでしょう。この段階では、コミットされたトランザクションの一部がターゲットの場所で表示されない可能性がありますが、トランザクションが失われることはありません（これについては後述します）。ターゲットの場所にある PDB に対する問合せ（この段階で実行できるのは問合せのみですが）の動作は、ソースの場所にある PDB で時刻 t1 時点に実行した場合に実行されたであろう動作とまったく同じです。これは、読取り一貫性の特殊なケースで見られることがあります。

2b：接続を転送する。ソースの場所からすべての接続を“除去”し、ターゲットの場所で再度確立する必要があります。新しい接続要求はターゲットの場所に転送され、そこにある PDB に接続されます。適切に設計された最新のアプリケーションは、基盤となるデータベースと接続プール経由で通信します。この方法を採ると、（コンピューティング・リソースの面では比較的コストがかさみますが）多数のアプリケーション・ユーザーを少数のデータベース接続でサポートできます。トランザクションが完了して接続プールに接続が解放されると、接続はクローズされます。実行が長引いているトランザクションや適切に記述されていないアプリケーションの場合は、適切なタイミングで自然に接続が解放されないことがあります。そのような接続は、少し時間を置いてから自動的に終了し、接続先で再度確立されます。接続先でログイン“ストーム”が発生しないようにするために、このプロセスは 1 つずつ実行されます。

つまり、一部のトランザクションは PDB 再配置の最中に中断される可能性があります。なおこれは、PDB 全体としての可用性はほぼ継続するという主張とは矛盾しません。そのため管理者は、長時間実行されているトランザクションの処理時間枠に配慮し、PDB 再配置への影響を最小限にとどめることが重要です。

接続の転送の詳細は、PDB のソースの場所とターゲットの場所でのリスナーの構成によって異なります。考えられるのは次の 3 つです。

2.b.i　共有リスナー。同じサーバーにある CDB 間の再配置を伴う場合にこのような状況になることがよくあります。たとえば、異なるオプション・セットがインストールされている CDB に PDB が再配置される場合が該当します。この場合は、新しい場所のリスナーに PDB が再登録されます。

2.b.ii　相互登録されたリスナー。ロードバランシングなどを目的として 1 つのデータセンター内にあるサーバー間で再配置を行う場合にこのような状況になるのが一般的です。この場合は、新しい場所で両方のリスナーに PDB が再登録されます。

2.b.iii　相互登録がない独立したリスナー。データセンター間で PDB を再配置する場合にこのような状況になることがよくあります。企業が所有するデータセンターから Oracle Database Cloud Service に再配置する場合もこの状況になるでしょう。相互登録がないと、ターゲットの場所のリスナーとソース CDB には互いに関する"知識"がないため、この状況は複雑です。このようなケースに備え、"高可用性"と"最大可用性"という 2 つの可用性レベルが用意されています。新しい場所にある PDB にはほぼ即座に直接接続できるようになるため、高可用性という名前が適しています。高可用性 PDB 再配置の構文は次のとおりです。

```
create pluggable database My_PDB from My_PDB@DBLink relocate availability high;
```

この文はターゲット CDB から発行する必要があります（availability high がデフォルトです）。

最大可用性にすると処理が 1 つ増え、元の場所にある PDB への接続が新しい場所に自動転送されます。最大可用性 PDB 再配置の構文は次のとおりです。

```
create pluggable database My_PDB from My_PDB@DBLink relocate availability max;
```

この文はターゲット CDB から発行する必要があります

ここで重要なのは、再配置された PDB と同じ名前で別の PDB を元の場所に作成できないようにすることです。これが可能だとすると、非常に紛らわしい事態になりかねません。このような事態を回避するために、最大可用性 PDB 再配置を実行する場合はソース CDB に"ツームストーン"を作成します。これはソース CDB のメタデータ・エントリで、ソース CDB のルート・コンテナから v$containers を問い合せると表示できます。"ツームストーン"PDB は、"relocated"というステータスで表示されます。

2c：ターゲットの場所で増分トランザクションを適用する。時刻 ***t1*** と時刻 ***t2*** の間に蓄積されたそのソース PDB の REDO データがすべてターゲットの場所にコピーされ、そこで適用されます。この段階で、ターゲット PDB は時刻 ***t2*** 時点のソース PDB の正確なコピーとなります。ただし、これにはコミットされたトランザクションとコミットされていないトランザクションの両方が含まれています。ここで必要なのは、コミットされていないトランザクションのロールバックです。そのため、コミットされていないすべてのトランザクションに UNDO を適用します。元の場所でトランザクションが発生しないようにしてあるため、ターゲットは最新の状態になっていると見なすことができます。

2d：ターゲットの場所にある PDB を読取り/書込みでオープンする。このとき、データファイルは元の場所から削除されます。

PDB 再配置はこれで完了です。すべての処理は何事もなかったかのように PDB の新しい場所で再開されますが、ここまで説明してきたとおり、さまざまな処理が実行されました。しかしながら、オペレーターが実行しなければならなかったのは、2〜3 の SQL 文を発行することだけでした。

» アプリケーションを停止する必要はありませんでした。

» アプリケーションを変更する必要はありませんでした。

» 接続文字列を変更する必要はありませんでした。

それでも、PDB を問題なく再配置できました。

なお、相互登録がない独立したリスナー同士で PDB を再配置するケースでは、元の場所に"ツームストーン"PDB が存在するため、PDB を元のコンテナに戻す再配置は（ツームストーンがその名前空間を"専有"しているため）、少なくとも元の名前では即座に実行できないことになります。ただし、この状態を永続化することは想定されていません。前述したように、最大可用性 PDB 再配置では接続が自動的に転送されます。そのため、クライアントは接続文字列を変更しなくてもアプリケーション・データベースに接続し続けることができます。ただし、これには余計なネットワーク"ホップ"が伴います。都合のよいタイミングでアプリケーション接続文字列を変更し、新しい場所にあるデータベースに直接接続できるようにしてください。そうなった時点で、ツームストーン PDB を元の場所から削除できます。その後は障害物がなくなるため、そうすることが望ましい場合は、PDB を元のコンテナに戻す再配置を実行できます。

PDB再配置の制限事項

通常、PDB 再配置も切断/接続と同じ接続互換性に関わる制約を受けます。このトピックについて詳しくは、12.1 の Oracle Multitenant のホワイト・ペーパー（2013 年発行）を参照してください。基本的に、PDB の接続互換性をチェックするには、PDB の xml マニフェスト・ファイルをターゲット CDB と比較します。ただし（PDB 再配置はオンライン操作であるため）、PDB を再配置する場合は、切断操作を実行して xml マニフェスト・ファイルを作成しない必要があります。代わりに、元の場所にある PDB に接続して次のコマンドを発行し、マニフェスト・ファイルを生成する必要があります。

```
execute DBMS_PDB.Describe() into PDB_manifest.xml;
```

切断/接続の機能強化

暗号化鍵の移行

現代のクラウド・デプロイメントには、Oracle Maximum Security Architecture に示されているような包括的なセキュリティ戦略を含める必要があります（Oracle Maximum Security Architecture については、オラクルが発行している別の資料を参照してください）。オラクルのセキュリティ理念は、予防、検出、管理を主要素とする包括的な戦略に基づく"多層防御"です。全体を保護する技術的要素は 1 つもありませんが、基本的な機能は透過的データ暗号化（TDE）に由来しています。Oracle Database Cloud Service では TDE はオプションではありません。Oracle Database Cloud Service のデータベースはすべて TDE で保護されます。オラクルのお客様の多くは"ハイブリッド・クラウド"という理念を採り入れ、"自社のクラウドもオラクルのクラウドのように構築する"ことを希望し、TDE の導入も希望します。したがって、切断/接続の操作中に PDB ごとの暗号化鍵を CDB 間で移行するための手法が必要です。

12.1 では、PDB の切断/接続操作とは別に、鍵のエクスポート/インポートを実行しました。この手順については、12.1 の Oracle Multitenant のホワイト・ペーパー（2013 年発行）を参照してください。12.2 では、暗号化鍵の移行が切断/接続操作と統合されたため、操作が簡素化されました。厳密に言うと、暗号化鍵は、共有秘密鍵で暗号化されたうえで xml マニフェスト・ファイルを介して転送されます。そのため、次のように、切断文および接続文にいくつかの句を追加する必要があります。

切断：ソース CDB のルート・コンテナで SysDBA として実行します。

```
alter pluggable database MyPDB unplug into MyPDB_manifest.xml
encrypt using <共有秘密鍵>;
```

接続：ターゲット CDB のルート・コンテナで SysDBA として実行します。

```
create pluggable database MyPDB using MyPDB_manifest.xml
decrypt using <共有秘密鍵>;
```

自動ログイン・ウォレットを使ったセルフサービス・プロビジョニング

Database as a Service（DBaaS）のおもな特徴はセルフサービス・プロビジョニングです。すでに説明したように、Oracle Database 12c のコンテナ・データベース・アーキテクチャは、DBaaS のデプロイ・モデルとして圧倒的に効率的です。PDB はデータベース・クラウドに接続する媒体であるだけでなく、データベース・クラウドを実現するための要素でもあります。***Pluggable*** Database as a Service（***P***DBaaS）と呼ぶことが多いのはそのためです。PDBaaS は無人操作に対応した設計になっていないため、当然のことながらパスワードベースのウォレットには適していません。そのため、"自動ログイン・ウォレット"を介して TDE を操作することもできます。このような操作は 12.1 でもサポートされましたが、12.2 では簡略化されています。

PDBアーカイブ

切断/接続は、CDB 間で PDB を高速かつ効率的に移動できる平易な手段です。切断操作の基本構文は次のとおりです。

```
alter pluggable database MyPDB unplug into PDB_manifest.xml;
```

この PDB_manifest.xml は、PDB の"xml マニフェスト・ファイル"と見なされます。ストレージを共有しない CDB 間で PDB を移動する場合は、この xml マニフェスト・ファイルを、PDB のすべてのデータファイルとともにターゲット・サーバーに物理的にコピーする必要があります。

この構文は 12.2 でも引き続きサポートされています。これに加え、マニフェスト・ファイルとすべてのデータファイルを結合して PDB アーカイブと呼ばれる 1 つのファイルにする新機能が導入されました。この PDB アーカイブの拡張子は常に.pdb です。PDB アーカイブは、おなじみの切断文に（拡張子.pdb を付けた）アーカイブ・ファイルを指定するだけで作成できます。したがって、切断文は次のようになります。

```
alter pluggable database MyPDB unplug into MyPDB_archive.pdb;
```

同様に、共通の SysDBA としてターゲット CDB のルート・コンテナに接続する場合は、次のコマンドを使用して、アーカイブ・ファイルから PDB を接続します。

```
create pluggable database MyPDB using MyPDB_archive.pdb;
```

PDB再配置の使用が適した状況および切断/接続の使用が適した状況

切断/接続は、12.2 でも引き続き完全にサポートされ、機能強化も図られている強力な操作です。ただし前述したとおり、12.2 で導入された PDB 再配置機能を使用すると、別の手段で同じ結果が得られます。PDB 再配置の大きなメリットはオンライン操作だということです。つまり、PDB を移行するときにアプリケーションを停止する必要がありません。切断/接続ではそうは行きません。したがって、CDB 間の PDB の移行は、ほとんどが PDB 再配置を使用して実行されることになるでしょう。ただし、切断/接続を使用した方が良い状況もいくつかあります。たとえば、次のような場合です。

» Oracle Database のバージョンまたはパッチ・レベルが異なる CDB 間で PDB を移動する場合

» インストールされているオプション・セットが異なる CDB 間で PDB を移動する場合

クローニングの機能強化

良い機会ですので、初回リリースである 12.1 以降に Oracle Multitenant のクローニング機能に対して図られたさまざまな機能強化を、このホワイト・ペーパーの発行に合わせて紹介します。

サブセット・クローンとメタデータのみのクローン

場合によっては、PDB の表領域のサブセットを含めたクローンを作成するのが望ましいこともあります。これについては別の資料で詳しく説明していますが、簡単に言うと、サブセット・クローンは、***create pluggable database*** 文の ***user_tablespaces()***句を使用することで作成できます。したがって、次のような文になります

```
create pluggable database NewPDB from OldPDB user_tablespaces(TS1, TS2);
```

技術的観点から言うと、この文を実行した結果は、除外された表領域に関連するデータファイルをオフラインにしたのと同じです。

スキーマ統合から Oracle Multitenant にアップグレードすると、この機能がとても便利であることがわかります。というのも、一般的なスキーマ統合モデルではスキーマごとに専用の表領域を保持するからです。たとえば、専用の表領域（Schema1_TS、Schema2_TS、Schema3_TS）を使用して 3 つのスキーマ（Schema1、Schema2、Schema3）が統合されたデータベースからアップグレードする場合は、次の一連の操作（12.2 にアップグレード済みの非 CDB を起点とすると仮定）を実行すると、Oracle Multitenant にアップグレードされます。

```
--（非 CDB で実行します）マニフェスト・ファイルを作成します。

execute DBMS_PDB.describe(non$cdb@DBLink) into Non_CDB.xml;

--（CDB のルートで共通の SysDBA として実行します） -- マルチスキーマ DB を PDB として導入します。

create pluggable database MultiSchema using Non_CDB.xml;

alter pluggable database MultiSchema open;

--（PDB のマルチスキーマに接続して実行します）PDB のメタデータを変換します。

@NonCDB_to_PDB.sql

--（CDB のルートで共通の SysDBA として実行します）スキーマから PDB を作成します。

Create pluggable database PDB1 from MultiSchema
user_tablespaces(Schema1_TS);
create pluggable database PDB2 from MultiSchema
user_tablespaces(Schema2_TS); create pluggable database PDB3 from
MultiSchema user_tablespaces(Schema3_TS);
-- 役割が終わったため、マルチスキーマ PDB をクローズして削除します。

alter pluggable database MultiSchema close;
drop pluggable database MultiSchema including datafiles;
```

以上の手順で、スキーマごとに PDB が作成されます。そのまま、順番に各 PDB に接続し、使用されなくなったスキーマのメタデータを削除します。たとえば、PDB1 に接続して次の文を発行します。

```
drop user Schema2 cascade;
drop user Schema3 cascade;
```

サブセット・クローンの極端な例はメタデータのみのクローンです。メタデータのみのクローンを作成する構文は次のとおりです。

```
create pluggable database DevPDB from ProdPDB@DBLink no data;
```

この機能は、デバッグを目的として本番データベースのクローンを"急場しのぎ"的に作成するときに使用できます。

ナップショット・クローン

PDB のクローンのバリエーションで特に強力なのはスナップショット・クローンです。スナップショット・クローンは、ソース PDB の完全なデータセット・クローンとして動作しますが、ストレージの消費量はほとんど増えません。スナップショット・クローンの作成時に物理的に書き込む必要があるデータはほとんどないため、作成処理は極めて高速です。基本的な構文は、標準のクローニング構文に ***snapshot copy*** 句を追加したものです。たとえば、次のようになります。

```
create pluggable database Snapshot_PDB from Source_PDB snapshot copy;
```

スナップショット・クローンの概要は 12.1 の Oracle Multitenant のホワイト・ペーパー（2013 年発行）に書かれていますので、ここではあえて繰り返しません。

ですが、このホワイト・ペーパーが発行されるのはちょうど良い機会ですので、スナップショット・クローンに対応するストレージ・テクノロジーの最新リストを紹介します。対応するストレージ・テクノロジーは、それぞれに異なる特徴を持つ 3 つのカテゴリに分類されます。

ストレージベースのスナップショット

ストレージ・テクノロジーの中には、ストレージ効率の高いファイル・コピーをネイティブでサポートするものがあります。Oracle Multitenant は、不要な独自レイヤーを追加導入して制約を作ることなく、そうした基盤機能を最大限に活用できます。これを、"ストレージベースのスナップショット"と呼びます。これがサポートされるのは次のプラットフォームです。

» Oracle ZFS Storage Appliance

» Oracle ASM Cluster File System（Oracle ACFS）

» NetApp™

» EMC™

スナップショット・クローンはソース PDB から作成しますが、ソース PDB を読取り/書込み可能のままにできる点がストレージベースのスナップショットの大きな特徴です。

ファイル・システム・ベースのスナップショット（スパース・ファイルによる有効化）

専用のストレージ・テクノロジーではなく単純なファイル・システムにデータベースが展開されることもあります。最新のファイル・システムはスパース・ファイルに対応しています。そうしたファイル・システムでも、CDB 初期化パラメータ CloneDB を TRUE に設定すればスナップショット・クローンの作成が可能です。ただし、そうすると、ソース PDB が読取り専用のままになり、スナップショット・クローンが 1 つでも存在する間は変更できないという制約があります。とはいえ、スナップショット・クローンの代表的なユースケースでは読取り専用のクローン・ソースを使用する傾向があるため、実際にはこの制約が特に重荷になることはありません。

» リフレッシュ可能 PDB を開発/テスト環境の"ゴールデン・マスター"データベースとして使用する例は、すでに本書で説明しました。

» ほかには、破壊的 what-if 分析のための"サンドボックス"環境をプロビジョニングするときにスナップショット・クローンを使用する例がよく見られます。

Oracle ASMを搭載したExadata Storage

Oracle Automatic Storage Management（Oracle ASM）を搭載した Exadata Storage でも PDB のスナップショット・クローンの作成がサポートされます。この場合も、ソース PDB が読取り専用のままになり、スナップショット・クローンが 1 つでも存在する間は変更できないという制約があります。

この場合も、ソース PDB が読取り専用のままになり、スナップショット・クローンが 1 つでも存在する間は変更できないという制約があります。

ローカルUNDO

12.1 では、CDB 全体のグローバル UNDO（共有 UNDO）があります。共有 UNDO を使用する場合は、（コールド）クローニングや切断などの操作を実行する前に、コミットされていないトランザクションがソース PDB にないかチェックする必要があります。これは、クローニング操作や接続操作の後で PDB にトランザクション一貫性に関わる問題が発生しないようにするためです（コミットされていないトランザクションが存在する場合は、「***ORA-65126 – プラガブル・データベースは正常にクローズしませんでした。アクティブなトランザクションがあるためリカバリが必要です***」というエラーが発行されます。このような場合は、PDB を読取り/書込みでオープンし、SMON プロセスによるクリーン・アップが完了するまで待機する必要があります）。

12.2 には PDB 単位の UNDO（ローカル UNDO）が導入されています。ローカル UNDO を使用するとこうした問題を回避できるため、このような重要な操作の結果を非常に予測しやすくなります。また、12.2 の Oracle Multitenant の次のような主要な新機能の多くが実行可能になります。

» ホット・クローン

» リフレッシュ可能 PDB

» PDB 再配置

» フラッシュバック PDB

12.2 でも引き続き共有 UNDO がサポートされますが、おもにアップグレードのための暫定的なサポートにすぎません。PDB 単位で UNDO を所持すると、技術面および管理面でわずかなオーバーヘッドがあると考えられますが、ローカル UNDO を所持することのメリット（このホワイト・ペーパーや他の資料で説明しているすべての新機能が使用できるようになること）の方がはるかに優ります。ルールは単純なのが一番ということが少なくありません。共有 UNDO が強く推奨される状況は特にないため、"どのような場合もできるだけすみやかにローカル UNDO に移行する"という単純なルールに従うのが一番です。Oracle Database Configuration Assistant（Oracle DBCA）で CDB を新規作成すると、デフォルトでローカル UNDO が有効化されます。

UNDO のモード（ローカルか共有）は、全部か無しかを CDB 全体で指定するプロパティです。部分的な指定はできません。つまり、すべての PDB にローカル UNDO を使用するか、CDB 全体で共有 UNDO を使用する必要があります。

ローカル UNDO と共有 UNDO は相互に切り替えることができます。切り替える場合は、CDB を再起動する必要があります。

共有 UNDO を使用する CDB からローカル UNDO を使用する CDB に PDB を移動すると、必要なローカル UNDO 表領域が自動的に作成されます。ローカル UNDO を使用する CDB から共有 UNDO を使用する CDB に PDB を移動すると、PDB を読取り/書込みで初めてオープンしたときに、コミットされていないトランザクションをロールバックするために少しの間ローカル UNDO 表領域が使用されますが、その後は不要になるので削除しても構いません。CDB を共有 UNDO からローカル UNDO（またはその逆）に移行する場合も同じです。

Oracle Real Application Clusters クラスタで PDB をオープンする場合は、ノードごとにローカル UNDO 表領域が必要です。2 ノードの Oracle RAC から 4 ノードの Oracle RAC に PDB を移動（し、すべてのノードでオープン）する場合は、追加で必要となる UNDO 表領域が自動的に作成されます。PDB を元に戻した場合は 2 つの UNDO 表領域が余分になるため、削除しても構いません。

注：

1. ローカルUNDOを有効にするためにCDBレベルのデータベース・パラメータcompatibleを12.2に設定する必要はありません。
2. UNDOモード（共有またはローカル）は接続互換性に関わる問題とは見なされていません。

スケーラビリティの実現 – 統合の障害物を取り除く

なぜ Oracle Multitenant を使用すると真のスケールメリットが得られ、2 つの側面が両立するのか、その理由を詳しく説明してきました。統合すればするほどコストが削減されます。そこで非常に重要になってくるのが、統合の障害となるものを取り除くことです。

12.1 の Oracle Multitenant アーキテクチャでは使用できない Oracle Database の機能がいくつかあったのは確かです。それら全体が大規模な統合の障害だったと見なすことができるでしょう。12.2 ではそうした障害が取り除かれています。この項では、12.2 の Oracle Multitenant で使用できるようになった Oracle Database の重要な機能の一部を紹介します。

フラッシュバックPDB

フラッシュバック PDB が完全にサポートされるようになりました。

PDB単位のキャラクタ・セット

12.2 の Oracle Multitenant では、キャラクタ・セットが異なる PDB を統合できます。

CDB を AL32UTF8 で構成することが要件ではありますが、キャラクタ・セットが異なる PDB を 1 つの CDB に統合することが可能になりました。文字列は PDB ごとに適切に処理されます。さらに、コンテナをまたぐ集計（通常は containers() SQL コンストラクトを使用）を実行する場合も、異なる PDB の文字列が適切に処理されます。つまり、Oracle Multitenant で統合する前に（DMU を使用して）非 CDB を AL32UTF8 に変換する必要はもはやありません。

たとえば、CDB でキャラクタ・セット AL32UTF8 を使用しているとすると、この CDB には次の PDB を統合できます。

» 漢字キャラクタ・セットを使用する PDB_Japan

» ISO8859_p1 キャラクタ・セットを使用する PDB_EMEA

» 任意の数の他の PDB

Oracle CloudおよびOracle Exadata上の1つのCDBに4k個のPDB

1 つの CDB に 252 個の PDB というのは 12.1 ではかなりの数でしたが、Oracle Cloud および Oracle Exadata 上の 12.2 ではこれが 16 倍の 4,096 個、すなわち 4k 個に増えました。Oracle Multitenant を使用した場合は統合すればするほどコストが削減される、というスローガンは冗談ではありません。オラクルは、大幅なコスト削減を実現するために、かなりの規模をサポートしたいと考えています。標準的な SaaS デプロイメントではこの種の規模が現実的ですが、これは Oracle Taleo Business Edition などいくつかのオラクル独自の SaaS アプリケーションが稼働する Oracle Multitenant を Oracle Cloud に実装した経験に基づくものです。その他のプラットフォームでの上限は、1 つの CDB に 252 個の PDB のままです。

PDBレベルの自動ワークロード・リポジトリ（AWR）データ

これを使用することで、特定の PDB 内のパフォーマンスをきめ細かく診断できます。これは、DBaaS 環境で特に重要です。というのも、この環境のユーザーは、その PDB 内のパフォーマンスに責任を持つ自立したプラガブル・データベース管理者（PDBA）がいると思っているはずだからです。

ヒート・マップ

ヒート・マップは Oracle Database 12c で導入された重要な機能です。12.2 の Oracle Multitenant は、ヒート・マップに完全に対応しています。つまり、自動データ最適化（ADO）を有効にすると使用できるストレージ効率化機能のすべてがコンテナ・データベース・アーキテクチャでも使用できます。

独立性

独立性を犠牲にした統合であれば、その統合によるスケールメリットは限定的です。12.2 には、独立性を支援する非常に高度な機能がいくつか導入されているため、PDB 同士を厳密に分離することができます。これにより、いわゆる“noisy neighbor”（うるさい隣人）と呼ばれる問題を回避できます。また、これは構成可能な機能であるため、ユースケースに合わせて独立性のレベルを適切に設定できる点が優れています。PDB の独立性はデータベース・クラウドなどの統合度の高い環境では特に重要なトピックですが、一般に、このトピックを検討する場合は、共有することにより発生しうるあらゆるリスクを検討する必要があります。リスクは次のようにいくつかのカテゴリに分類されます。

» 共有コンピューティング・リソースの競合

» システム・アクセス

» ファイル・システム・アクセス

» ネットワーク・アクセス

» 共通ユーザーまたは共通オブジェクト・アクセス

» 管理機能

12.2 では、既存の強力な分離機能をベースにして包括的なモデルを構築しましたが、簡単な構成だけで特定のユースケースに適した独立性のレベルを正確に実現できます。

これらの機能についてこれから詳しく説明しますが、その前に、"万能型"アプローチで独立性に対処するのはデータベース・クラウドでは適切でない理由と、本当に必要なのは構成可能な分離機能である理由を理解しておく必要があります。このトピックについて考える場合は、住宅のセキュリティという現実世界でよく聞くたとえを検討すると理解しやすくなります。最初は、セキュリティは厳しいほど良いと考えるかもしれませんが、実際にはセキュリティを優先すると利便性が損なわれます。"厳重なセキュリティ"という言葉を聞くと、住宅よりも刑務所が連想されます。窓に格子を付け、天辺に有刺鉄線を張り巡らせた高い塀で周りを囲み、鋼鉄の扉に錠前を 3 つ付け、武装した見張り番を扉の前に配置すれば、住宅のセキュリティは強化されるでしょう。しかし、これでは決して暮らしやすいとは言えません。逆に、子供たちが出入りしやすいように、扉にも窓にもまったく鍵をかけなかったら、所有物がなくなってしまう可能性があります。人は適切なところでバランスを取ろうとしますが、そのバランスは状況によって異なります。近所に知り合いが多い郊外よりも人口密度の高い都市環境にいる方が警戒心は強くなるはずです。小さな町ではわざわざ鍵をかけたりしないこともあります。誰が何をしているのか、全員が知っているからです。興味深い例として、ビジネス・ホテルのセキュリティについて検討してみましょう。通常、セキュリティは 24 時間体制で、すべての共有スペースに防犯カメラが設置され、警備員が配備され、客室に入るには精巧な鍵が必要です。人は、別の客室の利用者が自分の部屋に入れる可能性があることには非常に不安を覚えますが、ホテルの従業員が日中自分の知らない間に文字通り何度でも客室に入ることができ（ベッド・メーキングとバスルームの清掃ができている場合は入らないでしょうが）、たいていは夜間に自分がいるときでも客室に入れることはほとんど心配しようとしません。これは興味深いことではありませんか。どういうわけかこのような状況では、ホテルの経営者にセキュリティを任せることにまったく問題を感じません。

似たようなことが、別のユースケースのデータベース・クラウドについても言えます。

パブリック・クラウド上の Database as a Service（DBaaS）では、"隣接する"テナントが競合他社である可能性は十分にあります。各テナントは、自社の PDB 内を強い権限で管理できる必要があります。しかし、隣接する PDB のすべての PDBA からその PDB を完全に分離することも必要です。この 2 つの要件を両立させる必要があるため、これは特に難しいユースケースです。これに適した住居のたとえとして、マンションの所有権の話があります。住民は、自分が所有する空間は完全に自分で管理したいと思います。玄関より内側にあるものはすべてその人が責任を持ちます。

プライベート・クラウド上の DBaaS は開発チームにとって非常に生産性の高い構成になっています。それぞれの開発者は、ある開発者のテストが別の開発者のテストを妨害しない程度に他の開発者から分離されている必要がありますが、通常は共同作業に適した環境で、そこでは誰もが他の開発者の環境を尊重するものと考えられています。これに適したたとえとして、大きな家を友人同士でシェアする例があります。全員が同じ正面玄関の鍵を持っていて、普段は各自の寝室のドアに鍵をかけません。共有スペースと共有設備はいくつかありますが、各自の寝室内のプライバシーはほどほどに守られています。

Software as a Service（SaaS）はホテルでの滞在にたとえることができるでしょう。客室料金を支払っているため維持管理とセキュリティはすべてホテルの経営者に任せます。また、客室係は実質的にいつでも客室に入れるでしょうが、客室内の私物にはそれなりの敬意を払ってもらいます（この場合は通常、客室内の金庫を使用して貴重品を客室係から守ります）。ホテル内にほかの宿泊客がいることは誰もが知っていますが、ほかの客室の利用者は自分の部屋に入ることができないと、十分な根拠を持って見込んでいます。

オラクルは、こうした考慮事項を念頭に置いて、Oracle Multitenant に対応した構成可能な分離モデルを設計しました。この後の項では、このモデルの主要な機能について説明します。

リソース管理

ここまでの説明でおわかりのとおり、リソースを共有することで効率を大幅に向上させることができますが、ほかのものをすべて犠牲にして 1 つのワークロード（この場合は PDB）にシステムのリソースを独占させる余裕はありません。だからこそ、強力ながらも使いやすいリソース管理機能を持つことが極めて重要です。

12.1 では、リソース・マネージャのポリシーを定義して CPU、I/O（ただし、Exadata と SuperCluster のみ）、セッション、パラレル実行（PX）サーバーを制御できます。12.2 では、この機能の大幅な強化が図られています。12.2 を使用すると、非エンジニアド・システムにメモリ管理機能と I/O 管理機能が搭載されます。

メモリ管理

12.1 で要望が多かったのがこの機能ですが、多くのユーザーがこの機能なしで非常にうまく統合を済ませている点は押さえておく必要があります。一見すると、この機能がないのは極めて重大な欠陥のように思えます。しかしよく考えてみると、結局ほとんどの事例ではこの機能がさほど重要でなかった理由がはっきりしてきます。理由はいくつかあります。まず、Oracle Multitenant が効率的だということの意味ですが、これは、n 個のスタンドアロン非 CDB でアプリケーションを実行していたとすると、このすべてのアプリケーションをサポートしていた SGA の合計よりも、n 個の PDB に共有される SGA に使用できるメモリの方が大幅に多くなることです。これについては複数の徹底した技術的調査で実証および定量化されていますが、おおまかには次の方法で大幅な効率化が図られます。

» バックグラウンド・プロセスが複製されることで消費されるプログラム空間を排除する。

» スタンドアロン・データベースに過剰にプロビジョニングされている SGA を共有する。

» 複数のテナントで同じ SQL 文が発行された場合は親カーソルを共有する（特に SaaS の場合）。

使用できる合計メモリ量が増えることに加え、極めて効率的なメモリ管理アルゴリズムが Oracle SGA に組み込まれていることも、統合がうまく行く理由であると考えられます。1 つの非 CDB の中でも SGA は複数のユーザー同士で共有されることを思い出してください。複数の PDB を統合するとメモリ量が増え、それを共有するユーザーも増える、と単純に考えることもできます。統合するワークロードが多ければ多いほど、すべてのワークロードの需要が同時にピークに達する可能性が低くなるというのが、統合に見込まれる基本的効果です。そのため、柔軟に割り当てることがきるリソースの正味需要が平準化されやすくなります。そして、その好例が SGA です。

ここまでの説明はこじつけではなく、あくまで事実です。なぜなら、12.2 にはメモリ管理機能が実際に導入されているからです。この機能は必要に応じて使用できるものですが、すでに説明したように、メモリ管理を使用しなくてもまったく問題なく動作する統合のユースケースは多数あります。メモリ管理を使用するのが適切と思われる重要なユースケースは、ミッション・クリティカルな（すなわち"プラチナ SLA"の）いくつかのアプリケーション・データベースを比較的低密度で 1 つのサーバーに統合するケースです。

今回、PDB 単位で設定できるようになったパラメータは次のとおりです。

パラメータ	説明
SGA_Target	PDBの最大SGAサイズ
SGA_Min_Size （12.2の新規パラメータ）	（バッファ・キャッシュおよび共有プール用として）PDBへの割当てが保証されるSGAサイズ。ヒント：すべてのPDBのSGA_Min_Sizeの合計がSGAの50%未満になるようにしてください。
DB_Cache_Size	PDBへの割当てが保証されるバッファ・キャッシュ・サイズ
DB_Shared_Pool_Size	PDBへの割当てが保証される共有プール・サイズ
PGA_Aggregate_Limit	PDBの最大PGAサイズ
PGA_Aggregate_Target	PDBが利用するPGAの目標サイズ

汎用ストレージでのI/O管理

12.1 では、Exadata Storage（エンジニアド・システムである Oracle Exadata と Oracle SuperCluster のみで使用可能）以外では I/O を管理できませんでした。ストレージはデータベースに最適化され、データベースに統合されているため、データベースはストレージのスループット能力を IOPS と MBPS で"把握"しています。この場合は、PDB 間の I/O を制御するリソース・マネージャのポリシーを、割合と上限だけで定義できます。

12.2 では、Exadata 以外のストレージ上にある PDB にレート制限をかけられる 2 つの PDB 単位のパラメータが新たに導入されました。それが、Max_IOPS（1 秒あたりの最大 I/O 操作数）と Max_MBPS（1 秒あたりの最大転送量（メガビット単位））です。両方とも動的に設定や変更ができます（CDB$Root または Exadata Storage では設定できません。設定しようとするとエラー・メッセージが返されます）。

CPU管理

12.1 では、CDB のリソース・プランの一環として PDB ごとの CPU 制限を定義しました。12.2 には、PDB レベルのパラメータとして CPU_Count が導入されています。クラウド・モデルでは支払われた料金に相当する分を割り当てるため、特徴が異なるサーバー間で PDB が移動されたとしても、先を見越して PDB にパフォーマンス制限をかけることがはるかに容易です。50%の CPU 制限とは、X4-2 サーバーでは 12 コア、X6-2 サーバーでは 22 コアを割り当てるという意味です。新しいモデルでは、特定の PDB に CPU_Count（たとえば 8）を設定することができ、PDB がどのサーバーでホストされてもこの制限が適用されます（下位互換性を維持するために、リソース・マネージャのポリシーで定義した割合制限も引き続きサポートされます。両方とも指定されている場合は、低い方の設定が適用されます）。

システム・アクセス

Oracle Database 内からオペレーティング・システム（OS）を操作（SQL Plus でホスト・コマンドを使用）すると、OS の Oracle ユーザーの権限で操作が実行されます。このアカウントには強い権限があるのが一般的で、クラウド環境では明らかに潜在的リスクであるため、このリスクを軽減する必要があります。12.2 には、PDB レベルのパラメータ（PDB の OS 資格証明）が新たに導入されています。必要な場合は、OS へのアクセスに使用される OS ユーザー（ほとんどの場合は Oracle ユーザーよりはるかに権限が弱いユーザー）をこれで識別することができます。

ファイル・アクセス

基盤となるファイル・システムにあるファイルやフォルダにアクセスする操作（たとえば、外部の表にアクセスする場合）にもリスクが潜在するため、なんらかの対処が必要です。12.2 には、***create pluggable database*** 文に 2 つの新しい句（Path_Prefix と Create_File_Dest）が導入されています。これらは PDB のファイルの配置場所とそこへのアクセスを制御する句です。ファイル・システム内のマウント・ポイントを指定することで、ファイル・システムのサブブランチに各 PDB のデータファイル・オブジェクトやディレクトリ・オブジェクトが隔離されるようにします。

ロックダウン・プロファイル

これは、ネットワーク・アクセス、共通ユーザーと共通オブジェクト、管理機能を非常にきめ細かく制御できるようにする 12.2 の Oracle Multitenant の重要な新機能です。このテーマは別のホワイト・ペーパーで取り上げているため、ここでは簡単な説明にとどめます。

基本的な理念は、“多数のものを一括管理する”という原理に沿っています。想定されている使い方は次のとおりです。コンテナ・データベース管理者（CDBA）とでも呼ぶべき一元的な管理者がいくつかのロックダウン・プロファイルを指定します。このロックダウン・プロファイルは、それぞれのユースケースに基づいて個々の PDB に適用できます。このモデルの優れている点は、プロファイルが異なる複数の PDB が統合された 1 つのコンテナ・データベースに対応できるところです。

ロックダウン・プロファイルは grant を補完するものであり、特定の権限を付与すると標準で有効化される能力を制限する役割を果たします。たとえば、alter system 権限を付与されたユーザーは、次のようなさまざまな強力なパラメータを設定できます。

- cursor_sharing
- ddl_lock_timeout
- optimizer_mode
- parallel_degree_limit
- plsql_code_type
- plsql_debug
- plsql_warnings
- resource_manager_plan

たとえば、***plsql_code_type***、***plsql_debug***、***plsql_warnings*** を設定できる権限を開発者に付与する必要があるものの、それ以外のパラメータは設定できないようにする必要があるとします。ロックダウン・プロファイルを使用すれば、ユーザーが必要とするパラメータだけが有効になるように alter system 権限の範囲を狭め、他のパラメータは無効にすることができます。たとえば、次の文を発行すると、こうした目的に合わせて Dev_PDB ロックダウン・プロファイルをカスタマイズできます。

```
alter lockdown profile Dev_PDB
disable statement =
('ALTER SYSTEM')
clause = ('SET')
option = ALL EXCEPT
('plsql_code_type'
,'plsql_debug'
,'plsql_warnings');
```

この文を実行すると、alter system 権限を付与された開発者が設定できるパラメータは次の 3 つに制限されます。

» plsql_code_type

» plsql_debug

» plsql_warnings

技術的な細かいことで重要なのは、このような制限は実行時に評価および適用される点です。つまり、ロックダウン・プロファイルを適用することによって無効になるオブジェクトはありません。

技術的には、ロックダウン・プロファイルを PDB に適用できるのは共通ユーザーだけで、このユーザーには通常 SysDBA または ***alter system*** のいずれかが付与されています。たとえば、このようなユーザーで適切な PDB に接続し、次のようなコマンドを発行します。

```
alter system set pdb_lockdown = Dev_PDB;
```

Oracle Data Guard Brokerの機能強化

Oracle Data Guard Broker の機能が強化されたことにより、12.2 では PDB レベルでフェイルオーバーができるようになりました。12.1 は、データ・レプリケーションという意味で両極端の構成に対応しています。

» 一方の極にあるのは、CDB レベルで Oracle Data Guard（できれば Oracle Active Data Guard）を構成するものです。これは、多数のものを一括管理する（また、それにより運用コストを削減する）好例です。いったんこれを CDB レベルで構成すると、Oracle（Active）Data Guard によってすべての PDB のトランザクションが 1 つのストリームでプライマリからスタンバイにレプリケートされます。

» もう一方の極にあるのは、個々の PDB レベルでデータをレプリケートするもので、これに最適な選択肢は Oracle GoldenGate です。これはアクティブ-アクティブ構成でも使用できるため、真の停止時間ゼロのアップグレードをサポートできます。

しかしながら、一部のユーザーはこれら 2 つの極端な構成の中間にあたる機能を必要としています。12.2 では、Data Guard Broker の機能が強化され、PDB 単位のフェイルオーバーとでも呼ぶべき機能を使用できるようになりました。この機能を有効にするには、2 つのサーバーに CDB のペアを 2 つ用意し、それぞれの中で 1 つをプライマリ、もう 1 つをスタンバイとし、逆"方向"にレプリケーションします。PDB レベルの障害が発生すると、スタンバイ PDB は同じサーバーにある別の CDB に移動され、そこで新しいプライマリとなります。この構成では共有ストレージを使用するため、データファイルの物理的な移動やコピーは不要です。したがって、このプロセスは最小限の停止時間で完了できます。しばらくしたら、この新しいプライマリ PDB 用のスタンバイを作成できます。

このことを示しているのが図 1 です。

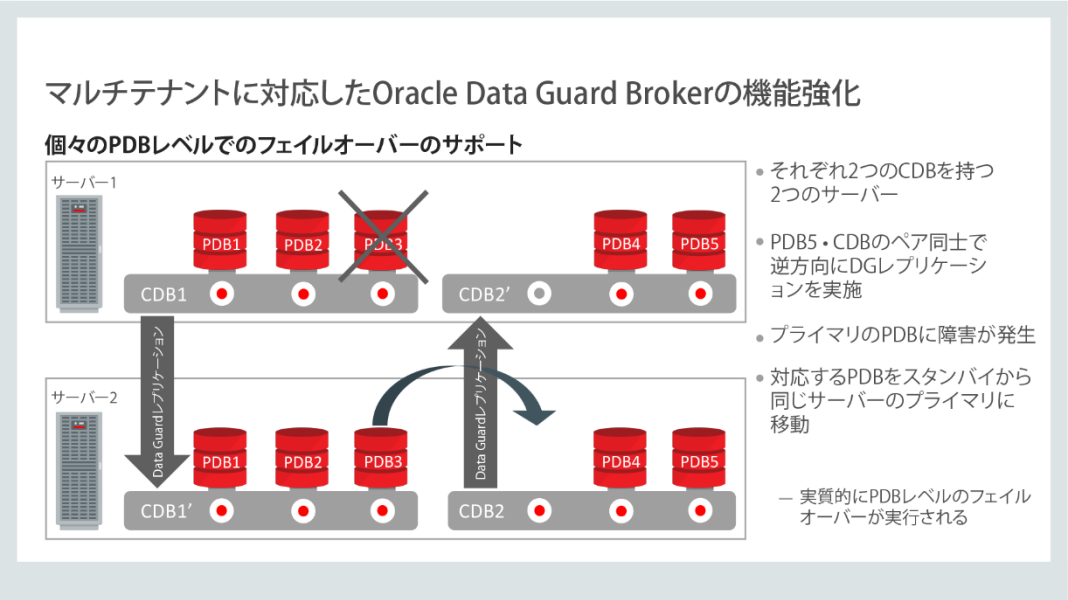

図1：マルチテナントに対応したOracle Data Guard Brokerの機能強化

Software as a Service

すでに説明したように、12.1 では Software as a Service（SaaS）が非常に重要な Oracle Multitenant のユースケースでした。何しろ、このユースケースが製品名の由来なのですから。"スケールメリットを伴う独立性と機敏性"という表現は SaaS にぴったりです。12.2 では、この分野の機能を大幅に拡張するためにいくつかの重要な機能を強化しています。ここからは、このことについて説明します。

このホワイト・ペーパーのほかの項で、SaaS をホテルの管理にたとえました。あのたとえは、大部分の機能を一元的な管理者に委任する統合環境で独立性を確保するための要件をとてもわかりやすく示しています。もう 1 つの優れたたとえとして、SaaS のスケールメリットを説明するスポーツ・ジムの会員権の話があります。たとえば、新しいスポーツ・ジムを作り、会員を 1,000 人募集するとします。ランニング・マシンは何台ぐらい購入すればよいでしょうか。せいぜい 20 台でしょう（1 月の最初の週を除けば、20 台すべてがひっきりなしに使用されることはあまりなさそうですが）。

これは理解しておくべきとても重要な点なのですが、Oracle Multitenant を基盤とする SaaS モデルでは単に効率性が上がるのではなく、数桁違いで向上します。程度が違うのではなく、種類が変化します（ちなみに、DBaaS で肝心なのは、SaaS ビジネス・モデルと同じスケールメリットをデータベース管理全般でも得ることですが、このトピックについての説明は別の資料に譲ります）。このトピックに関して言えるのは、SaaS は ISV だけのためにあるのではないということです。ISV に最適なのは確

かですが、フランチャイズ・モデルを運営しているビジネスや、世界中の子会社に同じビジネス手法を導入しているグローバル企業にも非常に適しています。これについて説明するために、ここからはSaaS の ISV のエンドユーザーもフランチャイジーも、グローバル企業の子会社も、ついでに言えばこの方法で管理されている自己完結型の事業単位も、すべてテナントという用語で呼ぶことにします。同様に、こうしたさまざまなテナントに一元的に対応するサービス・プロバイダのことを ISV という用語で呼びます。

SaaS ユースケースのメリットについて考える場合は、考慮すべき当事者に 2 つのタイプがあることを理解しておく必要があります。それぞれに関心事と要件が異なるため、SaaS モデルから得られるメリットも異なります。

エンドユーザー	ISV – スケールメリット
個々の要件に合わせてカスタマイズされた充実したアプリケーション環境	非常にスケーラブルなインフラストラクチャ
他のテナントからの独立性	すべてのコンピューティング・リソース（CPU、メモリ、ストレージ、I/O）の使用効率が極めて高い
ユーザー・ライフ・サイクル全般に平易なセルフサービス機能がある	必要な場合はきめ細かく制御しながら多数のテナントを一括管理できる
低価格でサービスを利用できる	プロビジョニング、ロードバランシング、柔軟なリソース配分、保守などの操作の機敏性
	採用しやすく使い勝手が良い

エンドユーザーにとってのメリットはとても重要ですが、簡単な説明にとどめます。充実したアプリケーション環境を開発するのは ISV の責任です。ここでも Oracle Database が理想的な開発環境であることは言うまでもありませんが、このホワイト・ペーパーのトピックから外れるため、世界でもっとも先進的なこのリレーショナル・データベースに代々受け継がれてきた豊富な機能、柔軟性と生産性、SQL と PL/SQL、無数の開発ツールについての説明は割愛します。ここで大いに役立つのが Oracle Multitenant です。スタンドアロンのアプリケーションは、マルチテナント対応に設計し直さなくても PDB でそのまま実行できるからです。このように、Oracle Multitenant は即席のクラウド・アーキテクチャになります。テナントの独立性については本書の別の項である程度詳しく説明していますが、PDB は本質的に独立しているうえ、Oracle Security の一連のテクノロジーでこれが補完されていて、テクノロジーはすべて Oracle Multitenant と互換性があります。セルフサービス・ライフ・サイクル管理は、エンドユーザーの要件に沿って ISV 側で簡単に構成できます。セルフサービス・ライフ・サイクル管理に使用される操作は非常に複雑ですが、それぞれの操作は ***alter pluggable database…***などの単純な SQL 文に巧みにカプセル化されています。最新のデータベース・クラウド環境では、そうした操作を RESTful API にラッピングし、Oracle Enterprise Manager Cloud Control（Oracle EMCC）か、OpenStack/Chef や Puppet といった業界標準の他の自動フレームワークを使用して連携を図るのが一般的です。Oracle Multitenant によってスケールメリットがもたらされることから、ISV は極めて効率的に事業を運営できるようになるため、サービスを低価格で提供できます。ISV は、削減された分をエンドユーザーに還元しても事業利益を上げることができるため、価格を下げることができます。

ここからは、ISV にとってのメリットについて説明します。ISV のアプリケーションは、かつてオンプレミス・デプロイメント向けに設計されたアプリケーション群と、SaaS デプロイメント・モデル向けに設計された"クラウド生まれ"のアプリケーション群の 2 つに分類できます。

かつてオンプレミス・デプロイメント向けに設計されたスタンドアロン・アプリケーションのほとんどは、何年も開発を重ねて機能の充実が図られている特徴を持ちますが、時代遅れのデプロイメント・モデルが足かせとなり、機能の観点からすればはるかに優れている場合が多いにもかかわらず、最新の SaaS アプリケーションには太刀打ちできません。Oracle Multitenant を使用しないとす

ると、ISV に残されるのは 2 つの好ましくない選択肢です。1 つは、マルチテナント対応のアプリケーションに設計し直すという選択肢です。この場合は通常、再構築とテストに甚大な労力が必要となるうえ、パフォーマンス、独立性、機能性を大幅に譲歩することが必要となります。もう 1 つの選択肢は、ホスト・モデルに移行し、第 1 世代のクラウド環境の VM 内で各テナントをスタンドアロンで実行することです。パブリック・クラウドの Infrastructure as a Service（IaaS）は一般に自動化が十分になされツール類も整備されていますが、すでに説明したように、ここでテナントをホストするこのモデルではコストが一次関数的に増加します。つまり、サポートするテナント数を倍にすると、必要な VM の数も倍になります。これでは本当にスケールメリットがあるとは言えません。このクラスのアプリケーションの場合は、Oracle Multitenant を採用することで、本当にスケールメリットがあるクラウド・アーキテクチャがすぐ手に入ります。

さらに新しい“クラウド生まれ”のアプリケーションでは、“行ベースの統合”とも表現されるモデルが採用されているのが一般的です。このモデルでは基本的に複数のテナントを 1 つのデータベースでホストし、各行が属するテナントを指定する列（Tenant_ID など）を表ごとに作り、この列で各テナントのデータを区別します。すべてのテナントのデータが同じ表に混在しているため、データの分離はアプリケーションに任されています。たとえば、あるテナントが注文の Web ページを調べようとして“私の注文を表示してください”という問合せを行った場合は、次のような SQL 文を構成するアプリケーション・コードで回答できるでしょう。

```
select * from orders
where Tenant_ID = :Tenant_ID;
```

最初の行（黒字で記載）がビジネス・ロジックで、2 行目（赤字で記載）がテナント間の独立性を確保するアプリケーション機能と見なすことができます。ほとんどの場合、当面はこれで非常にうまく行きますが、重大な制限事項がいくつかあります。たとえば、標準的なアプリケーション開発ツールを使用した場合は、必須の Tenant_ID 条件が（追加される場合があるとしても、）すべてのケースで追加される保証がないため、テナントにはそうしたツールを使用してカスタム・スクリーンやカスタム・レポートを開発する余地がありません。Oracle Business Intelligence Enterprise Edition などの最新の BI ツールや、独自に選択したどのようなツールでも、ツールの使用を許可するなど論外です。

テナントの移動性も大きな課題となります。通常、最初はどのテナントも"同じように作成"されますが、ご存知のとおり、成長のペースを予測するのは困難です。非常に急速に成長するテナントは、ホストされているサーバーの容量を簡単に超えてしまう恐れがあります。そのため、大きくなったテナントと、そのテナントと一緒に統合されている他のテナントの両方で、サービスの質が低下します。当然ながら最善の解決策はロードバランシングですが、それは通常、大きくなったテナントか現在同居している他のテナントのいずれかを、あまり負荷が高くないサーバーに移動することを意味します。行ベースの統合の課題は、このロードバランシングが理論的には簡単でも実際に行うのは非常に難しい点です。すべてのテナントのデータがまさに同じ表に存在しているため、行単位で操作しなければこの種の分離はできません。これには非常に時間がかかると予測されることから、これに伴うアプリケーション停止時間も著しく長くなる恐れがあります。ここでまた、可用性関連の SLA に違反することなくパフォーマンス関連の SLA を維持するにはどうすればよいか、という大きなジレンマにぶつかります。

要するにこれら 2 つの課題から言えるのは、アプリケーション・レイヤーにマルチテナンシーを実装するクラウド生まれのアプリケーション（通常、行ベースで統合する手法を使用）のスケールメリットは、独立性と機敏性を犠牲にして成り立っていることです。このクラスのアプリケーションの場合は、Oracle Multitenant を採用することで、スケールメリットを譲歩しなくても独立性と機敏性が手に入ります。

12.1のSaaSに対応したOracle Multitenant

12.1 の Oracle Multitenant にも、SaaS ユースケースに対応する優れた機能はありました。これを示しているのが図 2 です。

PDB はテナントごとのコンテナとして理想的です。かつてスタンドアロンだったアプリケーションを PDB にインストールでき、何も変更せずにそこで実行できます。各テナントのデータは、それぞれの専用コンテナに整然と分離されます。初期化されたアプリケーションがあらかじめインストールされた新しい PDB をテナント用にプロビジョニングすることで、新しいテナントを迅速に"同居させる"ことができます。そのために ISV 側でしなければならない作業は、空の PDB で簡単なアプリケーション・インストール・スクリプトを 1 度だけ実行し、これを"テナント・シード"に指定することだけです。新しいテナントの PDB は、このテナント・シードのクローンとして非常に簡単かつ迅速にプロビジョニングされます。

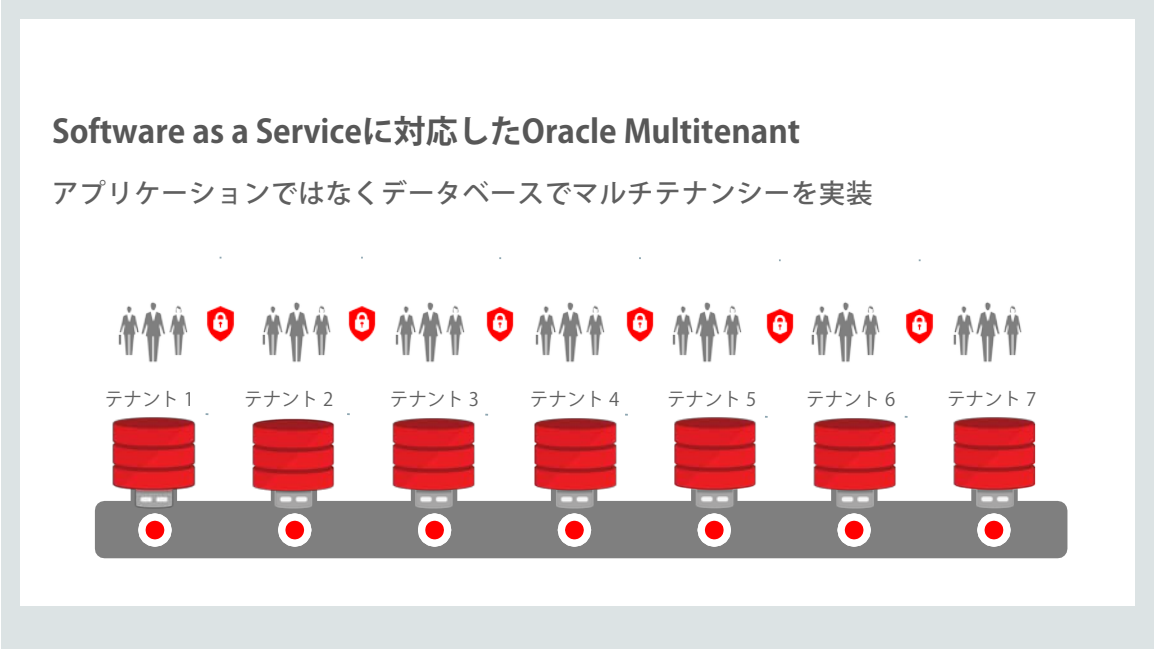

図2：Software as a Serviceに対応したOracle Multitenant

12.1 の Oracle Multitenant で SaaS をサポートすることには複数のメリットがあります。

» テナント間の独立性が高い

» 新しいテナントを極めて迅速にセットアップできる

» 相対的に少ない台数のサーバーで多数のテナントをサポートできるため、設備投資が大幅に減少する

» 多数のテナントの標準的な DBA 作業（バックアップ、パッチの適用など）をまとめて管理できるため、運用費が大幅に減少する

ただし、***データベース***管理者の観点からすれば多数の PDB をまとめて管理できると言えますが、***アプリケーション***管理者にとってはそうではありません。PDB ごとにアプリケーションのデータベース・コンポーネントの完全なコピーがあるため、アプリケーションのアップグレード・スクリプトはそれぞれのテナント PDB で個別に実行する必要があります。必要なのは、多数のものを一括管理するメリットを、データベース管理者からアプリケーション管理者にまで拡大できることです。

12.2のSaaSに対応したOracle Multitenant

こうした重要な SaaS の要件に対処するために、12.2 にはアプリケーション・コンテナという新しい概念が導入されています。このトピックについてはオラクルが発行している他のホワイト・ペーパーで詳しく説明していますが、おもな機能を以下にまとめます。

アプリケーション・コンテナの構成要素は、アプリケーション・ルート、オプションのテナント・シードおよびゼロ個または多数のテナント PDB です。アプリケーション定義はアプリケーション・ルートに一度だけインストールします。そこからテナント・シードを作成することができ、その後はテナント・シードをクローニングするだけで新しいテナントをプロビジョニングできます。このことを示しているのが図 3 です。

技術的には、メタデータのスケルトンがテナント・シードとテナント PDB のデータ・ディクショナリに格納されます。これを構成するのは“スタブ”と呼ばれる一連のポインタで、アプリケーション・ルートに格納されている完全なメタデータ定義を参照します。Oracle Multitenant 12.1 ではこれとまったく同じメカニズムを使用して、CDB のルートにある Oracle 提供のメタデータの完全な定義をさまざまな PDB と共有します（ただし、PDB 自体に保持されるのは一連のシステム・メタデータのスケルトンにすぎません）。12.2 にはアプリケーション・コンテナがあるため、この機能はユーザーも使用できます。

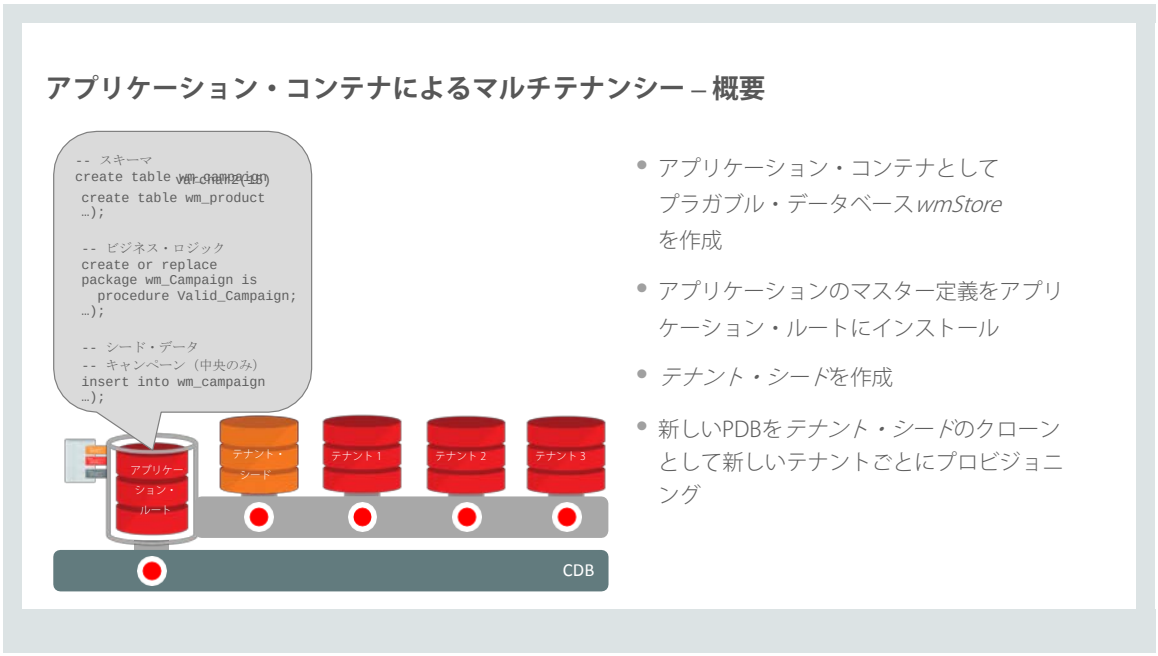

図3アプリケーション・コンテナの概要

アプリケーションの保守

このモデルの長所は、（アプリケーション・ルートにインストールされている）アプリケーションのマスター・コピーが 1 つしかないため、アプリケーションのパッチ適用やアップグレード・スクリプトをアプリケーション・ルートで実行するだけでよい点です。その後は、個々のテナントで新しいマスター・アプリケーション定義を単純に同期することができます。重要なのは、これをすべて同時に実行する必要がないことです。同期するタイミングは個々のテナントのスケジュールに沿って選択できます。そのため、全社レベルのアプリケーション アップグレードを実行できる都合のよいタイミングを調整する（、または、単純ながらも一般的には嫌がられるやり方ですが、一方的に実行する日を決定する）手間が省けます。これに伴い、アプリケーション・アップグレードを実施するという迷惑な作業が大幅に減少するため、アプリケーション開発プロセスの機敏性を大幅に向上させることができます。

アプリケーションのパッチ適用やアップグレードについて考慮すべき重要点は、アップグレードの実行の影響を個々のテナントにとどめることです。通常、アプリケーションのアップグレードにはスキーマ変更が伴い、関連データの変更が伴うこともあります。わかりやすい例として、Full_Name という新しい列を Person 表に導入する場合を考えてみましょう。この場合、各テナントではこの列を表に追加し、First_Name と Last_Name を結合した値をそこに移入することが必要になります。当然ながら、影響を受ける表の全体スキャンが必要となるため、膨大な時間がかかるでしょう。アプリケーション・コンテナ・テクノロジーを使用すると、この影響が及ぶ範囲は、アップグレードするテナントに限定されます。これは、アプリケーション・ルートに対してアプリケーション・アップグレードを実行する場合も同じで、アップグレードの影響が個々のテナントに及ぶことはありません。Oracle Multitenant だけでアプリケーション・アップグレードを実行した場合にこのすべてが手に入ります。これ以外にも、Oracle にはエディションベースの再定義（EBR）や GoldenGate などのさまざまな補完テクノロジーがあり、これらはすべて Oracle Multitenant（具

体的にはアプリケーション・コンテナ）と互換性があります。これらのテクノロジーについて詳しくは、オラクルが発行している他の資料を参照してください。アプリケーション・アップグレードに伴う停止時間を、アプリケーション・コンテナでできる範囲以上に短縮する必要がある場合は、既存の機能を引き続き使用できることを思い出していただくため、ここでこれらのテクノロジーに言及しました。

テナントをまたぐ集計

ISV にとって重要なのは、SaaS アプリケーションのパフォーマンスをすべてのテナントにわたって監視できることです。たとえば、今四半期の注文数といった特定の重要なメトリックを使用する場合があります。第 1 世代のクラウド・アーキテクチャの場合は、各テナントを順番にまわり、注文数をカウントする SQL 文を発行し、回答をスプレッドシートに書き出し、すべてのテナントのスプレッドシートを集計することを繰り返す、という面倒でエラーを誘発しがちなプロセスが必要でしょう。この簡単な例としては次のような問合せが考えられます。

```
select :Tenant_Name
,    count(*) from orders
where current_quarter = 'Y';
```

12.2 の Oracle Multitenant を使用すれば、containers()という新しいコンストラクトを使用してこの問合せを少し書き直すことで、次のような 1 つの SQL 文でこれを実行できます。

```
select con$name
,    count(*)
from containers(orders)
where current_quarter = 'Y'
group by con$name;
```

この問合せは、アプリケーション・ルートで一度実行するだけで構いません。技術的な話をすると、containers()コンストラクトは、テナント PDB ごとに SQL 文が繰り返し（かつパラレルで）実行されることを意味します。結果はアプリケーション・ルートに返され、そこで集計されます。このことを示しているのが図 4 です。

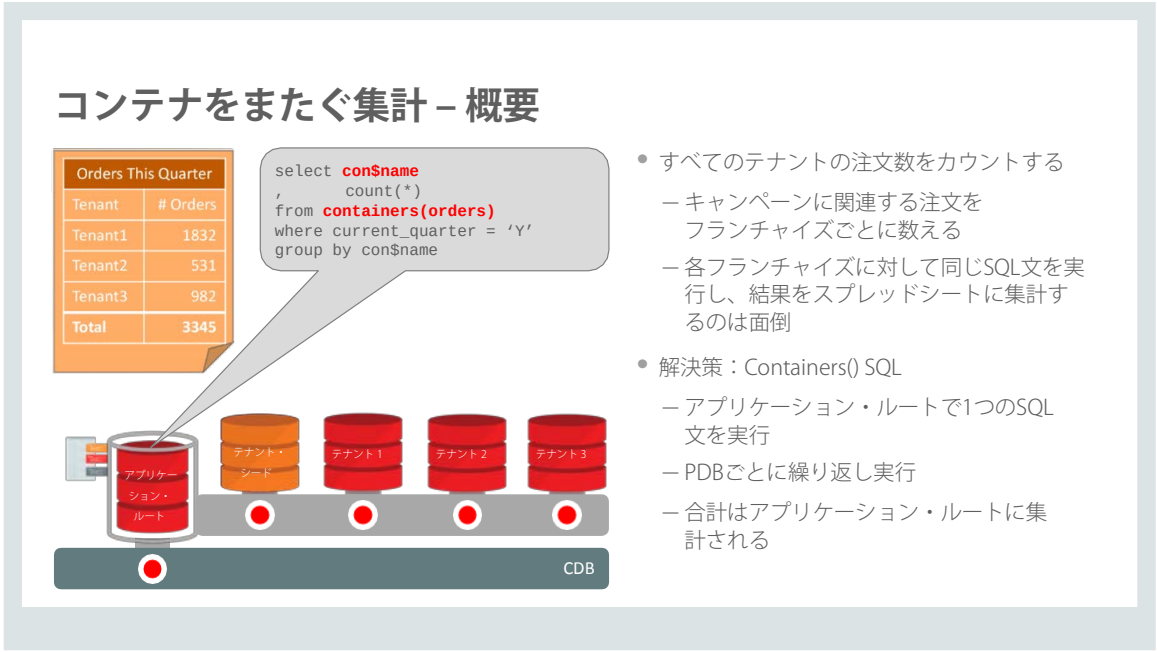

図4：テナントをまたぐ集計

アプリケーション・コンテナ - 評価

ここまで、12.1 の Oracle Multitenant で得られる次のようなメリットを、12.2 の Oracle Multitenant で導入された新しいアプリケーション・コンテナ機能で SaaS ユースケースにもたらす方法を説明してきました。

» テナント間の独立性が高い

» 新しいテナントを極めて迅速にセットアップできる

» 相対的に少ない台数のサーバーで多数のテナントをサポートできるため、設備投資が大幅に減少する

» 多数のテナントの標準的な DBA 作業（バックアップ、パッチの適用など）をまとめて管理できるため、運用費が大幅に減少する

このほかにも大幅な機能強化が図られているため、アプリケーション管理とデータベース管理の両方の面で多数のテナントの一括管理が可能です。

SaaSソリューションの要件の確認

この問題を解く鍵はいくつもありますが、Oracle Multitenant にはそれぞれの要件をサポートする強力な機能があります。

» テナントの分離
  » 各テナントのデータは、自己完結型の PDB に隔離されます。
» 一元管理
  » アプリケーションのマスター定義がアプリケーション・ルートに一元化されており、これですべてのテナントをサポートします。
  » アプリケーション・ルートで containers()問合せを使用してテナントをまたぐ集計を実行することで、グローバルなメトリックを集計できます。
  » アプリケーション・コンテナには複数のサーバーを含めることができます。
» 機敏性
  » オンライン・クローニング、リフレッシュ可能なクローン、PDB 再配置など、PDB ベースのプロビジョニングのすべての機能を使用できます。
  » アプリケーション・アップグレードをテナント単位で実行できるため、システム全体を停止する必要がありません。
» セキュリティ – 本質的に独立している PDB に次の機能が強化されています。
  » ロックダウン・プロファイル
  » 新しいパラメータ（path_prefix および create_file_dest）
  » PDB OS 資格証明
  » Oracle Security の主要な機能はすべて、Oracle Multitenant と完全に互換性があります。
» 位置の透過性 – さまざまな理由でワークロードを再配置できる柔軟なクラウド展開に不可欠の機能です。この要件は次の機能によってサポートされます。
  » プロキシ PDB
  » コンテナ・マップ
» 真のスケールメリット – Oracle Multitenant で実現できます。
  » 1 つの CDB に 4k 個の PDB
  » リソースの効率的な利用

結論

12.2 の Oracle Multitenant を使用すると、デプロイ先が Oracle Cloud かデータセンターかに関係なく、***スケールメリットを伴う独立性と機敏性を実現できます***。12.1 の Oracle Multitenant も機能的には優れており、開発/テスト、多様なアプリケーションの統合、Software as a Service（SaaS）など、幅広いユースケースに対する優れたサポート力がありますが、これらの各分野で大幅な機能強化が図られています。

開発やテストを担当する現代の組織は革新的で生産性が高く、優れた機敏性を必要としています。アプリケーションのデータベース・コンポーネントの場合、この敏捷性を達成するには、データベースを極めて効率的にプロビジョニングする機能が必要です。12.2 の Oracle Multitenant は、いくつかの強力なオンライン・プロビジョニング機能をサポートしています。たとえば、ソース側で停止時間を発生させることなくクローニングができるプラガブル・データベース（PDB）、非常に大規模な本番データベースの"開発マスター"クローンに対してもソースにほとんど影響を与えずにシンプルかつ効率的にフレッシュ・データを追加できる増分リフレッシュ機能がサポートされています。また、PDB のオンライン再配置を使用すれば、アプリケーションの停止を伴わずにクラウドに単純に移行するという理想が現実になります。クラウド・プロバイダの場合も同様で、このオンライン再配置機能を使用することにより、稼働中のアプリケーションにほとんど影響を与えずにロードバランシングを実行できるため、絶えず変化するワークロードを柔軟に調整できます。

第 1 世代のクラウド・アーキテクチャでは個々のデータベースが専用の VM でホストされるため、データベースの数とコストの関係は一次関数的ですが、Oracle Multitenant はこれとは異なり、Database Cloud に真のスケールメリットをもたらします。規模が大きくなるほどスケールメリットも大きくなるため、この可能性を現実のものにするには、統合の障害をすべて排除することが不可欠です。その主要な要素の 1 つがテナントの独立性です。12.2 では、PDB 間のメモリ管理、汎用ストレージでの I/O 管理、ロックダウン・プロファイルなど、大幅な機能強化が図られていて、どの機能も特定のユースケースに適した独立性のレベルを簡単に構成して適用できるようになっています。また、フラッシュバック PDB、連続問合せ通知（CQN）、ヒート・マップといった Oracle Database の重要な機能が PDB で完全にサポートされます。自動ワークロード・リポジトリ（AWR）レポートが PDB 単位でサポートされ、キャラクタ・セットが異なる PDB を 1 つのマルチテナント・コンテナ・データベース（CDB）に統合することができ、Data Guard Broker の機能強化により PDB 単位のフェイルオーバーが可能です。また、Oracle Cloud と Oracle Exadata では、CDB に収容できる PDB の最大数が 4,096 に増加したため、真のクラウド規模の統合が可能となりました。

12.2 の Oracle Multitenant は、Software as a Service（SaaS）用に強力な新しいアプリケーション・コンテナ機能を提供します。これにより、DBA だけでなくアプリケーション管理者も、多数のテナントを一括管理できるというメリットを享受できます。オンプレミスでのスタンドアロン・デプロイメント用として過去に設計されたアプリケーションの場合は、アプリケーション・コンテナを即席のクラウド・アーキテクチャとして使用できます。クラウド生まれのアプリケーションの場合は、SaaS の特徴である優れたスケールメリットを得るために敏捷性を犠牲してきました。アプリケーション・コンテナは、こうしたスケールメリットを損なうことなく、PDB の敏捷性を完全に達成します。マスター・アプリケーションの定義を、アプリケーション・ルートという単一コンテナにインストールできるようになりました。このアプリケーション・コンテナに含まれる個々のテナントでの操作にはこのマスター定義が使用されます。その結果、アプリケーションの保守業務（アプリケーション・パッチの適用など）は 1 箇所で行い、個々のテナントにはそれぞれのスケジュールに沿ってこの新しいバージョンを単純に同期することが可能です。さまざまな集計をコンテナ間で実行できるようになったことで、ダッシュボードの作成が容易になりました。ISV はこのダッシュ

ボードを元に、今四半期の注文数や総収益などの主要メトリックスを使用して、アプリケーションの全体的なパフォーマンスを評価できます。これらの機能は ISV のためだけにあるのではありません。世界中の複数の子会社で同じビジネス手法を実践しているグローバル企業やフランチャイズなど、その他多くのビジネス・モデルでも同様に利用できます。

12.2 の Oracle Multitenant：ぜひともクラウドをご利用ください。

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

海外からのお問い合わせ窓口
電話：+1.650.506.7000
ファクシミリ：+1.650.506.7200

ORACLE

Integrated Cloud Applications & Platform Services

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Oracle Multitenant：Oracle Database 12c Release 2 の新機能 2017 年 3 月
著者：Oracle Database の Product Management 担当 Senior Director、Patrick Wheeler

Oracle is committed to developing practices and products that help protect the environment