(19)日本国特許庁(JP)　(12)特　許　公　報(B2)　(11)特許番号
特許第6675419号
(P6675419)
(45)発行日　令和2年4月1日(2020.4.1)　(24)登録日　令和2年3月12日(2020.3.12)

(51)Int.Cl.　F I
G06F 16/182 (2019.01)　G06F 16/182

請求項の数 14　(全 36 頁)

(21)出願番号	特願2017-550489 (P2017-550489)
(86)(22)出願日	平成28年4月20日 (2016.4.20)
(65)公表番号	特表2018-514028 (P2018-514028A)
(43)公表日	平成30年5月31日 (2018.5.31)
(86)国際出願番号	PCT/US2016/028420
(87)国際公開番号	WO2016/172195
(87)国際公開日	平成28年10月27日 (2016.10.27)
審査請求日	平成30年11月28日 (2018.11.28)
(31)優先権主張番号	62/150,191
(32)優先日	平成27年4月20日 (2015.4.20)
(33)優先権主張国・地域又は機関	米国 (US)
(31)優先権主張番号	62/150,188
(32)優先日	平成27年4月20日 (2015.4.20)
(33)優先権主張国・地域又は機関	米国 (US)

(73)特許権者　502303739
オラクル・インターナショナル・コーポレイション
アメリカ合衆国カリフォルニア州９４０６５レッドウッド・シティー，オラクル・パークウェイ５００
(74)代理人　110001195
特許業務法人深見特許事務所
(72)発明者　ヘッジ，ビディヤ
インド、５６００２９　バンガロール、チッカ・アダゴディ、セカンド・クロス・ロード、ナンバー・１８、セント・ジョンズ・ウッズ、プレスティージ・ストリート、レキシントン・タワー、オラクル・インディア・プライベート・リミテッド内
最終頁に続く

(54)【発明の名称】シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて提供するためのシステムおよび方法

(57)【特許請求の範囲】
【請求項１】
シャードされたデータベースへのアクセスを提供するためのシステムであって、前記システムは、
プロセッサを含むコンピュータと、
データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するアプリケーションサーバまたはデータベース環境とを備え、
前記データベースは、
データベースドライバと、
前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、
前記接続プールおよび前記データベースドライバはともに動作することにより、前記接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、
前記データベースドライバは、接続要求の処理において使用するために、シャードトポロジとして、シャードキー範囲を前記データベース内のシャードの位置にキャッシュし、
前記データベースドライバは、接続要求の一部として、クライアントアプリケーションが前記データベースにアクセスすることを可能にするように構成されており、前記クライアントアプリケーションが前記データベースにアクセスすることを可能にすることは、前記データベース内の適切なシャードの位置を判断することと、前記シャードトポロジを使

用することと、前記データベースの前記適切なシャードへの前記クライアントアプリケーションによるアクセスを提供することとを含み、
シャードトポロジレイヤは、後続の接続要求が、シャードディレクタまたはリスナーコンポーネントをバイパスして代わりに適切なシャードまたはチャンクへの高速キー経路アクセスを使用することを可能にするように構成されており、前記シャードディレクタまたはリスナーコンポーネントは、データベースシャードへのソフトウェアクライアントアプリケーションによるアクセスを提供するように動作する、システム。
【請求項２】
シャードされたデータベースへのアクセスを提供するためのシステムであって、前記システムは、
プロセッサを含むコンピュータと、
データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するアプリケーションサーバまたはデータベース環境とを備え、
前記データベースは、
データベースドライバと、
前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、
前記接続プールおよび前記データベースドライバはともに動作することにより、前記接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、
前記データベースドライバは、接続要求の処理において使用するために、シャードトポロジとして、シャードキー範囲を前記データベース内のシャードの位置にキャッシュし、
前記データベースドライバは、接続要求の一部として、クライアントアプリケーションが前記データベースにアクセスすることを可能にするように構成されており、前記クライアントアプリケーションが前記データベースにアクセスすることを可能にすることは、前記データベース内の適切なシャードの位置を判断することと、前記シャードトポロジを使用することと、前記データベースの前記適切なシャードへの前記クライアントアプリケーションによるアクセスを提供することとを含み、
前記接続プールおよび前記データベースドライバは、前記データベースへの接続のチェックアウト中または後の時点のうちの少なくとも一方においてクライアントアプリケーションがシャードキー情報を提供することを可能にするように構成されており、
前記システムは、前記シャードキー情報を用いて、前記クライアントアプリケーションが使用する前記データベースの適切なシャードへの、前記クライアントアプリケーションによるダイレクトアクセスを提供するように、構成されている、システム。
【請求項３】
シャードされたデータベースへのアクセスを提供するためのシステムであって、前記システムは、
プロセッサを含むコンピュータと、
データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するアプリケーションサーバまたはデータベース環境とを備え、
前記データベースは、
データベースドライバと、
前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、
前記接続プールおよび前記データベースドライバはともに動作することにより、前記接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、
前記接続プールおよび前記データベースドライバは、前記データベースへの接続のチェックアウト中または後の時点のうちの少なくとも一方においてクライアントアプリケーションがシャードキー情報を提供することを可能にするように構成されており、

前記シャードキー情報は前記システムによって用いられて、前記クライアントアプリケーションが使用する前記データベースの適切なシャードへの、前記クライアントアプリケーションによるダイレクトアクセスを提供する、システム。

【請求項４】

前記データベースドライバおよび前記接続プールは、前記クライアントアプリケーションによって指定されたシャードキーを認識するように、かつ、前記クライアントアプリケーションが前記クライアントアプリケーションに対応付けられた特定のシャードおよびチャンクに接続することを可能にするように、構成されている、請求項２または３に記載のシステム。

【請求項５】

前記接続プールは、接続をそのシャードキーによって識別するように、かつ、同じシャードキーの要求をクライアントアプリケーションから受けたときは接続を再使用することを許可するように構成されている、請求項１～４のいずれかに記載のシステム。

【請求項６】

特定のシャードまたはチャンクへの接続が前記接続プールにない場合は、別のチャンクへの既存の利用できる接続を、その用途を変更して再使用することを試みる、請求項１～５のいずれかに記載のシステム。

【請求項７】

シャードされたデータベースへのアクセスを提供する方法であって、前記方法は、

コンピュータにより、データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するステップを含み、

前記データベースは、

データベースドライバと、

前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、

前記接続プールおよび前記データベースドライバはともに動作することにより、前記接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、

前記データベースドライバは、接続要求の処理において使用するために、シャードトポロジとして、シャードキー範囲を前記データベース内のシャードの位置にキャッシュし、

前記データベースドライバは、接続要求の一部として、クライアントアプリケーションが前記データベースにアクセスすることを可能にするように構成されており、前記クライアントアプリケーションが前記データベースにアクセスすることを可能にすることは、前記データベース内の適切なシャードの位置を判断することと、前記シャードトポロジを使用することと、前記データベースの前記適切なシャードへの前記クライアントアプリケーションによるアクセスを提供することとを含み、

シャードトポロジレイヤは、後続の接続要求が、シャードディレクタまたはリスナーコンポーネントをバイパスして代わりに適切なシャードまたはチャンクへの高速キー経路アクセスを使用することを可能にするように構成されており、前記シャードディレクタまたはリスナーコンポーネントは、データベースシャードへのソフトウェアクライアントアプリケーションによるアクセスを提供するように動作する、方法。

【請求項８】

シャードされたデータベースへのアクセスを提供する方法であって、前記方法は、

コンピュータにより、データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するステップを含み、

前記データベースは、

データベースドライバと、

前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、

前記接続プールおよび前記データベースドライバはともに動作することにより、前記

接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、
前記データベースドライバは、接続要求の処理において使用するために、シャードトポロジとして、シャードキー範囲を前記データベース内のシャードの位置にキャッシュし、
前記データベースドライバは、接続要求の一部として、クライアントアプリケーションが前記データベースにアクセスすることを可能にするように構成されており、前記クライアントアプリケーションが前記データベースにアクセスすることを可能にすることは、前記データベース内の適切なシャードの位置を判断することと、前記シャードトポロジを使用することと、前記データベースの前記適切なシャードへの前記クライアントアプリケーションによるアクセスを提供することとを含み、
前記接続プールおよび前記データベースドライバは、前記データベースへの接続のチェックアウト中または後の時点のうちの少なくとも一方においてクライアントアプリケーションがシャードキー情報を提供することを可能にするように構成されており、
前記シャードキー情報を用いて、前記クライアントアプリケーションが使用する前記データベースの適切なシャードへの、前記クライアントアプリケーションによるダイレクトアクセスを提供する、方法。
【請求項９】
シャードされたデータベースへのアクセスを提供する方法であって、前記方法は、
コンピュータにより、データを格納し提示するための複数のシャードを有するデータベースへのアクセスを提供するステップを含み、
前記データベースは、
データベースドライバと、
前記データベースに対して使用するための接続のプールを生成し維持する接続プールとに対応付けられており、
前記接続プールおよび前記データベースドライバはともに動作することにより、前記接続を用いて、前記データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供し、
前記接続プールおよび前記データベースドライバは、前記データベースへの接続のチェックアウト中または後の時点のうちの少なくとも一方においてクライアントアプリケーションがシャードキー情報を提供することを可能にするように構成されており、
前記シャードキー情報を用いて、前記クライアントアプリケーションが使用する前記データベースの適切なシャードへの、前記クライアントアプリケーションによるダイレクトアクセスを提供する、方法。
【請求項１０】
前記データベースドライバおよび前記接続プールは、前記クライアントアプリケーションによって指定されたシャードキーを認識するように、かつ、前記クライアントアプリケーションが前記クライアントアプリケーションに対応付けられた特定のシャードおよびチャンクに接続することを可能にするように、構成されている、請求項８または９に記載の方法。
【請求項１１】
前記接続プールは、接続をそのシャードキーによって識別するように、かつ、同じシャードキーの要求をクライアントアプリケーションから受けたときは接続を再使用することを許可するように構成されている、請求項７～１０のいずれかに記載の方法。
【請求項１２】
特定のシャードまたはチャンクへの接続が前記接続プールにない場合は、別のチャンクへの既存の利用できる接続を、その用途を変更して再使用することを試みる、請求項７～１１のいずれかに記載の方法。
【請求項１３】
命令を含むコンピュータ読取可能なプログラムであって、前記命令は、１つ以上のコンピュータによって読出されて実行されたときに、前記１つ以上のコンピュータに、請求項

７～１２のいずれかに記載の方法を実行させる、コンピュータ読取可能なプログラム。
【請求項１４】
請求項７～１２のいずれか１項に記載の方法を１つ以上のコンピュータに実行させるためのプログラム。
【発明の詳細な説明】
【技術分野】
【０００１】
著作権に関する注意
この特許文献の開示の一部は、著作権保護の対象となるものを含んでいる。この特許文献または特許開示の何者かによる複製に対しては、それが特許商標庁の特許ファイルまたは記録にある限り、異議を唱えないが、そうでなければ、いかなる場合もすべての著作権を留保する。
【０００２】
優先権の主張
本願は、２０１５年４月２０日に出願され「SYSTEM AND METHOD FOR PROVIDING DIRECT ACCESS TO A SHARDED DATABASE」と題された米国仮特許出願第６２／１５０，１９１号、２０１５年７月３０日に出願され「SYSTEM AND METHOD FOR PROVIDING DIRECT ACCESS TO A SHARDED DATABASE」と題された米国仮特許出願第６２／１９８，９５８号、および２０１５年４月２０日に出願され「SYSTEM AND METHOD FOR PROVIDING ACCESS TO A SHARDED DATABASE USING A CACHE AND A SHARD TOPOLOGY」と題された米国仮特許出願第６２／１５０，１８８号に基づく優先権の利益を主張し、上記出願各々を本明細書に引用により援用する。
【０００３】
発明の分野
本発明の実施形態は、概してアプリケーションサーバおよびデータベースに関し、具体的にはシャードされた（sharded）データベースへのアクセスを提供するためのシステムおよび方法に関する。
【背景技術】
【０００４】
背景
現代のウェブ指向のソフトウェアアプリケーションは、極めて大きなボリュームのデータを扱う必要性を含めて、スケーラビリティに関して増加している課題に直面している。たとえば、モバイルチャットシステム内で、メッセージを処理するのに必要なデータベーステーブルのサイズは、大幅に増しており、１つのテーブルのボリュームが、特定のアプリケーションのスケーラビリティを制限する要因になる可能性がある。この種の問題に対処するための一般的な手法は、データを複数のより小さなデータベースまたはシャード（shard）として提示するシャーディング（sharding）を利用することである。これらは、本発明の実施形態を使用できる環境の種類のいくつかの例である。
【発明の概要】
【課題を解決するための手段】
【０００５】
概要
ある実施形態に従い、シャードされたデータベースへのダイレクトアクセスを提供するためのシステムおよび方法を説明する。シャードディレクタまたはリスナーは、データベースシャードへの、ソフトウェアクライアントアプリケーションによるアクセスを、提供するように動作する。接続プール（たとえばユニバーサル接続プール（Universal Connection Pool）、ＵＣＰ）およびデータベースドライバ（たとえばＪａｖａ（登録商標）データベースコネクティビティ（Database Connectivity）、ＪＤＢＣ、コンポーネント）を、クライアントアプリケーションがシャードキーを接続チェックアウト中にまたは後の時点で提供することを許可するように、かつ、クライアントアプリケーションによって指

定されたシャードキーを認識するように、かつ、クライアントアプリケーションが特定のシャードまたはチャンクに接続することを可能にするように、構成し得る。この手法により、接続リソースを効率的に再使用することができ、適切なシャードにより速くアクセスすることができる。

【０００６】

ある実施形態に従うと、このシステムは、シャードされたデータベースへのアクセスを、キャッシュおよびシャードトポロジを用いて、可能にする。シャードされたデータベースに接続しているシャード認識クライアントアプリケーションは、接続プール（たとえばＵＣＰ）を用いて、シャードされたプール内のシャードされたデータベースのさまざまなシャードまたはチャンクへの接続を格納するまたはそれにアクセスすることができる。新たな接続が生成されると、シャードトポロジレイヤをデータベースドライバレイヤに構築することができ、これは、シャードキー範囲を学習してシャードの位置にキャッシュする。シャードトポロジレイヤは、クライアントアプリケーションからの後続の接続要求が適切なシャードまたはチャンクへの高速キー経路アクセスを使用することを可能にする。

【０００７】

ある実施形態に従うと、特定のシャードまたはチャンクへの、利用できる接続が、接続プールにない場合は、別のチャンクへの既存の利用できる接続を、用途を変更して再使用することを試みてもよい。

【０００８】

上記およびその他の実施形態を、以下でより詳細に説明する。

【図面の簡単な説明】

【０００９】

【図１】ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのシステムを示す。

【図２】ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのシステムをさらに示す。

【図３】ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのシステムをさらに示す。

【図４】ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのプロセスを示す。

【図５】ある実施形態に従う、シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて可能にするためのシステムを示す。

【図６】ある実施形態に従う、シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて可能にするためのシステムをさらに示す。

【図７】ある実施形態に従う、シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて可能にするためのシステムをさらに示す。

【図８】ある実施形態に従う、接続プールを用いて、シャードされたデータベースへの接続のプールを生成し維持するときのフローを説明する、シーケンス図を示す。

【図９】ある実施形態に従う、アプリケーションが接続プールなしでデータベースドライバを用いて、シャードされたデータベースへの接続をフェッチするときの、フローを説明する、シーケンス図を示す。

【図１０】ある実施形態に従う、シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて可能にするためのプロセスを示す。

【図１１】ある実施形態に従う、データベーストポロジクラス設計を示す。

【図１２】ある実施形態に従う、サービストポロジクラス設計を示す。

【発明を実施するための形態】

【００１０】

詳細な説明

上記のように、現代のウェブ指向のソフトウェアアプリケーションは、極めて大きなボリュームのデータを扱う必要性を含めて、スケーラビリティに関して増加している課題に

直面しており、１つのテーブルのボリュームが、特定のアプリケーションのスケーラビリティを制限する要因になる可能性がある。この種の問題に対処するための一般的な手法は、データを複数のより小さなデータベースまたはシャードとして提示するシャーディングを利用することである。このような環境に対するサポートを提供するために、さまざまな実施形態に従い、シャードされたデータベースへのアクセスを提供するためのシステムおよび方法を本明細書において説明する。

【００１１】

シャードされたデータベース

ある実施形態に従うと、シャーディングは、複数の独立した物理データベース全体にわたるデータの水平分割を用いるデータベーススケーリング技術である。各物理データベースに格納されるデータの部分をシャードと呼ぶ。ソフトウェアクライアントアプリケーション側からは、すべての物理データベースの集合体が１つの論理データベースのように見える。

【００１２】

ある実施形態に従うと、データベーステーブルは、たとえば特定のシャード内で各行が格納される場所を決める１つ以上の列として、シャードキー（SHARD_KEY）を用いることにより、分割することができる。シャードキーは、接続ストリングまたは説明（description）において、接続データ（CONNECT_DATA）の属性として与えることができる。

【００１３】

シャードのグループ分けは、シャードグループキー（SHARDGROUP_KEY）を用いたユーザ制御データ分割の形態を提供する別のレベルのシェアリングであり、データベースグループ（DBGROUPS）全体にわたってデータを分配するために任意で使用できる。たとえば次の通りである。

【００１４】

【数１】

```
(DESCRIPTION=(…)(CONNECT_DATA=(SERVICE_NAME=ORCL (SHARD_KEY=...)
(SHARDGROUP_KEY=...)))
```

【００１５】

シャードキーの例は、データベースにおける、VARCHAR2、CHAR、DATE、NUMBER、またはTIMESTAMPを含み得る。データベースにおいて指定されている国家言語支援フォーマットに従うシャードキーを提示する責任はユーザにある。ある実施形態に従うと、シャードされたデータベースは、シャードキーまたはシャードグループキーがない接続を受入れることもできる。

【００１６】

ある実施形態に従うと、システムパフォーマンスおよびデータアベイラビリティに対して再シャーディングが与える影響を減じるためには、各シャードをより小さな部分またはチャンクに再分割すればよい。各チャンクは、１つのシャードから別のシャードに移動させることができる再シャーディングの単位として機能する。チャンクはまた、シャードキーマッピングに対して間接的なレベルを追加することにより、ルーティングを単純化する。

【００１７】

たとえば、各チャンクは、ある範囲のシャードキー値に自動的に対応付けることができる。ユーザが提供したシャードキーを特定のチャンクにマッピングすることができ、このチャンクは特定のシャードにマッピングされる。データベース動作が、特定のシャード上に存在しないチャンクに対する動作を試みる場合は、エラーが発生するであろう。シャードグループを用いる場合、各シャードグループは、特定の値のシャードグループ識別子を有するチャンクの集合体である。

【００１８】

ある実施形態に従うと、シャード認識クライアントアプリケーションは、１つ以上のシ

ャーディング方法に基づいてデータが分割されたときの１つまたは複数のデータベースシャードに接続する機能を含むシャード化データベース構成と連携することができる。データベース動作が要求される度に、クライアントアプリケーションは、自身が接続する必要があるシャードを決定することができる。

【００１９】

ある実施形態に従うと、シャーディング方法を用いてシャードキー値を個々のシャードにマッピングすることができる。下記のようなさまざまなシャーディング方法をサポートすることができる。たとえば、ハッシュベースのシャーディングの場合、ある範囲のハッシュ値を各チャンクに割当て、データベース接続が確立されると、システムはハッシュ関数をシャーディングキーの所与の値に適用して対応するハッシュ値を計算し、このハッシュ値はこの値が属する範囲に基づいてチャンクにマッピングされる。範囲ベースのシャーディングの場合、ある範囲のシャードキー値が直接個々のシャードに対応付けられる。リストベースのシャーディングの場合、各シャードがシャードキー値のリストに対応付けられる。

【００２０】

ある実施形態に従うと、データベースは１つ以上のスーパーシャードに対応付けることもできる。スーパーシャードをデータベースに対応付けることにより、データベースに格納されている記録にはさらに制約が加えられる。たとえば、特定のデータベーステーブル内において、ある位置がシャードグループ識別子であることが特定された場合は、シャーディング方法を用いて、特定の顧客向けの記録が格納されているデータセンターが、確実にこの顧客が指定した位置に最も近くなるようにすることができる。

【００２１】

ある実施形態に従うと、再シャーディングは、シャードされたデータベースのシャード全体にわたってデータを再分配するプロセスである。再シャーディングが必要な状況はいくつかあり、たとえば、シャードされたデータベースにシャードを追加する場合、またはシャードされたデータベースからシャードを削除する場合であり、そうすることで、シャード全体におけるデータまたは作業負荷分布におけるスキューをなくす、または、たとえば特定のデータは一緒に格納しなければならない等のアプリケーション条件を満たす。

【００２２】

ある実施形態に従うと、グローバルデータサービス（global data service：ＧＤＳ）コンポーネントを用いて、マルチデータベース環境で使用するためのスケーラビリティ、可用性、および管理性フレームワークを提供することができる。ＧＤＳは、１つ以上のグローバルサービスマネージャ（global service manager：ＧＳＭ（登録商標））リスナーとともに動作することにより、フェイルオーバ、ロードバランシング、およびデータベースサービスの中央管理のサポートを含む、単一の論理データベースとしてのマルチデータベース構成を、クライアントに提示することができる。たとえば、クライアント要求を、可用性、負荷、ネットワークレイテンシ、複製ラグ、またはその他のパラメータに基づいて適切なデータベースにルーティングすることができる。ＧＤＳプールは、グローバルサービスを提供する一組の複製されたデータベースを与えるので、たとえば、ＧＤＳプール内のデータベースを、異なる領域それぞれの複数のデータセンターに配置することができる。シャードされたＧＤＳプールは、シャードされたデータベースのシャードをそれらの複製とともに含む。データベースクライアントの側からすると、シャードされたＧＤＳプールは、１つのシャードされたデータベースのように見える。

【００２３】

１．シャードされたデータベースへのダイレクトアクセス

ある実施形態に従い、シャードされたデータベースへのダイレクトアクセスを提供するためのシステムおよび方法を説明する。シャードディレクタまたはリスナーコンポーネントは、データベースシャードへの、ソフトウェアクライアントアプリケーションによるアクセスを、提供するように動作する。接続プール（たとえばユニバーサル接続プール、ＵＣＰ）およびデータベースドライバ（たとえばＪａｖａ（登録商標）データベースコネク

ティビティ、ＪＤＢＣ、コンポーネント）を、クライアントアプリケーションがシャードキーを接続チェックアウト中にまたはその後の時点で提供することを可能にするように、かつ、クライアントアプリケーションによって指定されたシャードキーを認識するように、かつ、特定のシャードまたはチャンクへのクライアントアプリケーションの接続を可能にするように、構成することができる。この手法により、接続リソースを効率的に再使用することができ、適切なシャードにより速くアクセスすることができる。

【００２４】

図１は、ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのシステムを示す。

【００２５】

図１に示されるように、物理コンピュータリソース（たとえばプロセッサ／ＣＰＵ、メモリ、ネットワーク）１１１とデータベースドライバ（たとえばＪＤＢＣコンポーネント）１１２とを含むアプリケーションサーバまたはデータベース環境１１０のある実施形態に従うと、シャードされたデータベース１２０に接続するシャード認識クライアントアプリケーションは、接続プールロジック１５０が対応付けられている接続プールコンポーネント１６０（たとえばＵＣＰ）を用いることにより、シャードプール内のシャードされたデータベースのさまざまなシャードまたはチャンクへの接続を格納するまたはこの接続にアクセスすることができる。

【００２６】

図１に示されるこの典型的な環境において、シャードされたデータベースは、第１のデータベース領域Ａ（ここではＤＢイースト、ＤＢＥと示す）１３０を含み得る。第１のデータベース領域Ａ　１３０は、チャンクＡ１、Ａ２、…Ａｎとして格納されているシャードＡを有するシャードされたデータベースインスタンス「ＤＢＥ１」１３２と、チャンクＢ１、Ｂ２、…Ｂｎとして格納されているシャードＢを有する「ＤＢＥ２」１３４とを含む。

【００２７】

図１にさらに示されているように、第２のデータベース領域Ｂ（ここではＤＢウェスト、ＤＢＷと示す）１４０は、チャンクＣ１、Ｃ２、…Ｃｎとして格納されているシャードＣを有するシャードされたデータベースインスタンス「ＤＢＷ１」１４２と、チャンクＤ１、Ｄ２、…Ｄｎとして格納されているシャードＤを有する「ＤＢＷ２」１４４とを含む。

【００２８】

ある実施形態に従うと、シャードされたデータベースインスタンスの各データベース領域またはグループは、データベースシャードへのソフトウェアクライアントアプリケーションによるアクセスを提供するように動作するシャードディレクタまたはリスナーコンポーネント（たとえばＧＳＭリスナーまたは別の種類のリスナー）に対応付けることができる。たとえば、シャードディレクタまたはリスナー１３８を第１のデータベース領域Ａに対応付けることができ、別のシャードディレクタまたはリスナー１４８を第２のデータベース領域Ｂに対応付けることができる。

【００２９】

ある実施形態に従うと、クライアントアプリケーションは、接続要求中に、１つ以上のシャードキーを接続プール（たとえばＵＣＰ）に与えることができ、この１つ以上のシャードキーに基づいて、接続プールは接続要求を正しいまたは適切なシャードにルーティングすることができる。

【００３０】

ある実施形態に従うと、接続プールは、複数の使用中接続１６２およびアイドル接続１６４を維持する。接続プールは、特定のシャードまたはチャンクへの接続をそのシャードキーによって識別することができ、同じシャードキーの要求をクライアントから受けたときは接続を再使用することを許可する。

【００３１】

たとえば、図１に示されるように、チャンクＡ１への接続を用いてこのチャンクに接続することができる１７４。特定のシャードまたはチャンクへの、利用できる接続が、プールにない場合、システムは、別のシャードまたはチャンクへの、利用できる既存の接続を、その用途を変更して再使用することを試みることができる。データベース内のシャードまたはチャンク全体にわたるデータ分布をユーザから見えるようにしてもよく、そうすると、ユーザに与えるチャンクの再シャーディングの影響は最小になる。

【００３２】

図２は、ある実施形態に従うシャードされたデータベースへのダイレクトアクセスを可能にするためのシステムをさらに示す。

【００３３】

図２に示されるように、接続要求に関連して、シャード認識クライアントアプリケーション１８０が１つ以上のシャードキー１８２を接続プール（たとえばＵＣＰ）に与えたときに、接続プールまたはデータベースドライバがすでにこのシャードキーに対するマッピングを有している場合、上記接続要求は、適切なシャードおよびチャンク、この例ではチャンクＣ２に、直接転送することができる１８４。

【００３４】

図３は、ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするシステムをさらに示す。

【００３５】

図３に示されるように、シャード認識クライアントアプリケーションが接続要求に関連してシャードキーを与えない場合、または、接続プール（たとえばＵＣＰ）もしくはデータベースドライバ（たとえばＪＤＢＣ）が、提供されたシャードキーに対するマッピングを有していない場合は、接続要求を、この例では第２のデータベース領域Ｂに対応付けられたシャードディレクタまたはリスナーを含む、適切なシャードディレクタまたはリスナー（たとえばＧＤＳ／ＧＳＭリスナー）に転送することができる１８６。

【００３６】

図４は、ある実施形態に従う、シャードされたデータベースへのダイレクトアクセスを可能にするためのプロセスを示す。

【００３７】

図４に示されるように、ステップ１９２において、複数のシャードを有し１つ以上のデータベースドライバと１つ以上の接続プールとに対応付けられているデータベースが与えられ、上記データベースドライバおよび接続プールはともに、データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供する。

【００３８】

ステップ１９４において、データベースドライバまたは接続プールのうちの１つ以上は、データベースへの接続のチェックアウト中にまたはその後の時点でクライアントアプリケーションがシャードキー情報を提供することを可能にするように構成され、次にこのシャードキー情報を用いて、適切なデータベースのシャードへのクライアントアプリケーションによるアクセスを提供する。

【００３９】

ステップ１９６において、データベースドライバおよび／または接続プールは、クライアントアプリケーションによって指定されたシャードキーを認識し、クライアントアプリケーションがこのクライアントアプリケーションに対応付けられた特定のシャードまたはチャンクに接続することを可能にする。

【００４０】

ステップ１９８において、接続プールは、接続をそのシャードキーによって識別し、同じシャードキーの要求をクライアントアプリケーションから受けたときは接続を再使用することを許可することができる。

【００４１】

接続要求のためのシャードキーの構築

ある実施形態に従うと、シャード認識クライアントアプリケーションは、たとえば、複合シャードキーを異なるデータ型で生成できるようにするShardKeyまたは同様のインターフェイスおよびビルダを用いて、シャードされたデータベースへの接続をフェッチするのに必要な、シャードキーおよび任意でシャードグループを、識別し構築することができる。

【００４２】

【数２】

```
subkey( Object subkey, java.sql.SQLTYPE subkeyDataType)
```

【００４３】

ある実施形態に従うと、複数の呼出しをShardKeyビルダ上のsubkey(...)方法に対して行なうことにより、複合シャードキーを構築することができ、この場合の各subkeyは、データ型が異なっていてもよい。データ型は、以下のように、列挙子oracle.jdbc.OracleTypeを用いて、たとえばストリングおよびデート複合シャードキーとして、定義することができる。

【００４４】

【数３】

```
ShardKey shardKey = datasource
       . createShardKeyBuilder()
       . subkey( <string>, oracle.jdbc.OracleType.VARCHAR2)
       . subkey (<date>, oracle.jdbc.OracleType.DATE )
       . build();
```

【００４５】

ある実施形態に従うと、データ型の選択セットはキーとしてサポートすることができ、対応するバリデーションがビルダに与えられてサポートされていないデータ型を防止する。典型的なデータ型は、OracleType.VARCHAR2、OracleType.CHAR、OracleType.NVARCHAR、OracleType.NCHAR、OracleType.NUMBER、OracleType.FLOAT、OracleType.DATE、OracleType.TIMESTAMP、OracleType.TIMESTAMP WITH LOCAL TIME ZONE、およびOracleType.RAWを含み得る。

【００４６】

シャードキー値による接続ストリングの更新

ある実施形態に従うと、１つ以上のシャードキーを含む接続要求に対してリスナーを用いて接続を生成する場合、ShardKeyインターフェイスは、シャードキーを対応するBASE64符号化ストリングに変換して、接続データが、データベースドライバに関連する以下の２つのフィールドを有するようにする。

【００４７】

【数４】

```
SHARD_KEY_B64 for base64 encoded binary representation of shard key
GROUP_KEY_B64 for base64 encoded binary representation of group key
```

【００４８】

base64符号化値（*_B64）のフィールドは、以下のフォーマットを有し得る。

【００４９】

【数５】

```
...(CONNECT_DATA=(SHARD_KEY_B64=[version] [type] [int literal]
[int literal] ... ,[base64 binary],[base64 binary],
[base64 binary],...))...
```

【００５０】

ある実施形態に従うと、接続ストリングは、「バージョン」番号＝１を有するヘッダで始まり、その値を表１に示すように定義できる「タイプ」がその後に続く。

【００５１】
【表１】

0	ストリングはハッシュ値を含まない。	符号値はAL32UTF8（varcharnの場合）およびAL16UTF16（nvarcharの場合）符号化において符号化される。
1	ストリングはハッシュ値を含む。	
2	ストリングはハッシュ値を含まない。	符号値はデータベース符号化（列ごとに特定のものであってもよい）において符号化される。
3	ストリングはハッシュ値を含む。	
4	ストリングはハッシュ値のみを含む。	

表１

【００５２】
ある実施形態に従うと、表２に示されるように、ストリングタイプの後に、スペースで分離された値「タイプ識別子」が与えられる（１０進整数定数として）。
【００５３】
【表２】

1	VARCHAR, NVARCHAR, [CHAR, NCHAR = 96]
2	NUMBER
12	DATE
23	RAW
180	TIMESTAMP
231	TIMESTAMP WITH LOCAL TIME ZONE

表２

【００５４】
ある実施形態に従うと、ヘッダはカンマで終わり、その後に、複合シャードキーの各部分についてカンマで分離されたbase64符号化値のリストが続き、データ型は次のように符号化される。
【００５５】
NUMBER、FLOAT、DATE、TIMESTAMP、TIMESTAMP WITH LOCAL TIMEZONE：これらのデータ型について、対応するOracle表現を用いて、これらを、B64符号化の前に、対応するバイトアレイに変換することができる。
【００５６】
RAW、VARCHAR、CHAR：これらのデータ型について、ユーザからの入力ストリングに対し、クライアントはB64符号化前のAL32UTF8符号を用いることができる。
【００５７】
NVARCHAR、NCHAR：これらのデータ型について、AL16UTF16符号化が推奨される。
たとえば、キー（「US」、「94002」）は次のように符号化できる。
【００５８】
【数６】

```
...(CONNECT_DATA=(SHARD_KEY_B64=1 1 2,VVM=,OTQwMDI=))...
```

【００５９】
既知のシャードキーを用いた接続プールからの接続チェックアウト

ある実施形態に従うと、接続を接続プールから借りるとき、シャード認識クライアントアプリケーションは、たとえばOracleDataSourceおよびPoolDataSourceを与えることができる上記接続ビルダを用いて、シャードキーおよびシャードグループキーを提供することができる。

【００６０】

【数７】

```
Connection conn =  dataSource
       . createConnectionBuilder()
       . shardkey(<shard_key>)  //of type ShardKey
       . shardGroupKey(<shard_group_key>)
       . build()
```

【００６１】

ある実施形態に従うと、上記ＡＰＩは、プールから生成したまたは借りた接続が正しいシャードおよびチャンクに接続されることを保証する。これは以下を含む。

【００６２】

（１）この接続に関するすべての動作は、接続チェックアウト中に与えられたキーによって指定されたシャードおよびチャンクに限定される。さもなければ、エラーまたは例外がアプリケーションに戻される。レース条件も、接続使用中の例外につながる。

【００６３】

（２）このデータソースに対して指定されるＵＲＬは、要求に対してシャードキーを明示的に指定するgetConnection　ＡＰＩを用いるときに、既にシャードキーを含んでいてはならない。さもなければ、例外がユーザに戻される。

【００６４】

既存のまたはチェックアウトされた接続に対するシャードキーの設定

ある実施形態に従うと、setShardKeyまたは同様のインターフェイスにより、接続に対してシャードキーを設定することができる。接続のチェックアウト時にアプリケーションが１つ以上のシャードキーを与えることができていない場合、接続プールおよびデータベースドライバ（たとえばＵＣＰ／ＪＤＢＣ）は、以下のように、たとえばOracleConnectionクラスにおいて、接続レベルでＡＰＩの使用をサポートすることもできる。

【００６５】

【数８】

```
connection.setShardKey(<shard_key>,<shard_group_key>);
```

【００６６】

ある実施形態に従うと、シャードキーおよびシャードグループキーは、ShardKeyタイプである。これが接続について呼出されたときは、以下のうちの１つが起こり得る。

【００６７】

（１）setShardKeyに与えられた１つ以上のシャードキーが、接続が最初に生成されたシャードおよびチャンクと一致する。この場合、他のアクションは不要であり、接続はそのまま使用することができる。

【００６８】

（２）setShardKey　ＡＰＩによって指定されたシャードキーは、異なる「チャンク」にマッピングされるが、接続が最初に生成された同じシャード上にある。この場合、接続を正しいチャンクに切換えることが必要である。

【００６９】

（３）setShardKeyによって指定されたシャードキーが、接続が既に生成されているシャードと異なるシャードである場合がある。よって、接続がさらに使用される可能性が生じる前に、正しいシャードへの、基礎となる物理接続の切換えが必要である。

【００７０】

ある実施形態に従うと、プールなしでデータベースドライバ（たとえばＪＤＢＣ）が直

接使用されるときに後者のシナリオが発生した場合は、エラーが、正しくないシャードキーを示すアプリケーションに戻される。接続は、依然として使用可能であり、setShardKeyが呼出される前に接続されていた元のチャンクに対する動作を実行するために使用できる。setShardKeyが成功すれば、この接続に対するすべての動作は、一旦シャードキーが設定されれば、setShardKey中に与えられたキーによって指定されたシャードおよびチャンクに限定される。また、アプリケーションは、setShardKeyが呼出されたとき、接続には進行中のトランザクションがないことを保証しなければならない。そうでなければ、setShardKeyが完了しなかったことを示す例外がアプリケーションに戻される。

【００７１】

ある実施形態に従うと、（３）の場合のように切換えが差し迫って必要である場合、または（２）の場合のようにサービス切換えを行なう必要がある場合は、接続に対する、すべてのオープンな結果セット、ステートメント、大きなオブジェクト、ストリーム等を、クローズしなければならない。

【００７２】

シャードキーを与えない接続チェックアウト

ある実施形態に従うと、接続を用いて単一シャードクエリを実行する前に、接続プールまたはデータベースドライバ（たとえばＵＣＰ／ＪＤＢＣ）からのチェックアウト時にアプリケーションがシャードキーを与えないときは、クエリは、このようなクエリの実行を容易にし結果をアプリケーションに返すコーディネータシャードに導かれる。

【００７３】

ある実施形態に従うと、接続がリスナーを通して生成されると、接続プール（たとえばＵＣＰ）は、チャンクに対応する実際の「チャンク名」、および、接続が生成された「インスタンス／シャード名」を抽出しこの情報を用いてプール内の接続を識別しようと試みる。

【００７４】

チャンク名およびインスタンス／シャード名情報は、接続生成後、コンテキスト（SYS_CONTEXT）に存在し、どの時点でもその接続に対する「チャンク切換え」動作があれば、更新することができる。チャンク情報は、プール内の接続オブジェクトの一部にすることもでき、チェックアウトにおいてユーザに戻すべき最良の候補接続のルックアップ中に使用することができるとともに、（たとえばOracleの高速アプリケーション通知（Fast Application Notification）、ＦＡＮからの、または、ランタイム接続ロードバランシング（Runtime Connection Load Balancing）、ＲＬＢ環境からの）システムステータスまたは通知イベントに応答して接続処理のために使用することができる。

【００７５】

接続プール内の接続借用中の接続選択

ある実施形態に従うと、接続プール（たとえばＵＣＰ）からのすべての接続チェックアウト要求について、接続プールは、プール内の各接続について格納されているチャンク情報を用いて、以下の方法のうちのいずれかで、ユーザに戻すべき最良の可能なマッチング接続を決定することができる。

【００７６】

（１）接続要求にシャードキーが含まれていれば、接続プールは、プールによってそれまでに発見されていたシャードトポロジ内のキーの一致するチャンク名を探そうと試みることができる。一致するチャンク名が見出されなければ、接続要求は、生成すべき新たな接続のために、リスナーに転送される。

【００７７】

（２）さもなければ、チャンク名を用いて、チャンクが存在するインスタンスのリストを、接続選択アルゴリズムを用いてフェッチすることができ、これらのインスタンスのうちの１つに対する接続が選択される。これにより、シャードキーに対応するチャンクがユーザによって要求されたインスタンスへの接続が生成されることを保証する。

【００７８】

（３）接続が選択されると、現在のセッションで使用すべきチャンク名およびシャードキーを指定する接続に対し、チャンク切換えピギーバックがなされる。一致する接続が発見されない場合は、要求をリスナーに転送することによって新たな接続が生成される。

【００７９】

フェイルオーバ／再シャーディングイベントの処理

ある実施形態に従うと、接続プール（たとえばＵＣＰ）は、たとえば以下のようなサブスクリプションストリングを用いてサービスごとにフェイルオーバイベントにサブスクライブすることができる。

【００８０】

【数９】

```
("eventType=database/event/service/<service_name>")
```

【００８１】

シャーディングサポートのために、OracleDBTopologyおよび接続プール（たとえばＵＣＰ）は、チャンクの移動または分割に対応するイベントの受信タイプにサブスクライブすることができる。チャンクレベルイベントサブスクリプションストリングは、たとえば以下の通りであってもよい。

【００８２】

【数１０】

```
("eventType=database/event/service/chunk")
```

【００８３】

チャンク名はこのイベント本体の一部であってもよい。特定のチャンクに対して第１の接続がなされたとき、対応するチャンク名がsys_contextから読み出され、チャンクイベントサブスクリプションを生成してすべてのチャンク関連イベントを受けることができる。ある実施形態に従うと、チャンクサブスクリプションはサービスベースではない。

【００８４】

ある実施形態に従うと、サーバ側におけるいかなる再シャーディング作業またはチャンク分割も、たとえば次のように、イベント本体において指定されたチャンク名を有する対応するチャンク・ダウン通知イベントを引起す。

【００８５】

【数１１】

```
VERSION=1.0 event_type=CHUNK chunk=<chunk name>
instance=<instance name> host=<host> database=<db name>
db_domain=<db domain name> status=<UP|DOWN>
timestamp=<timestamp> timezone=<timezone>
```

【００８６】

ある実施形態に従うと、接続プール（たとえばＵＣＰ）は、そのシャーディングトポロジを更新して最近のチャンクイベントと同期させることができる。

【００８７】

（１）接続プールは、対応する「インスタンス」情報を、チャンク配置情報（すなわちチャンクの、それが存在するインスタンスのリストへのマッピング）から削除する。

【００８８】

（２）チャンクに関するシャードキー情報はトポロジから削除しない。

（３）チャンクが既知のインスタンスに現われた場合、そのチャンクに関するインスタンス情報が、チャンク配置テーブル、および、そのチャンクに関するシャードキー情報を用いて更新されたシャードキートポロジキャッシュに、もしそれが変更されていた場合は、追加される。

【００８９】

（４）チャンクが未知のインスタンスに現われた場合は、インスタンス情報がＵＣＰトポロジおよびチャンク配置データに追加され、このインスタンスに関する全シャードキー

トポロジがフェッチされ接続プールのシャードキートポロジキャッシュに配置される。

【００９０】

（５）チャンク・アップイベントは接続プールによって処理されない。

ある実施形態に従うと、シャードされたデータベース内のチャンクを分割する必要がある場合、サーバは、以下のように見えるチャンク分割イベント（chunk Split events）を送信する。

【００９１】

【数１２】

```
 VERSION=1.0 event_type=CHUNK chunk=<chunk name>
instance=<instance name> host=<host> database=<db name>
db_domain=<db domain name> status=SPLIT timestamp=<timestamp>
timezone=<timezone> newchunk=<new chunkname>
 [hash=<split boundary hash value>]
```

【００９２】

ある実施形態に従うと、ハッシュは、境界が第１分割チャンクの高い値および第２分割チャンクの低い値に対応する、自動ハッシュベースシャーディングの場合に限り、適用できる。接続プール（たとえばＵＣＰ）は以下のようにチャンク分割イベントを処理する必要がある。

【００９３】

（１）分割を受けたインスタンス名をチャンク配置データから削除する。

（２）チャンク名を次に新バージョンで更新する。なぜなら、このときチャンクは分割を受けており、分割後の新たなチャンクはチャンクがまだ分割されていないレプリカ上の古いチャンクから区別する必要があるからである。

【００９４】

（３）シャードトポロジテーブルおよびチャンク配置テーブル双方が、他のすべてのデータはそのままで新たなチャンク名で更新される。これにより、まだ分割を受けていないレプリカが分割の影響を受けていない状態のままで引続き以前のように接続要求を扱うことを保証する。

【００９５】

（４）要求を扱う既存の接続がなく新たな接続が生成されたときに、接続が新たに分割されたチャンクに対して生成された場合は、新たなチャンク名およびその範囲をトポロジキャッシュに入れ、この時点で、分割／アップ分割チャンクいずれも、キー範囲についての接続要求の処理において利用できる。

【００９６】

（５）このプロセスは、すべてのレプリカが分割を受けるまで続き、この場合、古いチャンク名（更新されたバージョン）には最早何のインスタンスもなく、これは次に、チャンク配置テーブルおよびシャードキートポロジテーブル双方から削除される。

【００９７】

（６）自動シャーディングの場合、チャンク分割を受けた後のステップ（１）で、ハッシュ値によって指定された新たな範囲を有する新たなチャンク情報が、シャードキートポロジテーブルにおいて更新され、また、新たなチャンク配置情報が更新される。

【００９８】

シャードされたプールに対するランタイム接続ロードバランシング（ＲＬＢ）要求

ある実施形態に従うと、シャードされたデータベースへの接続を有するプールについて、ＲＬＢサブスクリプションは以下を含み得る。すなわち、接続プール（たとえばＵＣＰ）はＧＤＳクラスタからのＲＬＢ通知に対するグローバルサービスにサブスクライブする。ＧＤＳデータベースからのＲＬＢイベントはグローバルサービスに基づく。よって、グローバルサービスＲＬＢを受けたとき、プールが現在提供している各チャンクの分布パターンをチェックするために、システムは、そのチャンクに適用できるＲＬＢ％を得た後に

、接続分布を試みることができる。

【００９９】

シャードされたデータベースに対するチャンク配置データ

ある実施形態に従うと、シャードメタデータ情報が生成されて接続プール（たとえばＵＣＰ）内で保持され、これは、サービスを利用できるシャードインスタンスへのチャンク名のマッピング、および、このインスタンスに対するチャンクの優先度を有する。ある実施形態に従うと、内部において、生成されたデータ構造は、シャードされたデータベースに関する表３に示されているように見える。

【０１００】

【表３】

チャンク名	シャードインスタンスリスト
CHUNK_1_1	[<Instance:dbs1%1, Priority:0>, <Instance:dbs1%2, Priority:0>]
CHUNK_1_10	[<Instance:dbs1%1, Priority:0>, <Instance:dbs1%2, Priority:0>]
CHUNK_1_100	[<Instance:dbs1%1, Priority:0>, <Instance:dbs1%2, Priority:0>]
CHUNK_1_11	[<Instance:dbs1%1, Priority:0>, <Instance:dbs1%2, Priority:0>]

表３

【０１０１】

シャードされたデータベースのシャードキーからチャンク名へのマッピング

ある実施形態に従うと、シャードリストマッピングに対するサービスとは別に、システムは、シャードキー範囲／リストまたは範囲から、これらのシャードキーに対応するチャンクへのマッピングを含み得る。ある実施形態に従うと、シャードグループ分けは、シャーディング方法がハッシュベースに設定されるとき、リストまたは範囲ベースであってもよい。他のすべてのケースにおいて、シャードグループはサポートされない。以下のケースは、トポロジデータの例として可能であり、ハッシュベースのシャーディングおよびリストベースのスーパーシャーディングに関して以下を含む（表４）。

【０１０２】

【表４】

<グループキーリスト、シャードキーハッシュ範囲>	チャンク名
[gold, silver] , {Low:0, High:42949672}	CHUNK_2_1
[gold, silver] , {Low:42949672, High:85899344}	CHUNK_2_2
[regular, bronze] , {Low:0, High:42949672}	CHUNK_1_100

表４

【０１０３】

ハッシュベースのシャーディングおよび範囲ベースのスーパーシャーディングについては次の通りである（表５）。

【０１０４】

【表５】

<グループキー範囲、シャードキーハッシュ範囲>	チャンク名
{1-10} , {Low:0, High:42949672}	CHUNK_2_1
{1-10} , {Low:42949672, High:85899344}	CHUNK_2_2
[10-MAX] , {Low:0, High:42949672}	CHUNK_1_100

表５

【０１０５】
シャードグループ分けなしの範囲ベースのシャーディングについては次の通りである（表６）。

【０１０６】
【表６】

シャードキーハッシュ範囲	チャンク名
{Low:0, High:42949672}	CHUNK_2_1
{Low:42949672, High:85899344}	CHUNK_2_2
{Low:85899344, High: MAX}	CHUNK_1_100

表６

【０１０７】
シャードグループ分けなしのリストベースのシャーディングについては次の通りである（表７）。

【０１０８】
【表７】

シャードキーリスト	チャンク名
[IN,AU]	CHUNK_2_1
[US,CAN]	CHUNK_2_2
[DE,UK]	CHUNK_1_100

表７

【０１０９】
シャードメタデータキャッシュ
ある実施形態に従うと、接続プールは、シャードされたデータベースの異なるチャンクおよびシャードへの新たな接続がこのプールから生成されたときに学習され収集された情報を含む、シャードメタデータキャッシュを維持する。たとえば次の通りである。

【０１１０】
（１）接続プール（たとえばＵＣＰ）が接続を生成したチャンクに適用可能なシャードキーおよびシャードグループキー範囲
（２）適用可能であればデータベース符号化等のシャードキー列情報
（３）シャーディングおよびシャードグループ分け方法（たとえばハッシュベース、リストベース、範囲ベース）
接続生成に対するトポロジの構築
ある実施形態に従うと、新たな接続がシャードを用いて生成されたとき、LOCAL_CHUNKテーブルからのデータならびにシャーディングタイプおよびデータベース符号化タイプ等のシャーディングメタデータが、LOCAL_CHUNK_TYPESテーブルから抽出され、クライアン

ト側のメタデータキャッシュに格納される。インスタンスに対する各チャンクのチャンク優先度も読み出されてクライアント側に格納される。

【０１１１】

サーバ側のオラクル通知サービス（Oracle Notification Service、ＯＮＳ）イベントに基づく更新トポロジ

ある実施形態に従うと、メタデータキャッシュを、以下のデータベース高可用性（high-availability、ＨＡ）イベントすべてを処理しかつクライアント側の対応するデータ構造を更新することにより、サーバ側の変化と同期させ続けることができる。

【０１１２】

（１）グローバルサービスメンバダウンイベントまたはインスタンスダウンイベントの場合、対応するインスタンスを、影響されたチャンクのインスタンスリストから削除する。チャンクが他のインスタンスになければ、対応するチャンクデータをトポロジに格納されているチャンクメタデータから削除する必要がある。

【０１１３】

（２）グローバルサービスダウンイベントの場合、サービスおよびチャンク関連データ双方を削除する必要がある。

【０１１４】

（３）チャンクダウンイベントの場合、トポロジに格納されているチャンク情報内の対応する影響を受けたエントリを削除する必要がある。

【０１１５】

シャードキールックアップ

ある実施形態に従うと、内部方法をメタデータキャッシュに与えることができる。これは、フォーマットがチャンクであり（ShardKey shardKey、ShardKey groupKey）、この方法に対してパラメータとして与えられた１つ以上のシャードキーに対応するチャンク名を得るために使用される。ShardKeyオブジェクトは、２つのシャードキーの比較および同等性をチェックできる方法を含み得る。

【０１１６】

セキュリティ

ある実施形態に従うと、シャードされたデータベース内のさまざまなチャンクのコンテンツ（シャードキーおよび範囲）および位置の露出に関するセキュリティの問題に対処するために、システムは、LOCAL_CHUNK_TYPE、LOCAL_CHUNK_COLUMNS、LOCAL_CHUNKS等のテーブル内のデータへのユーザアクセスを可能にするためにサーバ上で定義された特定の「パスワードで保護されたロール」の使用をサポートすることができる。ＵＣＰもしくはＪＤＢＣを用いまたはカスタム接続プールを構築し、シャードされたデータベースへの接続（またはプール）の維持を希望し、かつシャードされたデータベースのクライアント側トポロジスナップショットを構築することを望む、データベースユーザは、これらのためにプロビジョニングされたこのロールを有しなければならない可能性がある。

【０１１７】

２．キャッシュおよびシャードトポロジを用いた、シャードされたデータベースへのアクセス

ある実施形態に従うと、システムは、キャッシュおよびシャードトポロジを用いて、シャードされたデータベースへのアクセスを可能にする。新たな接続が生成されると、シャードトポロジレイヤをデータベースドライバレイヤに構築することができ、シャードトポロジレイヤは、シャードキー範囲を学習してシャードの位置にキャッシュする。シャードトポロジレイヤにより、クライアントアプリケーションからの、後続の接続要求は、シャードディレクタまたはリスナーをバイパスすることができ、代わりに、適切なシャードまたはチャンクへの高速キー経路アクセスを用いることができる。図５は、ある実施形態に従う、キャッシュおよびシャードトポロジを用いて、シャードされたデータベースへのアクセスを可能にするためのシステムを示す。

【０１１８】

図５に示されるように、ある実施形態に従うと、前述のように、シャードされたデータベースに接続するシャード認識クライアントアプリケーションは、接続プール（たとえばＵＣＰ）を用いて、シャードされたプール内のシャードされたデータベースのさまざまなシャードまたはチャンクへの接続を格納するまたはこの接続にアクセスすることができる。

【０１１９】

これも前述のように、ある実施形態に従うと、シャードされたデータベースは、チャンクＡ１、Ａ２、…Ａｎとして格納されているシャードＡを有するシャードされたデータベースインスタンス「ＤＢＥ１」と、チャンクＢ１、Ｂ２、…Ｂｎとして格納されているシャードＢを有する「ＤＢＥ２」とを含む第１のデータベース領域Ａ（ＤＢＥ）と、チャンクＣ１、Ｃ２、…Ｃｎとして格納されているシャードＣを有するシャードされたデータベースインスタンス「ＤＢＷ１」と、チャンクＤ１、Ｄ２、…Ｄｎとして格納されているシャードＤを有する「ＤＢＷ２」とを含む第２のデータベース領域Ｂ（ＤＢＷ）とを含み得る。

【０１２０】

またこれも前述のように、ある実施形態に従うと、シャードされたデータベースインスタンスの各データベース領域またはグループを、シャードディレクタまたはリスナーコンポーネント（たとえばＧＳＭ）と対応付けることができる。アプリケーションは、接続要求中にシャードキーをＵＣＰに与えることができ、シャードキーに基づいて、接続プールはこの接続要求を正しいシャードまでルーティングすることができる。

【０１２１】

ある実施形態に従うと、新たな接続が生成されると、シャードトポロジレイヤ２１０をデータベースドライバ（たとえばＪＤＢＣ）レイヤに構築することができ、シャードトポロジレイヤは、シャードキー範囲を学習して各シャードの位置にキャッシュする。後続の接続要求は、シャードディレクタまたはリスナーをバイパスすることができ、代わりに、適切なシャードまたはチャンクへの高速キー経路アクセスを用いることができる。

【０１２２】

図６はさらに、ある実施形態に従う、キャッシュおよびシャードトポロジを用いて、シャードされたデータベースへのアクセスを可能にするためのシステムを示す。

【０１２３】

図６に示されるように、シャード認識クライアントアプリケーションは、シャードキーおよびスーパーシャードキーを用いて（たとえばシャードキーＸを用いて）接続要求を生成することができる。シャードトポロジレイヤは、シャード／チャンクへの高速キー経路アクセスを可能にし２１４、これは、シャードキー範囲を各シャードの位置にキャッシュすることを含む。

【０１２４】

図７はさらに、ある実施形態に従う、キャッシュおよびシャードトポロジを用いて、シャードされたデータベースへのアクセスを可能にするシステムを示す。

【０１２５】

図７に示されるように、この例において、シャードトポロジを用いて、シャードディレクタまたはリスナーをバイパスすることにより、シャード認識クライアントアプリケーションによる、この例ではチャンクＣ２への高速キー経路アクセスを、与えることができる。

【０１２６】

GetConnection　ＡＰＩ設計

ある実施形態に従うと、OracleDatasourceにおいて定義されている典型的なＡＰＩは以下を含み得る。

【０１２７】

【数１３】

```
datasource. getConnectionToShard(<shard_key>)
```

【０１２８】

これは、データベース内に１レベルのシャーディングがありかつスーパーシャードキーが関与していないときに、用いることができる。接続は、データソースに設定されているデフォルトユーザ名およびパスワードを用いて取得することができる。シャードキーは必要なパラメータであり、ＡＰＩをナル（null）または間違ったシャードキーとともに用いると、アプリケーションに戻される例外が生じる。

【０１２９】

【数１４】

```
datasource. getConnectionToShard (<shard_key>, <super_shard_key>,
<user name>, <password>)
```

【０１３０】

これは、特定セットのシャードキーまたは特定のユーザに対して接続をフェッチまたは生成する必要があるときに用いることができる。シャードキーは必要なパラメータであり、ＡＰＩをナルまたは間違ったシャードキーとともに用いると、アプリケーションに戻される例外が生じる。スーパーシャードキーはナルの可能性がある。

【０１３１】

【数１５】

```
datasource. getConnectionToShard (<shard_key>, <super_shard_key>,
<user name>, <password>, <label_properties>)
```

【０１３２】

これは、ユーザ定義のラベルにも一致する、特定セットのシャードキーまたは特定のユーザに対して接続をフェッチまたは生成する必要があるときに用いることができる。シャードキーは必要なパラメータであり、ＡＰＩをナルまたは間違ったシャードキーとともに用いると、アプリケーションに戻される例外が生じる。スーパーシャードキーはナルの可能性がある。

【０１３３】

シャードキーを用いた接続の生成および取出し

図８は、ある実施形態に従う、接続プールを用いて、シャードされたデータベースへの接続のプールを生成し維持するときの、フローを説明する、シーケンス図を示す。

【０１３４】

図８に示されるように、ある実施形態に従うと、接続プール（たとえばＵＣＰ）を用いて、シャードされたデータベースへの接続のプールを生成し維持することができるとともに、PoolDatasourceに対して生成されたgetConnectionToShard　ＡＰＩコールの使用をサポートすることができる。たとえば、示されているように、クライアントアプリケーション２３０は、接続プール２３２（たとえばＵＣＰ）と、データベースドライバ２３４（たとえばＪＤＢＣ）と、データベーストポロジモジュール２３６とを、シャードディレクタまたはリスナー２３８とともに用いて、シャードされたデータベース２４０にアクセスすることができるとともに、シャードドポロジレイヤの使用をサポートすることができる。クライアントアプリケーションが、任意で１つ以上のシャードキーを用いて、接続獲得要求（get connection request）２５０を行なったとき、接続プールは、サービストポロジ獲得要求（get service topology request）２５２を、データベーストポロジモジュールに対して行ない、これはインスタンスリスト２５４を返すことができる。

【０１３５】

ある実施形態に従うと、インスタンスリストが空の場合、接続プールは接続獲得要求をデータベースドライバに対して行なうことができ２６０、これは、シャードディレクタま

たはリスナーを呼出して接続を生成する２６２。データベースへの接続が生成されると２６４、接続は、クライアントアプリケーションに返され２６６、２６８、２７０、２７２、それに応じてシャードトポロジが更新される２７４。

【０１３６】

代わりに、インスタンスリストが空でない場合、接続プールは、インスタンスのマッチングのためにプール内の利用できる接続に対してルックアップを行ない２８０、適切な接続をクライアントアプリケーションに返すことができる２９０。

【０１３７】

ある実施形態に従うと、インスタンスリストが、発見された利用できる接続の有効なリストを有する場合、接続プールは、既存の接続を、その用途を変更して再使用することができ２９２、仮想サービスをこの接続に切換えることができる２９４。用途が変更された接続は、クライアントアプリケーションに返すことができる２９６、２９８、３００。

【０１３８】

代わりに、インスタンスリストが一致する接続を有していない場合、接続プールは、上記のように、接続獲得要求をデータベースドライバに対して行なうことができ３１０、これは、シャードディレクタまたはリスナーを呼出して接続を生成する３１２。データベースへの接続が生成されると３１４、この接続はクライアントアプリケーションに返され３１６、３１８、３２０、３２２、それに応じてシャードトポロジは再び更新される３２６。

【０１３９】

図９は、ある実施形態に従う、アプリケーションが接続プールなしでデータベースドライバを用いて、シャードされたデータベースへの接続をフェッチするときの、フローを説明する、シーケンス図を示す。

【０１４０】

図９に示されるように、ある実施形態に従うと、アプリケーションは、ＪＤＢＣドライバ等のデータベースドライバを、接続プールなしで使用することにより、シャードされたデータベースへの接続をフェッチすることができる。接続要求は、シャードディレクタまたはリスナーをバイパスし、適切なシャードまたはチャンクへの高速キー経路アクセスを用いることができる。

【０１４１】

図９に示されるように、クライアントアプリケーションが１つ以上のシャードキーを用いて接続獲得要求を行なうと３３０、データベースドライバは、接続プールからの入力を要することなく、シャードディレクタまたはリスナーを直接呼出して接続を生成することができる３３２。データベースへの接続が生成されると３３４、この接続はクライアントアプリケーションに返され３３６、３３８、３４０、それに応じてシャードトポロジが更新される３４２。

【０１４２】

図１０は、ある実施形態に従う、シャードされたデータベースへのアクセスをキャッシュおよびシャードトポロジを用いて可能にするためのプロセスを示す。

【０１４３】

図１０に示されるように、ステップ３４３において、複数のシャードを有し１つ以上のデータベースドライバと１つ以上の接続プールとに対応付けられたデータベースが与えられ、上記データベースドライバおよび接続プールはともに、データベースに格納されているデータへのクライアントアプリケーションによるアクセスを提供する。

【０１４４】

ステップ３４５において、データベースドライバは、クライアントアプリケーションがシャードキー情報を提供することを可能にするように構成されており、シャードキー情報は、データベースの適切なシャードへのクライアントアプリケーションによるアクセスを提供するために使用され、データベースドライバは、接続要求の処理において使用する、シャードキー範囲を学習してデータベース内のシャードの位置にキャッシュするシャード

トポロジレイヤを含む。

【０１４５】

ステップ３４７において、接続プールは、接続をそのシャードキーによって識別することができ、同じシャードキーの要求をクライアントアプリケーションから受けたときは、接続を再使用することを許可する。

【０１４６】

ステップ３４９において、シャードトポロジレイヤは、後続の接続要求がシャードディレクタまたはリスナーをバイパスして代わりに適切なシャードまたはチャンクへの高速キー経路アクセスを用いることを可能にする。

【０１４７】

接続プール（たとえばＵＣＰ）を通した接続に対するSetShardKey

ある実施形態に従うと、クライアントアプリケーションまたはユーザが、借用した接続に対してsetShardKey(..)を実行しようとするときは、以下の動作が接続に対して実行される。

【０１４８】

（１）シャードキーがシャードトポロジテーブル内の所与の範囲にマッピングされる場合は、このキー範囲に対する仮想サービス名がルックアップされる。仮想サービス名が接続上の名前と同一であれば、同じチャンクへの接続が生成され直接再使用できる。

【０１４９】

（２）シャードキーが、接続が生成されたインスタンスと同一のインスタンス上にある新たな仮想サービスにマッピングされる場合は、この接続上のサービスを新たな仮想サービスに切換えることにより、この接続を再使用できる。接続の仮想サービスを切換える前に、この接続上のすべてのオープンアーティファクトを、ＪＤＢＣ　ＡＰＩを用いてクローズする必要がある。接続に進行中のトランザクションがある場合、動作は失敗し例外がユーザに与えられる。

【０１５０】

（３）シャードキーが、異なるインスタンス上にある新たな仮想サービスにマッピングされる場合、接続プール（たとえばＵＣＰ）は、プロキシ接続の下にある物理接続を、たとえば以下のプロセスによって切換える必要がある。すなわち、アプリケーションに返される接続は、下にある物理接続を表わすプロキシ接続オブジェクトである。接続を新たなシャードインスタンスに切換える必要があるとき、接続プールは先ず、ターゲット仮想サービス名に対して生成されたプール内の既存の接続を発見しようとし、なければ新たな接続を生成する。プロキシによって包まれている元の接続は、すべてのオープンアーティファクトをクローズした後に、プールに返される。進行中のトランザクションがある場合、この動作は結果として例外になる。そうすると、新たな接続が、プロキシオブジェクトの基で、下にある物理接続を変更するsetDelegateと呼ばれる機能を用いて、プロキシ接続オブジェクトに対応付けられる。ユーザは引続きプロキシオブジェクトを使用することができ、これはこのとき、setShardKeyコールにおいて新たなキーによって指定されたシャードおよびチャンクを指している。

【０１５１】

データベースドライバ（たとえばＪＤＢＣ）を通した接続に対するSetShardKey

ある実施形態に従うと、ユーザが、データベースドライバ（たとえばＪＤＢＣ）を用いて、接続にオブジェクトに対してsetShardKeyを実行しようとするときは、以下のプロセスに従うことができる。

【０１５２】

（１）シャードキーがシャードトポロジテーブル内の所与の範囲にマッピングされる場合は、このキー範囲に対する仮想サービス名がルックアップされる。仮想サービス名が接続上の名前と同一であれば、同じチャンクへの接続が生成され直接再使用できる。

【０１５３】

（２）シャードキーが、接続が生成されたインスタンスと同一のインスタンス上にある

新たな仮想サービスにマッピングされる場合は、この接続上のサービスを新たな仮想サービスに切換えるだけで、この接続を再使用できる。接続の仮想サービスを切換える前に、この接続上のすべてのオープンアーティファクトを、データベースドライバ（たとえばＪＤＢＣ）ＡＰＩを用いてクローズする必要がある。接続に進行中のトランザクションがある場合、動作は失敗し例外がユーザに与えられる。

【０１５４】

（３）シャードキーが、異なるインスタンス／シャード上にある新たな仮想サービスにマッピングされる場合は、対応するエラーメッセージがユーザに返される。

【０１５５】

接続プール（たとえばＵＣＰ）における接続選択

ある実施形態に従うと、接続プール（たとえばＵＣＰ）において以下のプロセスを用いて接続を選択することができる。

【０１５６】

（１）プールにおいてシャードキー情報とともにgetConnection(..)要求を受けたときは、シャードキーおよびスーパーシャードキーに対応するサービス名を得るために、接続プールはOracleDBTopologyモジュールが提供するＡＰＩを呼出すことができる。

【０１５７】

【数１６】

```
dbTopology.getServiceName (<shard_key>, <super_shard_key>)
```

【０１５８】

（２）OracleDBTopologyにおいてこのチャンクのためのサービス名がまだ利用できない場合、すなわち、この特定のチャンクへの接続がなかった場合、リスナーを用いてシャードへの新たな接続を生成することができる。この場合、接続ストリングを、シャードおよびスーパーシャードキーを含むように修正し、リスナーによって扱われるべき接続生成要求を行なう。データベースドライバ（たとえばＪＤＢＣ）は、接続を生成しようと試みる前に、ＵＲＬをこれら２つのキーを用いて更新することにより、getConnection (<shard_key>, <super_shard_key>)を扱う必要がある。

【０１５９】

（３）シャードキーに対する既存の有効なマッピングが存在しかつ上記ＡＰＩによって対応するサービス名が返された場合、接続プールは、このサービスへの、既に利用できる接続があるか否か確認する、すなわち、同じチャンクまたは仮想サービスへの接続が既に生成されているか否か確認する。もしあれば、この接続を直接再使用できる。

【０１６０】

（４）接続プール内に、このサービスのために利用できる接続がなく、プールに増大の余地がある場合は、新たな接続を生成する必要があるか否か、または、利用できる接続の用途を変更することで所望のチャンクへの接続を取得できるか否かを、検討すればよい。

【０１６１】

（５）（４）において接続の用途を変更すると決定した場合、用途を変更する候補を選択するために、以下のプロセスに従うことができる。すなわち、接続プール（たとえばＵＣＰ）は、OracleDBTopology API getServiceTopology (<service_name>)を用いて、仮想サービス（チャンク）を利用できるインスタンスのリストを得る。仮想サービストポロジが存在しない場合は、接続の用途を変更することを中止する必要があり、システムは、仮想サービス（チャンク）のために新たな接続を生成するよう試みなければならない。仮想サービストポロジを利用できるのであれば、仮想サービスに対応するチャンクを有するインスタンスのリストを取得し、たとえばプールで可能にされた場合はＲＬＢアルゴリズムを用いて、用途を変更する候補の接続を選択するために使用する最良のインスタンスを選ぶ。前のステップで選択されたインスタンス／シャードについては、利用できるすべての接続のリストを取得してから、この時点で最大数の利用できる接続を有するサービスから、用途を変更するのに最適な候補と思われる接続を選択する。次に、この候補接続を、接続上のサービスを切換えることにより用途変更し、この接続をユーザに返す。切換えサー

ビスが何らかの理由で失敗した場合は、上記提供されたインスタンスのリストからインスタンスを選択するためのランダムアルゴリズムに戻って、同じプロセスを繰返し、接続の用途変更を試みる。ランダムアルゴリズムを用いて接続の用途を変更することが、何らかの理由で失敗に終わった場合、プール内に増大の余地があればシャードキーによって指定されたチャンクへの新たな接続を生成しようと試みる。そうでなければエラーが報告される。

【０１６２】

ある実施形態に従うと、ユーザが接続ラベルおよびシャードキーを渡すことができるようにするgetConnectionを用いて、シャードキーを用いたプールからの接続取出しに対して、接続のラベリングをサポートできる。通常の接続選択は、ＲＬＢおよびアフィニティによる選択を含み、直後にラベリングマッチ（またはコスト）が続く。シャーディングサポートを用いて、システムは、最初にシャードキー（仮想サービス）によって、次にＲＬＢ／アフィニティによって、接続を選択することができ、接続ラベリングアルゴリズムが通常通り適用される。

【０１６３】

接続プール（たとえばＵＣＰ）における接続格納およびルックアップ

ある実施形態に従うと、接続が生成されると、接続プール（たとえばＵＣＰ）は、接続が生成されたチャンクの内部仮想サービスに対応する実際の「サービス」名を抽出してこれをこの接続のための接続要求識別子の中に配置しようとするであろう。接続要求識別子を用いて、ＵＣＰの接続格納部から、正しいサービス名（チャンク）を有する接続をルックアップして借用する。また、プール内で一致する接続が得られず接続サービスの切換えが差し迫って必要である場合は、サービス名を用いて、用途変更するのに最適な候補接続を決定する。

【０１６４】

OracleDBTopologyモジュール

図１１は、ある実施形態に従う、データベーストポロジクラス設計３５０を示す。

【０１６５】

ある実施形態に従うと、OracleDBTopologyモジュールは、データベースドライバ（たとえばＪＤＢＣ）の中に置くことができ、データベースのすべてのトポロジデータを含む。これは、たとえば、プールまたはアプリケーションがデータベースドライバを用いて参照するすべてのサービスおよびシャードキーに関する、サービスからインスタンスへのマッピングおよびシャードキーからサービスへのマッピングである。

【０１６６】

ある実施形態に従うと、OracleDBTopologyインスタンスは、OracleDataSourceインスタンスに対応付けられ、これらの間には１：１のマッピングがある。OracleDBTopology内にある動作およびデータを、シャードまたはサービストポロジをルックアップするためにデータベースドライバのユーザが使用することができ、キャッシュされたトポロジに基づいて接続要求を効率的にルーティングするために接続プール（たとえばＵＣＰ）が使用することができ、このモジュールを強化することによりプールがトポロジデータを使用しシャードされたデータベースへの接続のプールを維持できるようにするカスタムプール実装が使用することができる。

【０１６７】

ある実施形態に従うと、OracleDBTopologyは、getConnection () APIのうちの１つを用いてOracleDataSourceを用いて最初の接続が生成されたときは、ゆっくりと初期化してもよく、接続オブジェクトを用いて、OracleDBTopologyモジュールを初期化するのに必要な情報を得る。生成された接続を用いて、LOCAL_CHUNK_TYPESテーブルをルックアップする。SHARD_TYPEおよびSUPER_TYPE列を読取って、シャーディングおよびスーパーシャーディング手法を決定する。これらの値はプロパティとしてOracleDBTopologyインスタンスに格納される。DEF_VERSIONも抽出されクライアント側に置かれる。

【０１６８】

サービストポロジ

図１２は、ある実施形態に従う、サービストポロジクラス設計３６０を示す。

【０１６９】

ある実施形態に従うと、ServiceTopologyMapは、仮想サービス名と、サービスが現在実行されているインスタンスのリストとの間の、１：ｎマッピングのマップを含む。このマップの中身は、各接続が生成されupdatedOnCreateConnection()という方法がOracleDbTopology上で呼出されると徐々に増えてゆく。

【０１７０】

ある実施形態に従うと、sys_contextには、仮想サービス名も、現在の接続が生成されているインスタンス名も、存在する。新たな接続が生成されるたびに、このシステムは、仮想サービス名が既にマップ内にあるか否か確認し、そうでなければ新たなエントリを仮想サービス名およびインスタンスに対して追加する。仮想サービス名がすでにマップ内にありインスタンス名もサービスのインスタンスリスト内にある場合、他のアクションは不要である。そうでなければインスタンスをインスタンスリストに追加する。

【０１７１】

ある実施形態に従うと、ShardTopologyMapは、インターフェイスを提供し、チャンクキー範囲またはリストを仮想サービス名にマッピングするのに使用できるデータ構造を規定する。

【０１７２】

インターフェイスおよびＡＰＩ

ある実施形態に従い、データベースドライバ（たとえばＪＤＢＣ）ＡＰＩおよび接続プール（たとえばＵＣＰ）ＡＰＩを含む典型的なアプリケーションプログラムインターフェイス（ＡＰＩ）を以下で説明する。他の実施形態および実施例に従うと、他の種類のＡＰＩまたはインターフェイスを使用することができる。

【０１７３】

アペンディックスＡ：データベースドライバ（たとえばＪＤＢＣ）ＡＰＩ

ある実施形態に従い、典型的なデータベースドライバ（たとえばＪＤＢＣ）ＡＰＩを以下で説明する。他の種類のデータベースドライバによると、他の種類のデータベースドライバＡＰＩまたはインターフェイスを使用することができる。

【０１７４】

シャードキーインターフェイス

【０１７５】

【数１７】

```
public interface ShardKey extends Comparable<ShardKey>{
  /**
   * Used to compare two shard keys. If the shard keys are
   * compound the corresponding sub-keys in two keys will be
   * compared.
   *
   * @param o
   *        ShardKey to which this Shard key is to be compared.
   * @return -1, 0 or 1 as this ShardKey is less than, equal
     *        to, or greater than the shard key that is passed in
     *        as a method parameter
   * @see java.lang.Comparable#compareTo(java.lang.Object)
   */
  int compareTo(ShardKey o);
}
```

【０１７６】

接続ビルダインターフェイス

【０１７７】

【数１８】

```
/**
 * Builder class for building connection objects with additional parameters
 * other than just the username and password. To use the builder, the
 * corresponding builder method needs to be called for each parameter that needs
 * to be part of the connection request followed by a build() method.
 * The order in which the builder methods are called is not important.
 * However if the same builder attribute is applied more than once, only
 * the most recent value will be considered while building the connection.
 * The builder's build() method can be called only once on a builder object.
 *
 * Example usage :
 *   OracleDataSource ods = new OracleDataSource();
 *   ..//set the required properties on the datasource
 *   Connection conn = ods.createConnectionBuilder()
 *                    .user("user")
 *                    .password("password")
 *                    .proxyClientName("proxy_client")
 *                    .serviceName("service_name")
 *                    .build();
 *
 * @param <B>
 *         Type of connection builder
 * @param <S>
 *         Type of connections created using this builder
```

【０１７８】

【数１９】

```
 */
public interface ConnectionBuilder<B extends ConnectionBuilder<B, S>, S>
    extends Builder<S> {
  /**
   * @param user
   * @return This connection builder object
   */
  B user(String user);

  /**
   * @param password
   * @return This connection builder object
   */
  B password(String password);

  /**
   * @param serviceName
   * @return This connection builder object
   */
  B serviceName(String serviceName);

  /**
   * @param shardKey
   *        Shard Key object that needs to be part of connection request
   * @return This instance of the connection builder.
   */
  B shardKey(ShardKey shardKey);

  /**
   * @param shardGroupKey
   *        Shard Group Key object that needs to be part of connection request
   * @return This instance of the connection builder.
   */
  B shardGroupKey(ShardKey shardGroupKey);
}
```

【０１７９】

【数２０】

OracleConnectionBuilder

```
class oracle.jdbc.pool.OracleConnectionBuilder
    implements oracle.jdbc.ConnectionBuilder<OracleConnectionBuilder,Connection>
{
    //Oracle's Implementation class for ConnectionBuilder.
}
```

OracleDataSource

```
class oracle.jdbc.pool.OracleDataSource
{
   OracleConnectionBuilder createConnectionBuilder() { }
   OracleShardKeyBuilder createShardKeyBuilder();
}
```

【０１８０】

【数２１】

OracleConnection

```
interface oracle.jdbc.OracleConnection
{
  /**
   * Used to check the validity of the connection and if the shard keys passed
   * to this method are valid for this connection.
   *
   * @param shardKey
   *         shard key to be validated against this connection
   * @param groupKey
   *         shard group key to be validated against this connection
   * @param timeout
   *         time in seconds before which the validation process is expected to
   *         be completed,else the validation process is aborted.
   * @return true if the connection is valid and the shard keys are valid to be
   *        set on this connection.
   * @throws SQLException
   *        if there is any exception while performing this validation.
   */
  boolean isValid(ShardKey shardKey, ShardKey groupKey, int timeout)
     throws SQLException;

  /**
   * Used to set the shard key and the shard group key on this connection.
   *
   * @param shardKey
   *         shard key to be set on this connection
   * @param groupKey
   *         shard group key to be set on this connection
   * @throws SQLException
   *          if there is an exception while setting the shard keys
   *          on this connection. In this case the connection will continue to
   *          be associated with the shard keys that was set on this connection
   *          before this method was called.
   */
  void setShardKey(ShardKey shardKey, ShardKey groupKey) throws SQLException;}
}
```

【０１８１】

アペンディックスＢ：接続プール（たとえばＵＣＰ）ＡＰＩ

ある実施形態に従い、典型的な接続プール（たとえばＵＣＰ）ＡＰＩを以下で説明する。他の種類の接続プールによると、他の種類の接続プールＡＰＩまたはインターフェイスを使用することができる。

【０１８２】

ＵＣＰ接続ビルダ

【０１８３】

【数２２】

```
/**
 * UCP's Connection Builder class for building connection objects with additional parameters
 * other than just the username,password and labels. To use the builder, the
 * corresponding builder method needs to be called for each parameter that needs
 * to be part of the connection request followed by a build() method. The order
 * in which the builder methods are called is not important. However if the same
 * builder attribute is applied more than once, only the most recent value will
 * be considered while building the connection. The builder's build() method can
 * be called only once on a builder object.
 *
 * Example usage :
 * PoolDataSource pds = PoolDataSourceFactory.getPoolDataSource();
 * ..//set the required properties on the datasource
 *
 * Connection conn = pds.createConnectionBuilder()
 *               .user("user")
 *               .password("password")
 *               .serviceName("service_name")
 *               .build();
 *
 * @param <C>
 *        Type of connections created using this builder
 * @param <B>
 *        Type of connection builder
```

【０１８４】

【数２３】

```
*/
public interface oracle.ucp.jdbc.UCPConnectionBuilder extends
    oracle.jdbc.ConnectionBuilder<UCPConnectionBuilder, Connection> {

/**
   * Sets the user attribute on the builder
   *
   * @param user
   *        - the database user on whose behalf the connection is being made
   * @return - this builder object
   */
  @Override
  public UCPConnectionBuilder user(String user) ;

/**
   * Sets the password attribute on the builder
   *
   * @param password
   *        - the user's password
   * @return - this builder object
   */
  @Override
  public UCPConnectionBuilder password(String password) ;
```

【０１８５】

【数２４】

```
  /**
   * Sets the labels attribute on the builder
   *
   * @param labels
   *         - The requested connection labels.
   * @return - this builder object
   */
  public UCPConnectionBuilder labels(Properties labels) ;

  /**
   * @param serviceName to retrieve the connection
   * @return this connection builder instance
   */
  public UCPConnectionBuilder serviceName(String serviceName);

  /**
   * @param shardKey
   *         Shard Key object that needs to be part of connection request
   * @return This instance of the connection builder.
   */
  public UCPConnectionBuilder shardKey(ShardKey shardKey) ;

  /**
   * @param shardGroupKey
   *         Shard Group Key object that needs to be part of connection request
   * @return This instance of the connection builder.
```

【０１８６】

【数２５】

```
   */
  public UCPConnectionBuilder shardGroupKey(ShardKey shardGroupKey);

  /**
   * @return Connection built considering the builder attributes
   * @throws SQLException if there is a failure in building the connection.
   */

 public Connection build() throws SQLException;

  /**
   * Sets the labels attribute on the builder
   *
   * @param labels
   *         - The requested connection labels.
   * @return - this builder object
   */
  ConnectionBuilder labels(Properties labels);
}
```

【０１８７】

【数２６】

```
PoolDataSource Interface
Interface oracle.ucp.jdbc.PoolDataSource
{
  /**
   * @return UCPConnectionBuilder that can help build Connection with multiple
   *        parameters other than just the user,password and labels.
   */
  public UCPConnectionBuilder createConnectionBuilder();

  /**
   * @return OracleShardKeyBuilder that can help build Shard Keys.
   */
  public OracleShardKeyBuilder createShardKeyBuilder();
}
```

【０１８８】

本発明の実施形態は、本開示の教示に従ってプログラムされた、１つ以上のプロセッサ、メモリ、および／またはコンピュータ読取可能な記憶媒体を含む、１つ以上の従来の汎用または専用デジタルコンピュータ、コンピューティングデバイス、マシン、またはマイクロプロセッサを用いて、適宜実現し得る。熟練したプログラマは、本開示の教示に基づいて、適切なソフトウェアコーディングを容易に作成することができ、このことはソフトウェア技術における当業者には明らかであろう。

【０１８９】

いくつかの実施形態において、本発明は、本発明のプロセスのうちのいずれかを実行するようにコンピュータをプログラムするのに使用できる命令が格納されている非一時的な記憶媒体またはコンピュータ読取可能な媒体（複数の媒体）であるコンピュータプログラムプロダクトを含む。上記記憶媒体の例は、フロッピー（登録商標）ディスク、光ディスク、ＤＶＤ、ＣＤ－ＲＯＭ、マイクロドライブ、および光磁気ディスクを含む何らかの種類のディスク、ＲＯＭ、ＲＡＭ、ＥＰＲＯＭ、ＥＥＰＲＯＭ、ＤＲＡＭ、ＶＲＡＭ、フラッシュメモリデバイス、磁気もしくは光カード、ナノシステム（分子メモリＩＣを含む）、または、命令および／またはデータの格納に適した何らかの種類の媒体またはデバイスを含み得るが、これらに限定される訳ではない。

【０１９０】

本発明の実施形態のこれまでの説明は、例示と説明を目的として提供されている。すべてを網羅すること、または、本発明を開示されている形態そのものに限定することは意図されていない。多くの修正および変形が当業者には明らかであろう。これらの実施形態は、本発明の原理およびその実際の応用を最も上手く説明することによって、意図されている特定の用途に適した本発明のさまざまな実施形態とさまざまな変形を当業者が理解できるようにすることを目的として、選択されて記載されている。

【図１】

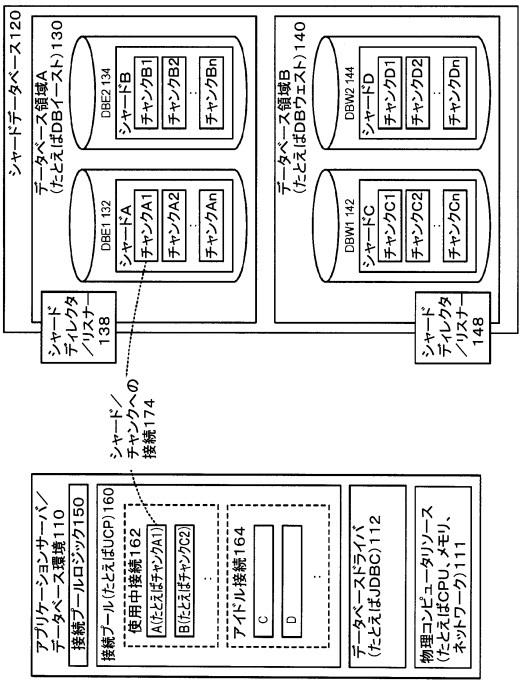

FIGURE 1

【図２】

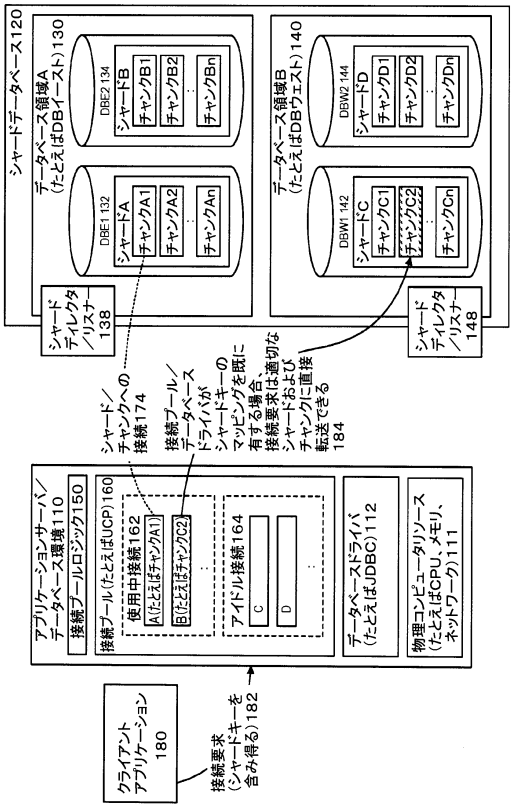

FIGURE 2

【図３】

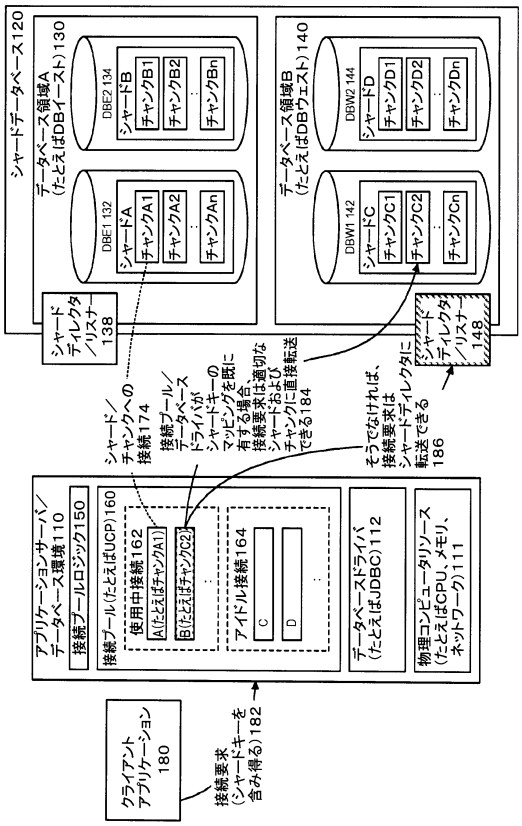

FIGURE 3

【図４】

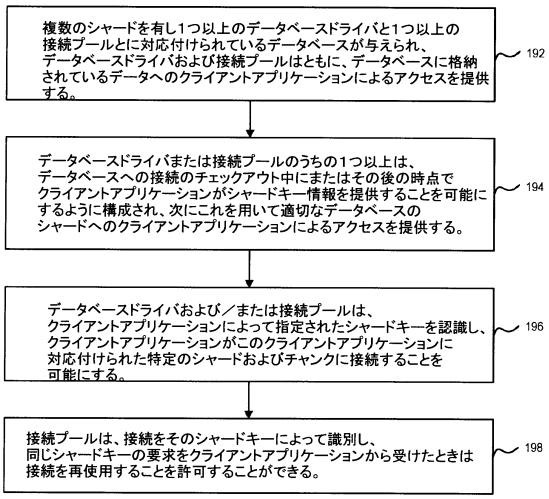

FIGURE 4

【図５】

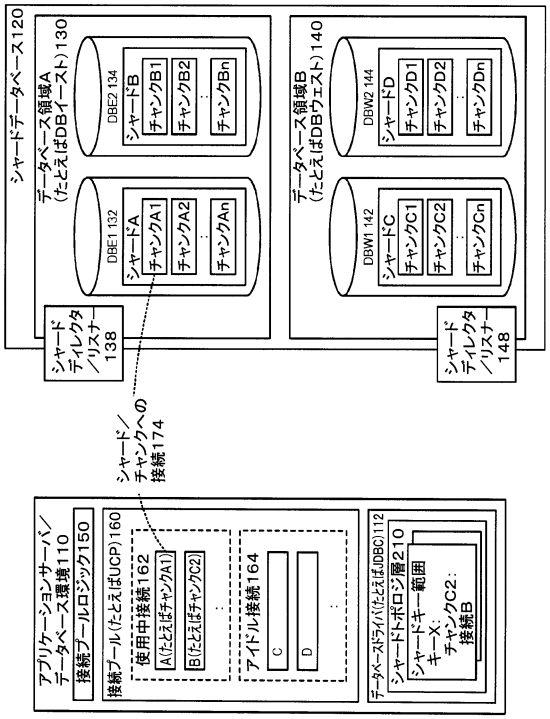

FIGURE 5

【図６】

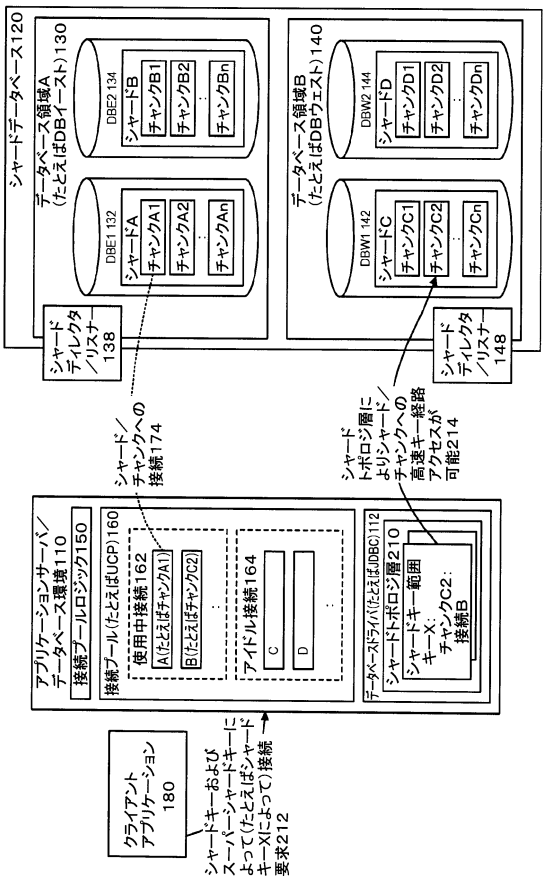

FIGURE 6

【図７】

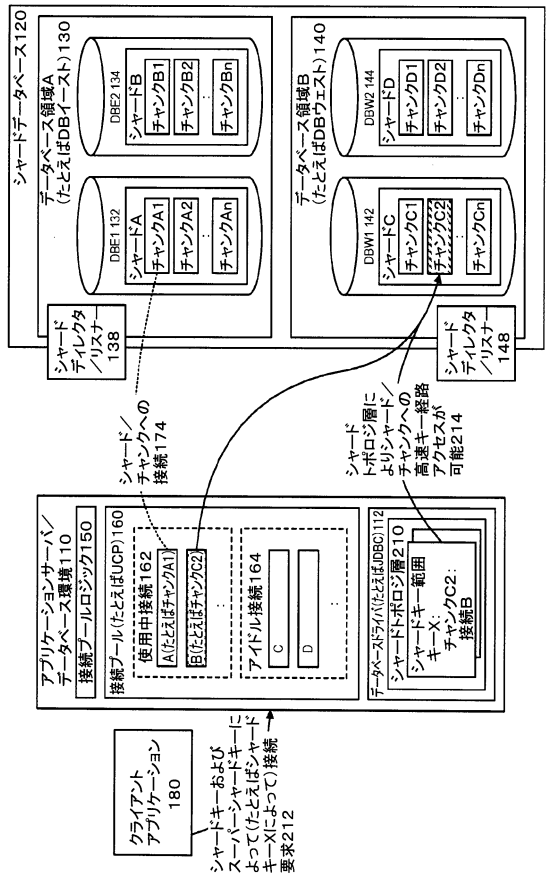

FIGURE 7

【図８】

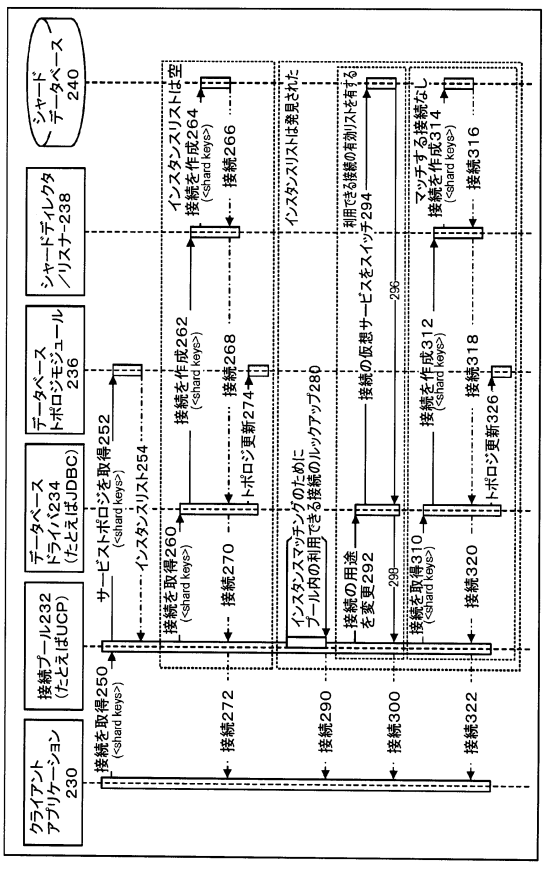

FIGURE 8

【図９】

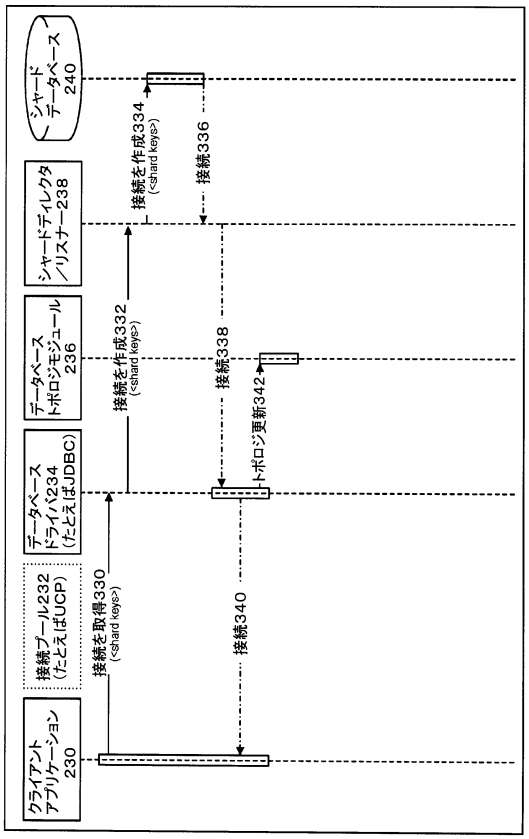

FIGURE 9

【図１０】

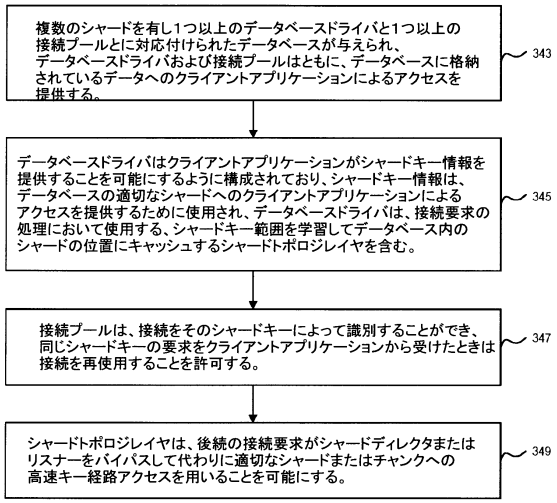

FIGURE 10

【図１１】

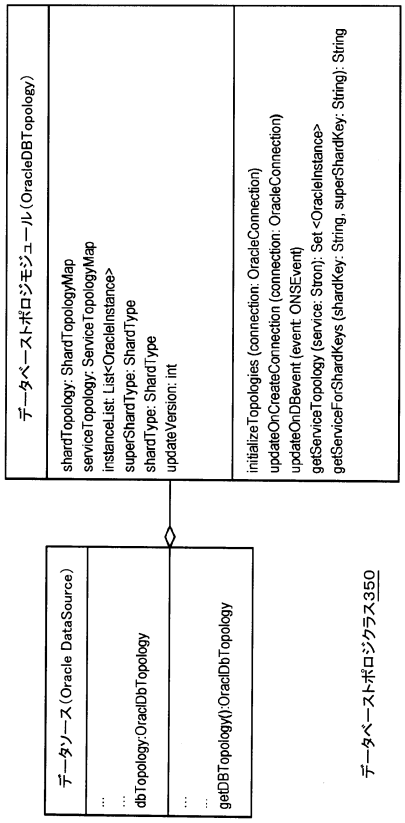

FIGURE 11

【図１２】

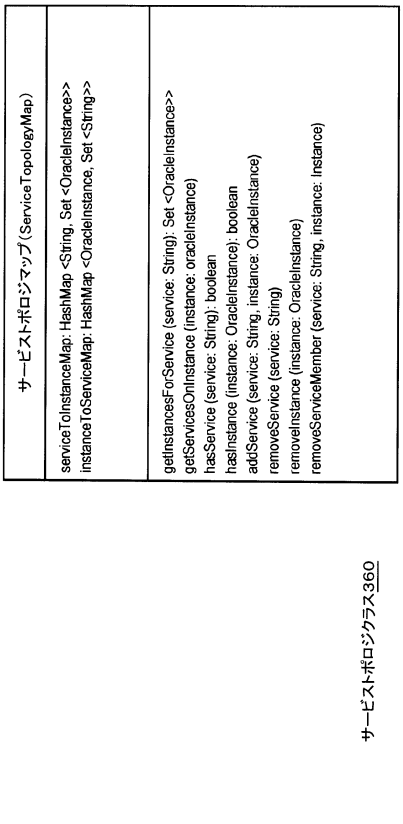

FIGURE 12

フロントページの続き

(31)優先権主張番号 62/198,958
(32)優先日 平成27年7月30日(2015.7.30)
(33)優先権主張国・地域又は機関
米国(US)

(72)発明者 ドゥ・ラバルネ, ジャン
フランス、７８０００ ベルサイユ、リュ・デュ・マル・ドゥ・ラトル・ドゥ・タシニー、１６
(72)発明者 サーバー, ダグラス
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 ディルマン, マーク
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 ノバーク, レオニード
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 フ, ウェイ
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 シバルドライアー, アショク
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 ヂョウ, トン
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内
(72)発明者 タラノブ, イリヤ
アメリカ合衆国、９４０６５ カリフォルニア州、レッドウッド・ショアーズ、オラクル・パークウェイ、５００、エム／エス・５・オゥ・ピィ・７、オラクル・インターナショナル・コーポレイション内

審査官 鹿野 博嗣

(56)参考文献 特開２００６－２５９７９０（ＪＰ，Ａ）
バーグステン ハンス, ＪａｖａＳｅｒｖｅｒ Ｐａｇｅｓ 第2版 初版, 株式会社オライリー・ジャパン, ２００３年 5月３１日, 第513頁～第522頁

(58)調査した分野(Int.Cl., ＤＢ名)
Ｇ０６Ｆ １６／１８２