



- (51) International Patent Classification:
G06F 9/50 (2006.01) **H04L 29/08** (2006.01)
- (21) International Application Number:
PCT/US2017/045283
- (22) International Filing Date:
03 August 2017 (03.08.2017)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
15/227,897 03 August 2016 (03.08.2016) US
- (71) Applicant: **ORACLE INTERNATIONAL CORPORATION** [US/US]; 500 Oracle Parkway, M/S 5op7, Redwood Shores, California 94065 (US).
- (72) Inventors: **DE LAVARENE, Jean**; 16 rue du Mal de Latre de Tassigny, 78000 Versailles (FR). **VERMA, Saurabh**;

c/o Oracle India Pvt Ltd., Lexington Tower, Prestige Street, St. John's Woods, No. 18, 2nd Cross Road, Chikka Audugodi, Bangalore 560 029 (IN). **HEGDE, Vidya**; c/o Oracle India Pvt Ltd., Lexington Tower, Prestige Street, St. John's Woods, No. 18, 2nd Cross Road, Chikka Audugodi, Bangalore 560 029 (IN). **MAHIDHARA, Chandra Sekhar Krishna**; c/o Oracle India Pvt Ltd., Lexington Tower, Prestige Street, St. John's Woods, No. 18, 2nd Cross Road, Chikka Audugodi, Bangalore 560 029 (IN). **NAMACHIVAYAM, Aramvalarthanathan**; c/o Oracle India Pvt Ltd., Lexington Tower, Prestige Street, St. John's Woods, No. 18, 2nd Cross Road, Chikka Audugodi, Bangalore 560 029 (IN).

(74) Agent: **MEYER, Sheldon, R.** et al.; TUCKER ELLIS LLP, One Market Plaza, Steuart Tower, Suite 700, San Francisco, California 94105 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ,

(54) Title: SYSTEM AND METHOD FOR PROVIDING DYNAMIC RELOCATION OF TENANTS IN A MULTI-TENANT DATABASE ENVIRONMENT

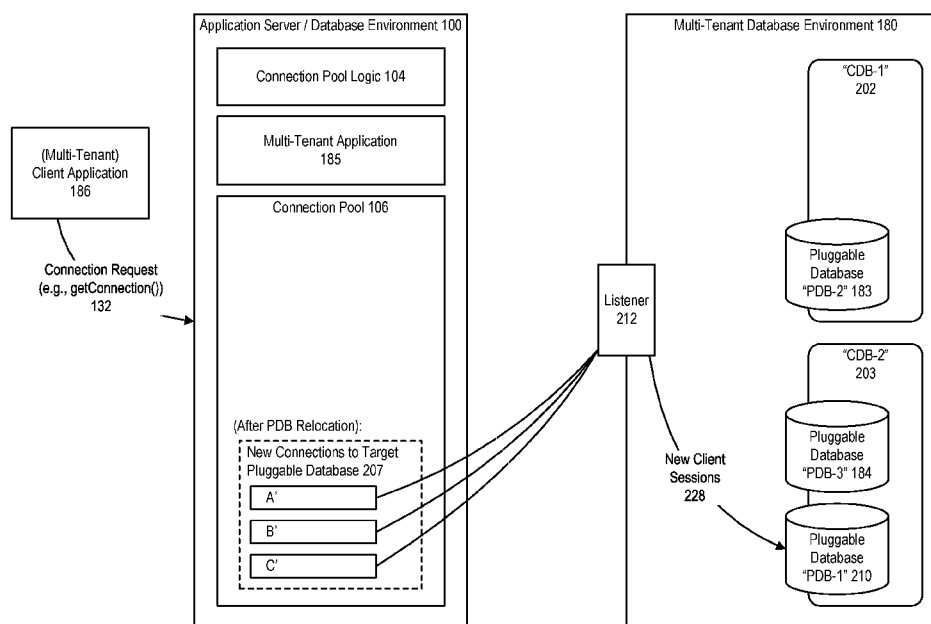


FIGURE 9

(57) Abstract: Described herein are systems and methods for providing access to a database in a multi-tenant environment, including the use of a connection pool, and support for dynamic relocation of tenants. In accordance with an embodiment, a software application can obtain a connection from the connection pool, on behalf of a tenant, which enables the software application or tenant to access the database. A relocation process enables a tenant which is associated with a multi-tenant or other client application, to be relocated within the database environment, for example across a plurality of container databases, with near-zero downtime to the client application, including managing the draining of existing connections, and the migrating of new connections, without requiring changes to the underlying application.



CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO,
DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN,
HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP,
KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME,
MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ,
OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA,
SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN,
TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

**SYSTEM AND METHOD FOR PROVIDING DYNAMIC RELOCATION OF TENANTS
IN A MULTI-TENANT DATABASE ENVIRONMENT**

COPYRIGHT NOTICE

A portion of the disclosure of this patent document contains material which is subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure, as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

Claim of Priority:

[0001] This application claims the benefit of priority to U.S. Patent Application titled "SYSTEM AND METHOD FOR PROVIDING DYNAMIC RELOCATION OF TENANTS IN A MULTITENANT DATABASE ENVIRONMENT", Application No. 15/227,897 filed August 3, 2016, which application is herein incorporated by reference.

Field of Invention:

[0002] Embodiments of the invention are generally related to software application servers and databases, and are particularly related to systems and methods for providing access to a database in a multi-tenant environment, including the use of a connection pool, and support for dynamic relocation of tenants.

Background:

[0003] Generally described, in a database environment, a connection pool operates as a cache of connection objects, each of which represents a connection that can be used by a software application to connect to a database. At runtime, an application can request a connection from the connection pool. If the connection pool includes a connection that can satisfy the particular request, it can return that connection to the application for its use. In some instances, if no suitable connection is found, then a new connection can be created and returned to the application. The application can borrow the connection to access the database and perform some work, and then return the connection to the pool, where it can then be made available for subsequent connection requests from the same, or from other, applications.

Summary:

[0004] Described herein are systems and methods for providing access to a database in a multi-tenant environment, including the use of a connection pool, and support for dynamic

relocation of tenants. In accordance with an embodiment, a software application can obtain a connection from the connection pool, on behalf of a tenant, which enables the software application or tenant to access the database. A relocation process enables a tenant which is associated with a multi-tenant or other client application, to be relocated within the database environment, for example across a plurality of container databases, with near-zero downtime to the client application, including managing the draining of existing connections, and the migrating of new connections, without requiring changes to the underlying application.

Brief Description of the Figures:

[0005] Figure 1 illustrates a system that includes a connection pool, in accordance with an embodiment.

[0006] Figure 2 further illustrates a system that includes a connection pool, including support for use of a sharded database, in accordance with an embodiment.

[0007] Figure 3 further illustrates a system that includes a connection pool, including support for use in a multi-tenant environment, in accordance with an embodiment.

[0008] Figure 4 illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0009] Figure 5 further illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0010] Figure 6 further illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0011] Figure 7 further illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0012] Figure 8 further illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0013] Figure 9 further illustrates support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0014] Figure 10 illustrates a method of providing support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

Detailed Description:

[0015] As described above, a connection pool operates as a cache of connection objects, each of which represents a connection that can be used by a software application to connect to a database. At runtime, an application can request a connection from the connection pool. If the connection pool includes a connection that can satisfy the particular request, it can return that connection to the application for its use. In some instances, if no suitable connection is found, then a new connection can be created and returned to the

application. The application can borrow the connection to access the database and perform some work, and then return the connection to the pool, where it can then be made available for subsequent connection requests from the same, or from other, applications.

[0016] Creating connection objects can be costly in terms of time and resources. For example, tasks such as network communication, authentication, transaction enlistment, and memory allocation, all contribute to the amount of time and resources it takes to create a particular connection object. Since connection pools allow the reuse of such connection objects, they help reduce the number of times that the various objects must be created.

[0017] One example of a connection pool is Oracle Universal Connection Pool (UCP), which provides a connection pool for caching Java Database Connectivity (JDBC) connections. For example, the connection pool can operate with a JDBC driver to create connections to a database, which are then maintained by the pool; and can be configured with properties that are used to further optimize pool behavior, based on the performance and availability requirements of a requesting software application.

Connection Labeling

[0018] Figure 1 illustrates a system that includes a connection pool, in accordance with an embodiment.

[0019] As illustrated in Figure 1, in accordance with an embodiment, an application server or database environment 100, which includes physical computer resources 101 (e.g., a processor/CPU, memory, and network components), for example an Oracle WebLogic Server, Oracle Fusion Middleware, or other application server or database environment, can include or provide access to a database 102, for example an Oracle database, or other type of database.

[0020] As further illustrated in Figure 1, in accordance with an embodiment, the system also includes a connection pool logic 104 or program code, which when executed by a computer controls 105 the creation and use of connection objects in a connection pool 106, including, for example, connections that are currently in use 108 by a software application, and connections that are idle 110, or are not currently being used.

[0021] Software applications can initialize connections retrieved from a connection pool, before using the connection to access, or perform work at the database. Examples of initialization can include simple state re-initializations that require method calls within the application code, or more complex initializations including database operations that require round trips over a network. The computational cost of these latter types of initialization may be significant.

[0022] Some connection pools (for example, UCP) allow their connection pools to be configured using connection pool properties, that have get and set methods, and that are

available through a pool-enabled data source instance. These get and set methods provide a convenient way to programmatically configure a pool. If no pool properties are set, then a connection pool uses default property values.

[0023] In accordance with an embodiment, labeling connections allows a client software application to attach arbitrary name/value pairs to a connection. The application can then request a connection with a desired label from the connection pool. By associating particular labels with particular connection states, an application can potentially retrieve an already-initialized connection from the pool, and avoid the time and cost of re-initialization. Connection labeling does not impose any meaning on user-defined keys or values; the meaning of any user-defined keys and values is defined solely by the application.

[0024] For example, as illustrated in Figure 1, in accordance with an embodiment, the connection pool can include a plurality of connections that are currently in use by software applications, here indicated as connections A 112 and B 114. Each of the connections can be labeled, for example connection A is labeled (Blue) and connection B is labeled (Green). These labels/colors are provided for purposes of illustration, and as described above can be arbitrary name/value pairs attached to a connection by a client application. In accordance with various embodiments, different types of labels can be used, to distinguish between different connection types; and different applications can attach different labels/colors to a particular connection type.

[0025] As further illustrated in Figure 1, in accordance with an embodiment, the connection pool can also include a plurality of connections that are idle, or are not currently being used by software applications, here indicated as connections C 116, D 118, E 120, F 122, G 124 and N 126. Each of the idle connections can be similarly labeled, in this illustration as (Blue) or (Green), and again these labels/colors are provided for purposes of illustration.

[0026] As further illustrated in Figure 1, in accordance with an embodiment, if a software application 130 wishes to make a request on the database, using a particular type of connection, for example a (Red) connection, then the application can make a "getConnection(Red)" request 132. In response, the connection pool logic will either create a new (Red) connection, here indicated as X 134 (Red); or repurpose an existing idle connection from (Blue or Green) to (Red), here indicated as E 135 (Red).

Sharded Databases

[0027] In accordance with an embodiment, sharding is a database-scaling technique which uses a horizontal partitioning of data across multiple independent physical databases. The part of the data which is stored in each physical database is referred to as a shard. From the perspective of a software client application, the collection of all of the physical databases appears as a single logical database.

[0028] In accordance with an embodiment, the system can include support for use of a connection pool with sharded databases. A shard director or listener provides access by software client applications to database shards. A connection pool (e.g., UCP) and database driver (e.g., a JDBC driver) can be configured to allow a client application to provide a shard key, either during connection checkout or at a later time; recognize shard keys specified by the client application; and enable connection by the client application to a particular shard or chunk. The approach enables efficient re-use of connection resources, and faster access to appropriate shards.

[0029] **Figure 2** further illustrates a system that includes a connection pool, including support for use of a sharded database, in accordance with an embodiment.

[0030] In accordance with an embodiment, a database table can be partitioned using a shard key (SHARD_KEY), for example as one or more columns that determine, within a particular shard, where each row is stored. A shard key can be provided in a connect string or description as an attribute of connect data (CONNECT_DATA). Examples of shard keys can include a VARCHAR2, CHAR, DATE, NUMBER, or TIMESTAMP in the database. In accordance with an embodiment, a sharded database can also accept connections without a shard key or shard group key.

[0031] In accordance with an embodiment, to reduce the impact of resharding on system performance and data availability, each shard can be subdivided into smaller pieces or chunks. Each chunk acts as a unit of resharding that can be moved from one shard to another. Chunks also simplify routing, by adding a level of indirection to the shard key mapping.

[0032] For example, each chunk can be automatically associated with a range of shard key values. A user-provided shard key can be mapped to a particular chunk, and that chunk mapped to a particular shard. If a database operation attempts to operate on a chunk that is not existent on a particular shard, then an error will be raised. When shard groups are used, each shard group is a collection of those chunks that have a specific value of shard group identifier.

[0033] A shard-aware client application can work with sharded database configurations, including the ability to connect to one or multiple database shards in which the data is partitioned based on one or more sharding methods. Each time a database operation is required, the client application can determine the shard to which it needs to connect.

[0034] In accordance with an embodiment, a sharding method can be used to map shard key values to individual shards. Different sharding methods can be supported, for example: hash-based sharding, in which a range of hash values is assigned to each chunk, so that upon establishing a database connection the system applies a hash function to a given value of the sharding key, and calculates a corresponding hash value which is then mapped

to a chunk based on the range to which that value belongs; range-based sharding, in which a range of shard key values is assigned directly to individual shards; and list-based sharding, in which each shard is associated with a list of shard key values.

[0035] As illustrated in Figure 2, in accordance with an embodiment a sharded database 140 can comprise a first database region A (here indicated as "DB East", DBE) 141, including sharded database instances "DBE-1" 142, with a shard A stored as chunks A1, A2, ... An; and "DBE-2" 143, with a shard B stored as chunks B1, B2, ... Bn.

[0036] As further illustrated in Figure 2, in accordance with an embodiment, a second database region B (here indicated as "DB West", DBW) 144, includes sharded database instances "DBW-1" 145, with a shard C stored as chunks C1, C2, ... Cn; and "DBW-2" 146, with a shard D stored as chunks D1, D2, ... Dn.

[0037] In accordance with an embodiment, each database region or group of sharded database instances can be associated with a shard director or listener (e.g., an Oracle Global Service Managers (GSM) listener, or another type of listener). For example, as illustrated in Figure 2, a shard director or listener 147 can be associated with the first database region A; and another shard director or listener 148 can be associated with the second database region B. The system can include a database driver (e.g., a JDBC driver) 152 that maintains a shard topology layer 154, which over a period of time learns and caches shard key ranges to the location of each shard in a sharded database.

[0038] In accordance with an embodiment, a client application can provide one or more shard keys to the connection pool during a connection request 162; and, based on the one or more shard keys, and information provided by the shard topology layer, the connection pool can route the connection request to a correct or appropriate shard.

[0039] In accordance with an embodiment, the connection pool can also identify a connection to a particular shard or chunk by its shard keys, and allow re-use of a connection when a request for a same shard key is received from a particular client application.

[0040] For example, as illustrated in Figure 2, in accordance with an embodiment, a connection to a particular chunk (e.g., chunk A1) can be used to connect 174, to that chunk. If there are no available connections in the pool to the particular shard or chunk, the system can attempt to repurpose an existing available connection to another shard or chunk, and re-use that connection. The data distribution across the shards and chunks in the database can be made transparent to the client application, which also minimizes the impact of re-sharding of chunks on the client.

[0041] When a shard-aware client application provides one or more shard keys to the connection pool, in association with a connection request; then, if the connection pool or database driver already has a mapping for the shard keys, the connection request can be directly forwarded to the appropriate shard and chunk, in this example, to chunk C2.

[0042] When a shard-aware client application does not provide a shard key in association with the connection request; or if the connection pool or database driver does not have a mapping for a provided shard key; then the connection request can be forwarded to an appropriate shard director or listener.

5

Multi-Tenant Environments

[0043] In accordance with an embodiment, the system can include support for cloud-based or multi-tenant environments using connection labeling. For example, a multi-tenant cloud environment can include an application server or database environment that includes or provides access to a database for use by multiple tenants or tenant applications, in a cloud-based environment.

10

[0044] Figure 3 further illustrates a system that includes a connection pool, including support for use in a multi-tenant environment, in accordance with an embodiment.

[0045] Software applications, which can be accessed by tenants via a cloud or other network, may, similarly to the environments described above, initialize connections retrieved from a connection pool before using the connection.

15

[0046] As described above, examples of initialization can include simple state re-initializations that require method calls within the application code, or more complex initializations including database operations that require round trips over a network.

[0047] As also described above, labeling connections allows an application to attach arbitrary name/value pairs to a connection, so that the application can then request a connection with a desired label from the connection pool, including the ability to retrieve an already-initialized connection from the pool and avoid the time and cost of re-initialization.

20

[0048] As illustrated in Figure 3, in accordance with an embodiment, a multi-tenant database environment 180 can include, for example, a container database (CDB) 181, and one or more pluggable database (PDB), here illustrated as "PDB-1" 182, "PDB-2" 183, and "PDB-3" 184.

25

[0049] In accordance with an embodiment, each PDB can be associated with a tenant, here illustrated as "Tenant-1", "Tenant-2", and "Tenant-3", of a multi-tenant application that is either hosted by the application server or database environment 185, or provided as an external client application 186, and which provides access to the database environment through the use of one or more Oracle Real Application Cluster (RAC) instances 186, 188, including in this example "RAC-Instance-1", and "RAC-Instance-2"; one or more services, including in this example Service-1", "Service-2", and "Service-3", and a mapping of tenants to services 190.

30

35

[0050] In the example illustrated in Figure 3, an application being used by a tenant to access the database environment, can make connection requests associated with that tenant's

data source 192, 194, 196, and the system can switch services 198 if necessary, to utilize connections to existing RAC instances or PDBs.

Server-Side Connection Pools

- 5 **[0051]** In accordance with an embodiment, the system can utilize a server-side connection pool tagging feature, such as that provided, for example, by Oracle Database Resident Connection Pool (DRCP). A server-side connection pool tagging feature allows user applications or clients to selectively obtain a connection to a database environment, based on use of a single tag that is understood by that database environment.
- 10 **[0052]** In accordance with an embodiment, only one tag is associated per connection. The database server does not communicate the tag value to the user applications or clients, but rather communicates a tag-match (for example, as a Boolean value).

Dynamic Relocation of a Tenant in the Pool

- 15 **[0053]** In accordance with an embodiment, the system can include support for dynamic relocation of tenants. A software application can obtain a connection from the connection pool, on behalf of a tenant, which enables the software application or tenant to access the database. A relocation process enables a tenant which is associated with a multi-tenant or other client application, to be relocated within the database environment, for example across a plurality of
- 20 container databases, with near-zero downtime to the client application, including managing the draining of existing connections, and the migrating of new connections, without requiring changes to the underlying application.

[0054] **Figures 4-9** illustrate support for dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

- 25 **[0055]** As illustrated in **Figure 4**, in accordance with an embodiment, a database, for example a container database (e.g., “CDB-1” 202), or another type of database, supports the use of a plurality of connections 204.

- [0056]** A tenant, which is associated with a multi-tenant or other client application hosted either by the application server or database environment, or provided as an external
- 30 client application, can use the connection pool to access the database, including where appropriate accessing a pluggable database of a container database, via a database service.

- [0057]** For example, in accordance with an embodiment, each particular tenant can be associated with its own particular pluggable database at the container database, and can use connections provided by the connection pool, to access (e.g., 205) the particular pluggable
- 35 database associated with that tenant, via a database service associated with the particular pluggable database.

[0058] In accordance with an embodiment, if the database environment changes, for

example a second container database (e.g., “CDB-2” 203) is added to the system, or in response to an application server that is hosting the connection pool receiving a service-down event 206 from the database environment, the system can provide new connections 207 to a new database location, for use by a particular tenant.

5 **[0059]** For example, in accordance with an embodiment, the system can initiate a migration of a pluggable database, for use by a tenant, including draining connections that are associated with an original pluggable database location and its associated database service (for example, those connections associated with “PDB-1” 182, in “CDB-1” 202); and migrating or otherwise relocating the availability of those connections 208 to a new pluggable database
10 location and associated database service (for example, here illustrated as “PDB-1” 210, in “CDB-2” 203).

[0060] This enables the connection pool to support near-zero-downtime tenant relocation, by draining the existing connections associated with a tenant’s original location, and creating new connections that point to the tenant’s new location, in a manner that is
15 transparent to the client or tenant application.

[0061] For example, in a multi-tenant environment, the system supports moving a pluggable database associated with a particular tenant, from a first Oracle Real Application Cluster (RAC) database, to a second RAC database; or from a first container database, to a second container database.

20 **[0062]** However, these pluggable databases generally operate as different/separate databases, which can result in connections being lost.

[0063] To address this, in accordance with an embodiment, in the case of an application that is currently using a connection string which points to a listener 212 of an original container database (e.g., “CDB-1”), the listener can be configured to redirect
25 connection requests to a new location or container database (e.g., “CDB-2”). This allows the listener to send a redirect to the database driver at the application server, which in turn causes the database driver to send the new connection requests to the new container database.

[0064] Additionally, existing connection requests must be drawn away from the original container database. However, the pool may not yet know about the existence of the new
30 container database, since it is considered a different database.

[0065] To address this, in accordance with an embodiment, a system event notification (e.g., an Oracle Notification Service event) can be used to inform the connection pool that the pluggable database is shutting down, and to close its associated connections and prepare for migration to a new database service associated with a new location.

35 **[0066]** Generally, there is a small period of time during which the new database location will not be immediately available to support new connections. During this time, existing connections will be closed, and the connection pool will not create a new connection until it

receives a new request. This can result in a slight system downtime, for example, due to the need to update redo logs, including stopping the redo logs to switch over the source of truth to the new location.

[0067] For example, in the example illustrated in Figure 4, in which it is desired to migrate a pluggable database (e.g., “PDB-1”), from a first container database (e.g., “CDB-1”), to a second container database (e.g., “CDB-2”); then, in accordance with an embodiment, the process involved in relocation of the pluggable database includes:

[0068] 1. Initiating relocation of the pluggable database. For example, as illustrated in Figure 5, the server can initiate relocation of a pluggable database by running an “alter pluggable database relocate” command, which will affect those sessions 220 running on the original pluggable database.

[0069] 2. Open the pluggable database at the new location, and then terminate all of the client sessions on the original instance container database. For example, as illustrated in Figure 6, the system can respond to the “alter pluggable database relocate” command by opening the pluggable database “PDB-1” in container database instance “CDB-2”, and then terminating all of the client sessions on the original container database instance “CDB-1”. After that, it will close the pluggable database “PDB-1” on “CDB-1”, and flush its buffer cache.

[0070] 3. Enable clients to reconnect to the new database location. For example, as illustrated in Figure 7, clients will then need to reconnect to the (now migrated) service themselves. The connection pool enables this in a transparent manner to the application, including, for example, as illustrated in Figure 8, by draining existing connections upon receiving a service down event from the server, and re-creating new connections to the migrated pluggable database.

[0071] 4. Forward connection requests to the new location. For example, as illustrated in Figure 9, on the server side, the listener will forward the new connection requests 228 from the connection pool to the new target container database (e.g., “CDB-2”) once the migration is complete. Applications do not need to change their connect string, which makes the relocation process transparent to the application.

Dynamic Relocation Process

[0072] Figure 10 illustrates a method of providing support for the dynamic relocation of a tenant, in a connection pool environment, in accordance with an embodiment.

[0073] As illustrated in Figure 10, in accordance with an embodiment, at step 231, at an application server or database environment, a connection pool logic or program code is provided that controls the creation and use of connection objects in a connection pool, wherein software applications can request a connection from the connection pool, and use a provided connection to access a database.

[0074] As illustrated in Figure 10, in accordance with an embodiment, at step 233, an instruction is received to migrate a pluggable database associated with a tenant, from a first container database instance, to a new location at a second container database instance.

[0075] At step 235, the server initiates relocation of the pluggable database, which affects those sessions running on the pluggable database.

[0076] At step 237, the system responds by opening the pluggable database at the new location, and then terminating all of the client sessions on the first container database instance.

[0077] At step 239, clients are enabled to reconnect to the (migrated) service associated with the new location.

[0078] At step 241, on the server side, a listener forwards new connection requests from the connection pool to the (new) container database location once the migration is complete.

[0079] Embodiments of the present invention may be conveniently implemented using one or more conventional general purpose or specialized digital computer, computing device, machine, or microprocessor, including one or more processors, memory and/or computer readable storage media programmed according to the teachings of the present disclosure. Appropriate software coding can readily be prepared by skilled programmers based on the teachings of the present disclosure, as will be apparent to those skilled in the software art.

[0080] In some embodiments, the present invention includes a computer program product which is a non-transitory storage medium or computer readable medium (media) having instructions stored thereon/in which can be used to program a computer to perform any of the processes of the present invention. Examples of the storage medium can include, but is not limited to, any type of disk including floppy disks, optical discs, DVD, CD-ROMs, microdrive, and magneto-optical disks, ROMs, RAMs, EPROMs, EEPROMs, DRAMs, VRAMs, flash memory devices, magnetic or optical cards, nanosystems (including molecular memory ICs), or any type of media or device suitable for storing instructions and/or data.

[0081] The foregoing description of embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Many modifications and variations will be apparent to the practitioner skilled in the art. The modifications and variations include any relevant combination of the disclosed features. The embodiments were chosen and described in order to best explain the principles of the invention and its practical application, thereby enabling others skilled in the art to understand the invention for various embodiments and with various modifications that are suited to the particular use contemplated.

Claims:

What is claimed is:

- 5 1. A system for providing access to a database in a multi-tenant environment, including the use of a connection pool, and support for dynamic relocation of tenants, comprising:
a computer including a processor, and at least one of an application server or database environment executing thereon;
a connection pool that enables software applications to request a connection from the
10 connection pool, and use a provided connection to access a database; and
wherein the connection pool enables a tenant associated with a client application, to be relocated across a plurality of database locations, including
controlling draining of connections to a database location originally associated
with the tenant, and
15 migrating connections to a new database location associated with the tenant.
2. The system of Claim 1, wherein during draining of existing connections, and migrating of new connections from a first pluggable database at a first container database, to a new location at a second container database,
20 a second pluggable database is opened at the second container database, and
client sessions are terminated on the first pluggable database, and are enabled to reconnect to a migrated service associated with the new location.
3. The system according to Claim 1 or 2, wherein a system event is used to inform the
25 connection pool that the database location originally associated with the tenant is shutting down, and to close associated connections and prepare for migration.
4. The system according to any preceding Claim, further comprising a listener configured to send a redirect to a database driver at the at least one of an application server or database
30 environment, to cause the database driver to send new connection requests to the new database location.
5. The system according to any preceding Claim, wherein the system enables software applications to associate particular labels with particular connection states.
35
6. The system according to any preceding Claim, wherein the connection pool supports a plurality of tenants, including a different database location associated with each tenant.

7. A method for providing access to a database in a multi-tenant environment, including the use of a connection pool, and support for dynamic relocation of tenants, comprising:

providing, at a computer including a processor, at least one of an application server or database environment executing thereon, a connection pool that includes connection objects and that enables software applications to request a connection from the connection pool, and use a provided connection to access a database; and

relocating, by the connection pool, a tenant associated with a client application, across a plurality of database locations, including

controlling draining of connections to a database location originally associated with the tenant, and

migrating connections to a new database location associated with the tenant.

8. The method of Claim 7, wherein during draining of existing connections, and migrating of new connections from a first pluggable database at a first container database, to a new location at a second container database,

a second pluggable database is opened at the second container database, and

client sessions are terminated on the first pluggable database, and are enabled to reconnect to a migrated service associated with the new location.

9. The method according to Claim 7 or 8, wherein a system event is used to inform the connection pool that the database location originally associated with the tenant is shutting down, and to close associated connections and prepare for migration.

10. The method according to any of Claims 7 to 9, further comprising providing a listener configured to send a redirect to a database driver at the at least one of an application server or database environment, to cause the database driver to send new connection requests to the new database location.

11. The method according to any of Claims 7 to 10, wherein software applications are enabled to associate particular labels with particular connection states.

12. The method according to any of Claims 7 to 11, wherein the connection pool supports a plurality of tenants, including a different database location associated with each tenant.

13. A non-transitory computer readable storage medium, including instructions stored thereon which when read and executed by one or more computers cause the one or more

computers to perform the method comprising:

providing, at a computer including a processor, at least one of an application server or database environment executing thereon, a connection pool that includes connection objects and that enables software applications to request a connection from the connection pool, and
5 use a provided connection to access a database; and

relocating, by the connection pool, a tenant associated with a client application, across a plurality of database locations, including

controlling draining of connections to a database location originally associated with the tenant, and

10 migrating connections to a new database location associated with the tenant.

14. The non-transitory computer readable storage medium of Claim 13, wherein during draining of existing connections, and migrating of new connections from a first pluggable database at a first container database, to a new location at a second container database,

15 a second pluggable database is opened at the second container database, and client sessions are terminated on the first pluggable database, and are enabled to reconnect to a migrated service associated with the new location.

15. The non-transitory computer readable storage medium according to Claim 13 or 14,
20 wherein a system event is used to inform the connection pool that the database location originally associated with the tenant is shutting down, and to close associated connections and prepare for migration.

16. The non-transitory computer readable storage medium according to any of Claims 13
25 to 15, further comprising providing a listener configured to send a redirect to a database driver at the at least one of an application server or database environment, to cause the database driver to send new connection requests to the new database location.

17. The non-transitory computer readable storage medium according to any of Claims 13
30 to 16, wherein software applications are enabled to associate particular labels with particular connection states.

18. The non-transitory computer readable storage medium according to any of Claims 13
35 to 17, wherein the connection pool supports a plurality of tenants, including a different database location associated with each tenant.

19. A computer program comprising program instructions in machine-readable format that

when executed by a computer system cause the computer system to perform the method of any of Claims 7 to 12.

20. A computer program product comprising the computer program of Claim 19 stored in a
5 non-transitory machine readable data storage medium.

21. An apparatus comprising means for performing the method of any of Claims 7 to 12.

10

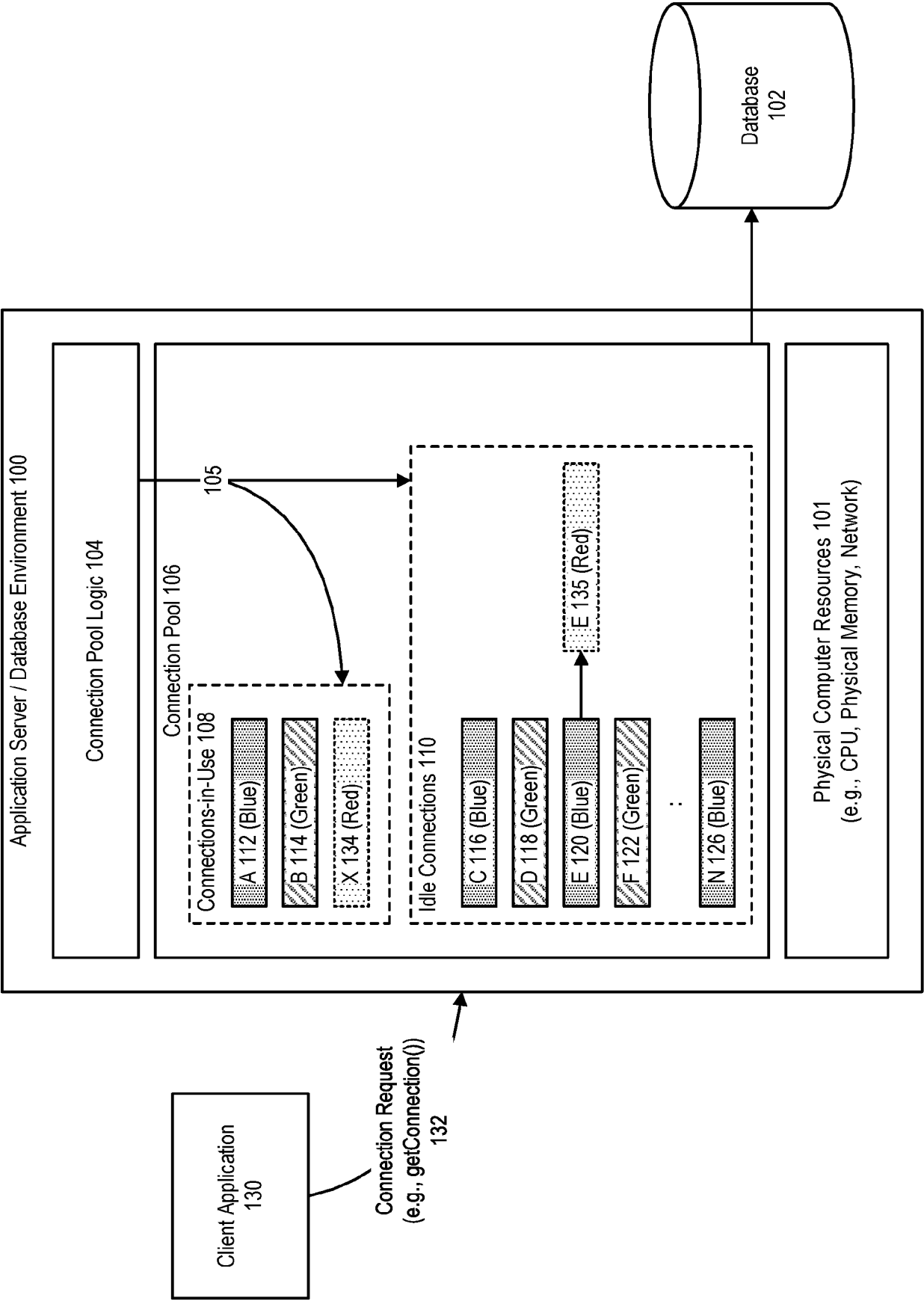


FIGURE 1

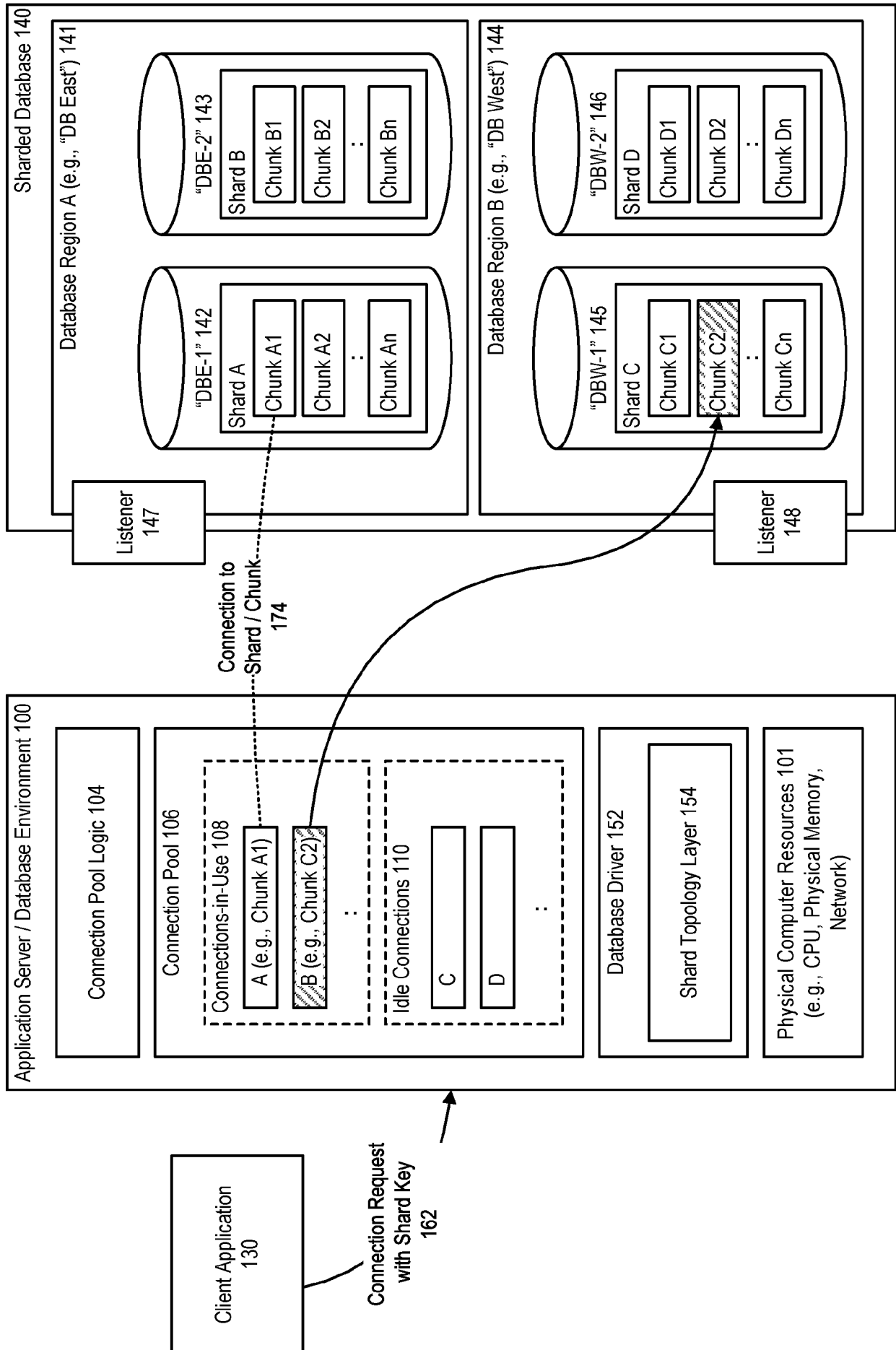


FIGURE 2

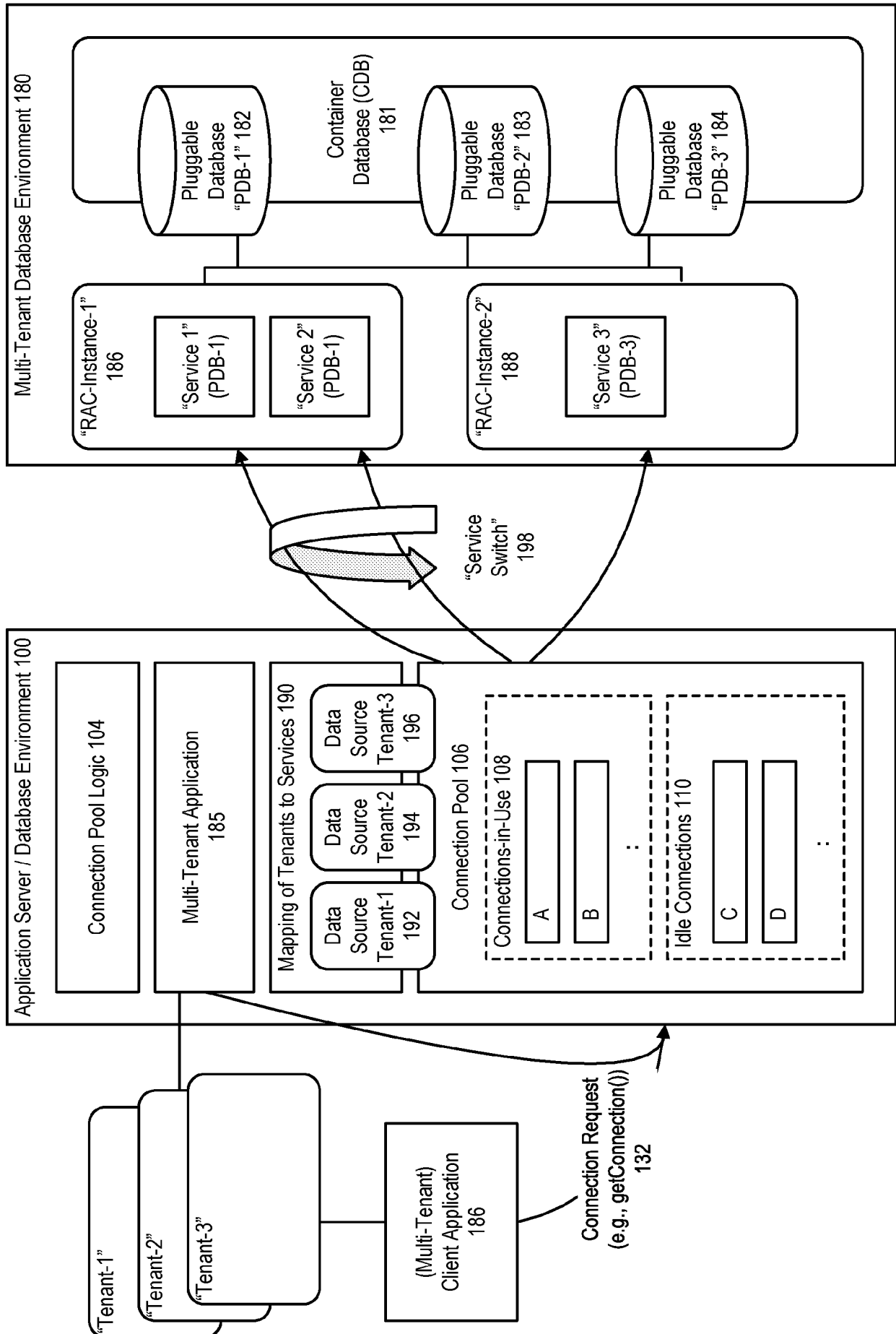


FIGURE 3

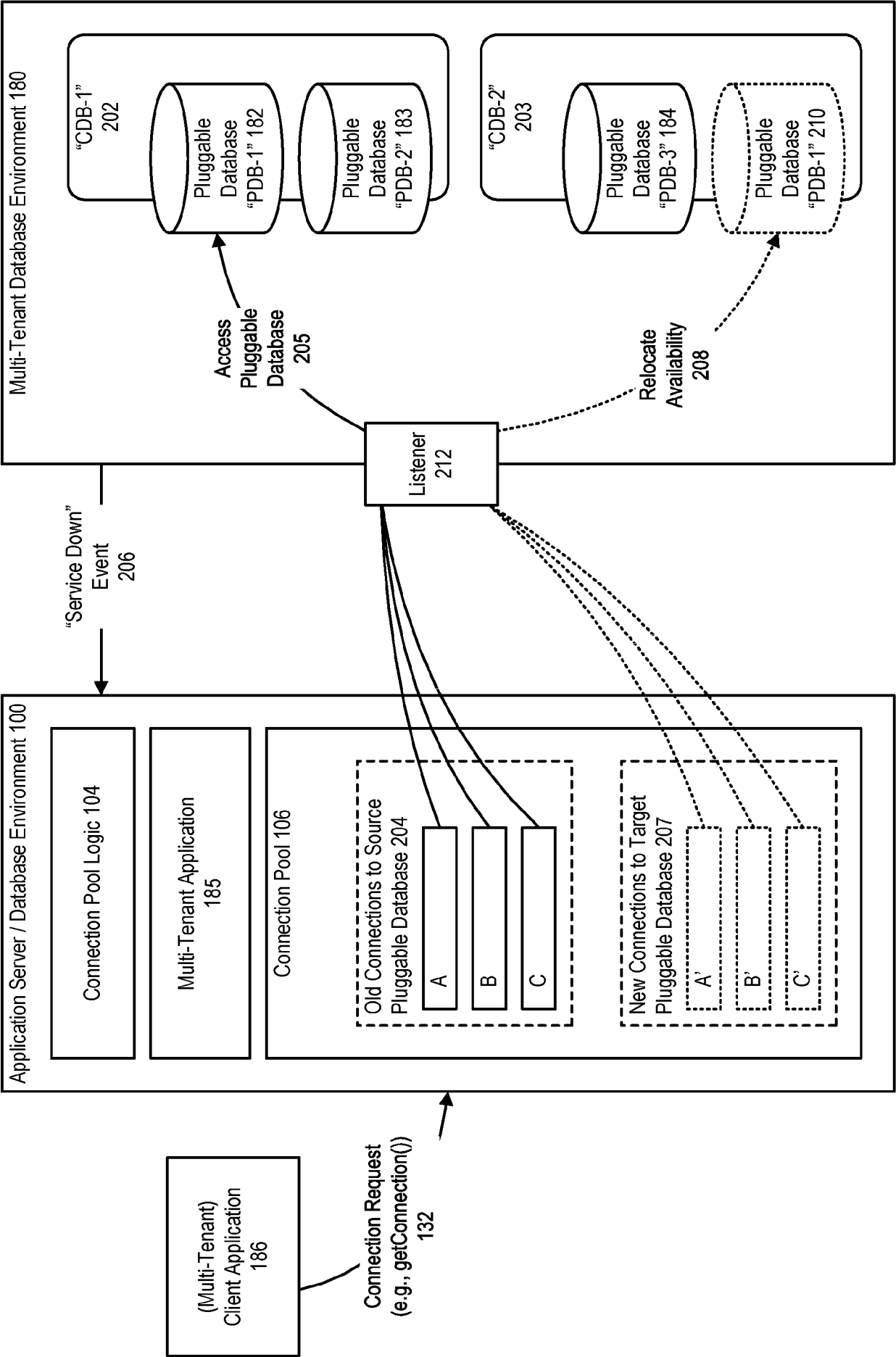


FIGURE 4

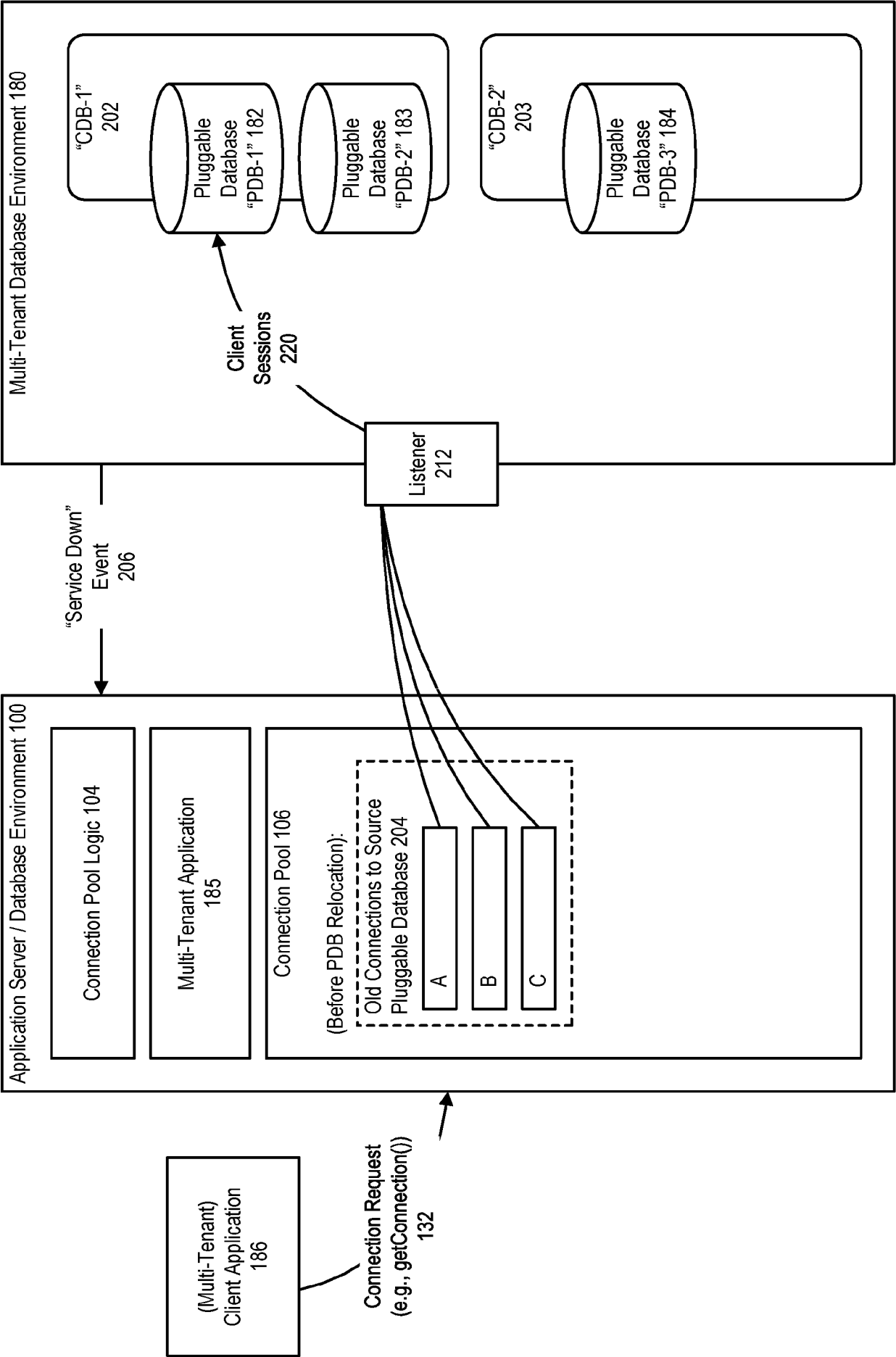


FIGURE 5

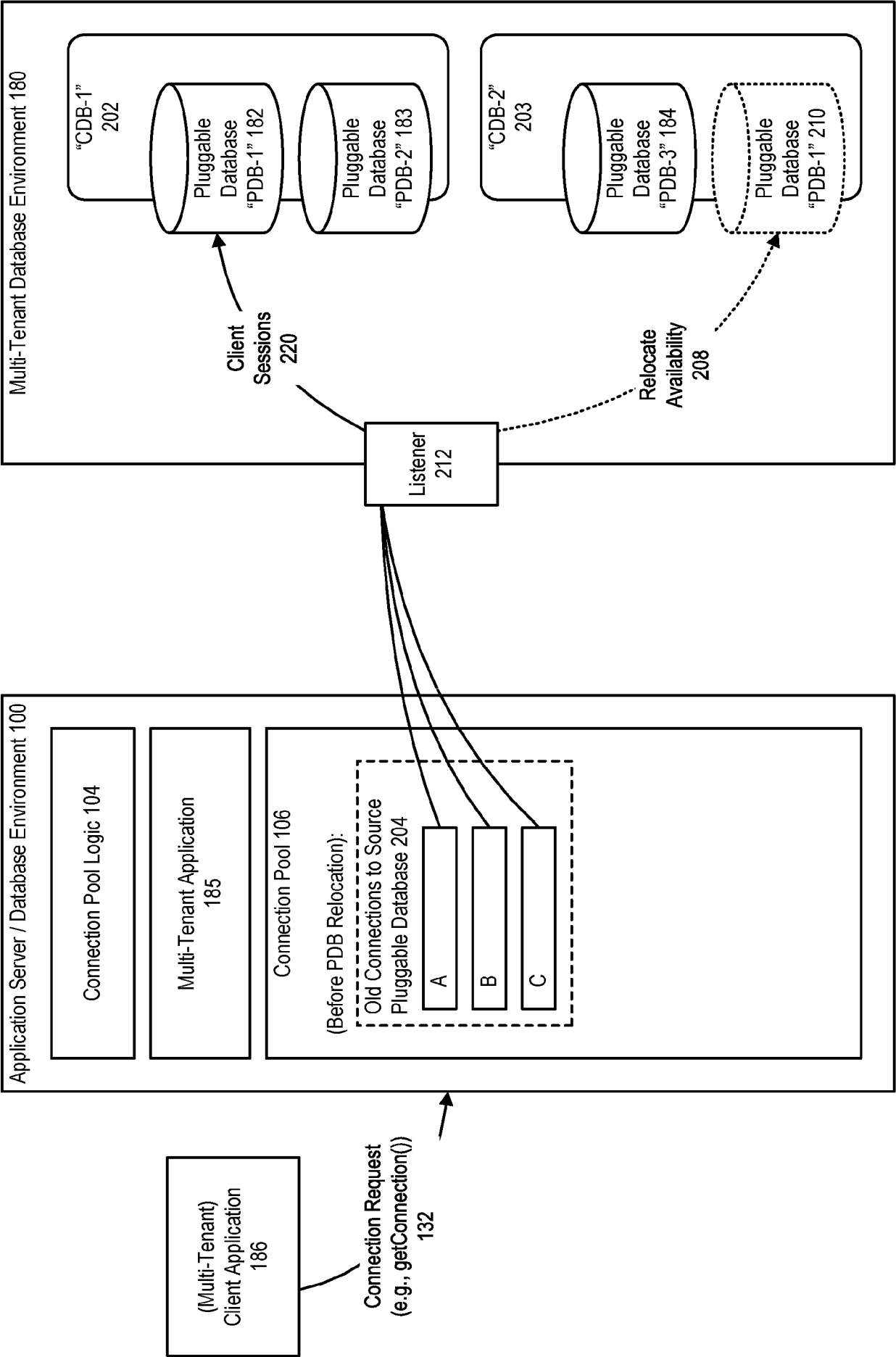


FIGURE 6

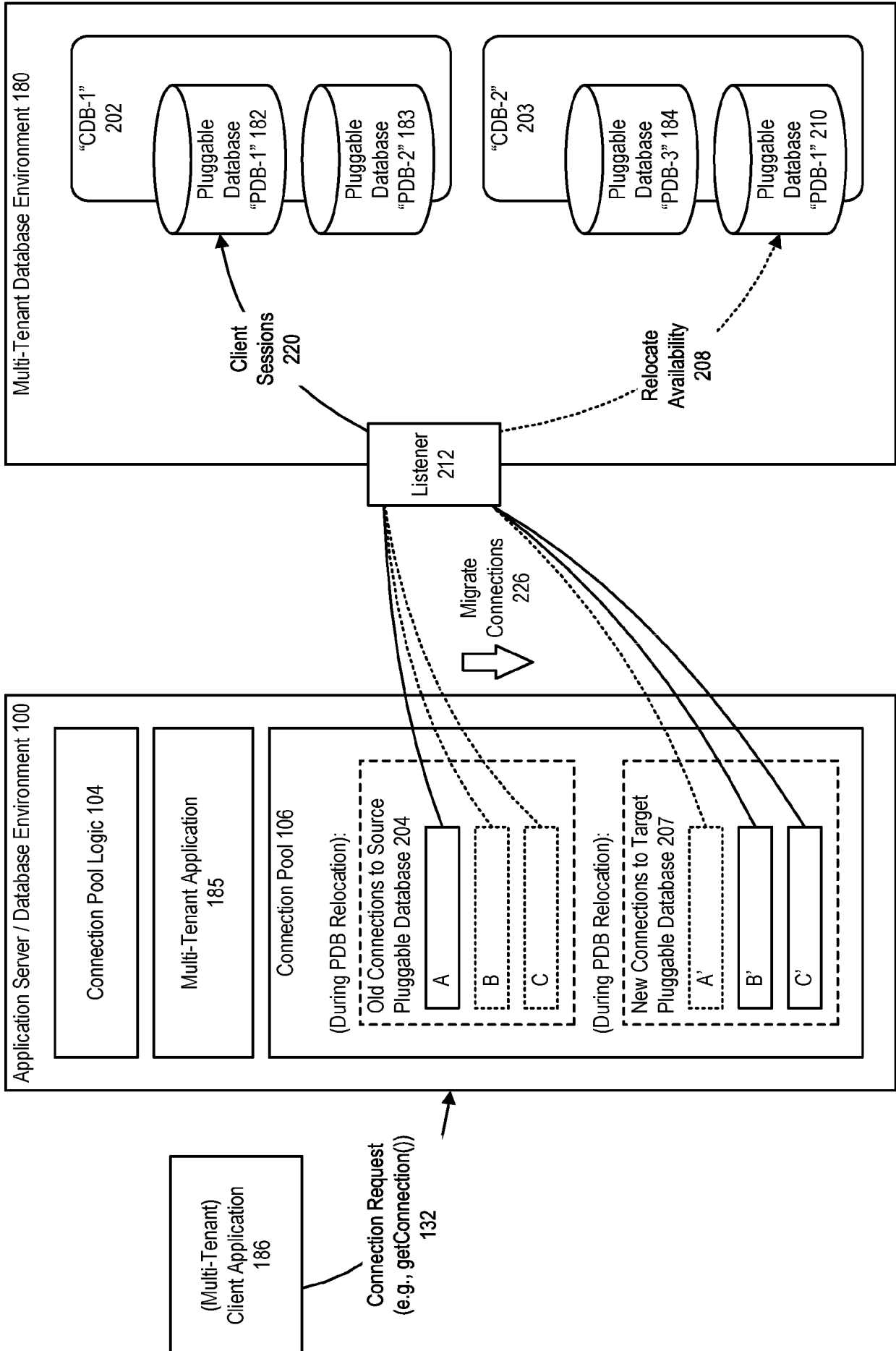


FIGURE 7

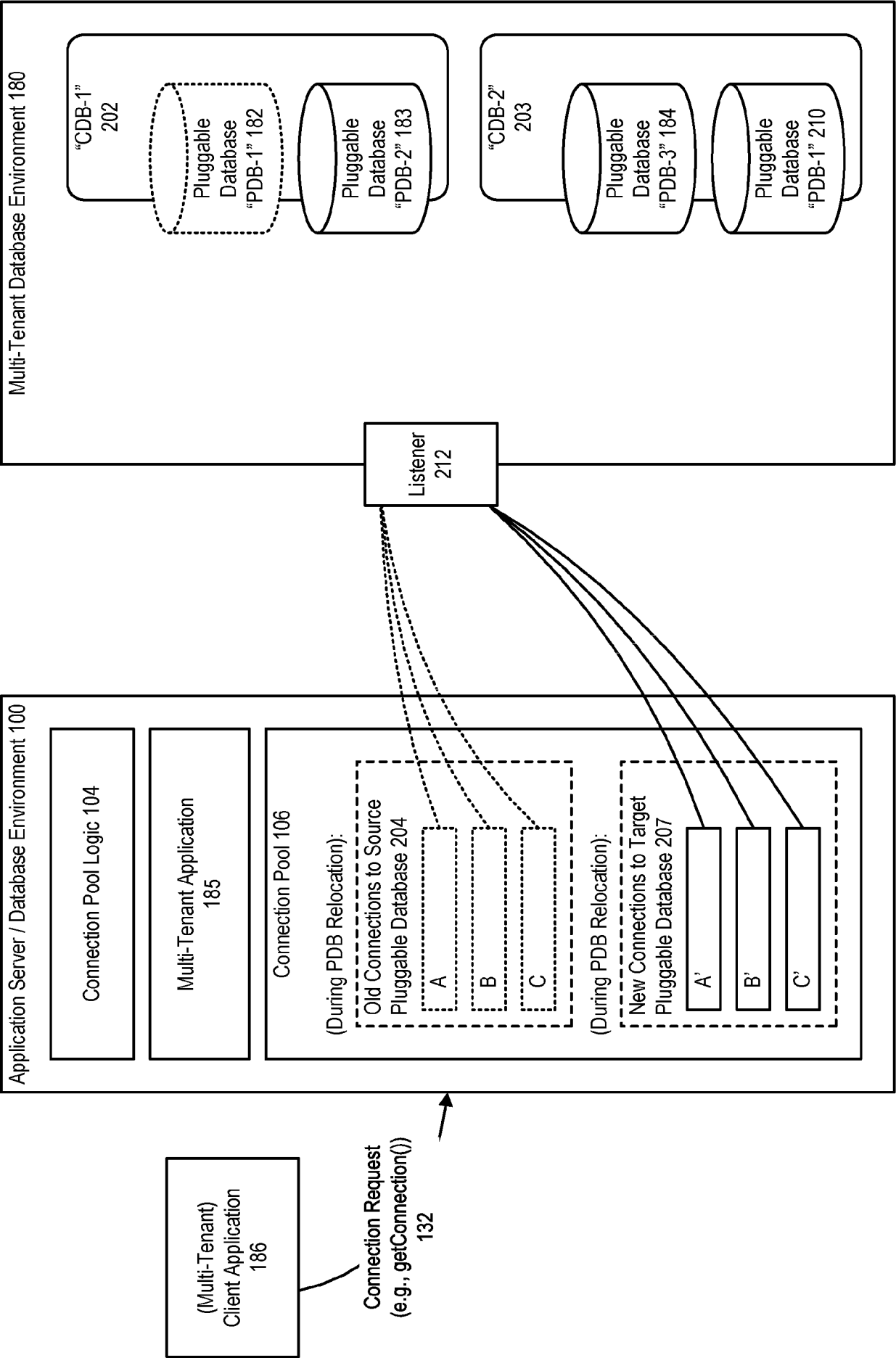


FIGURE 8

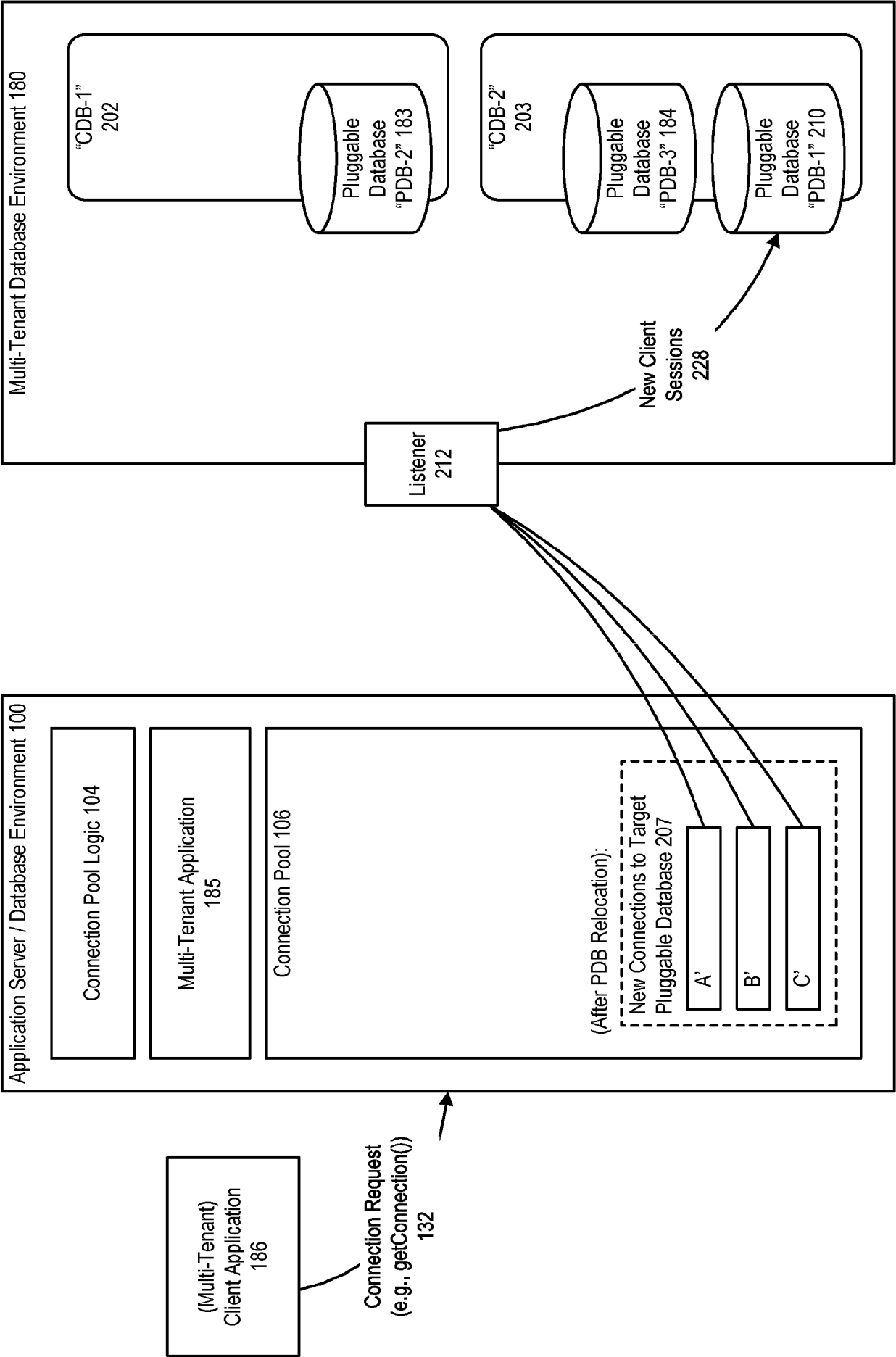


FIGURE 9

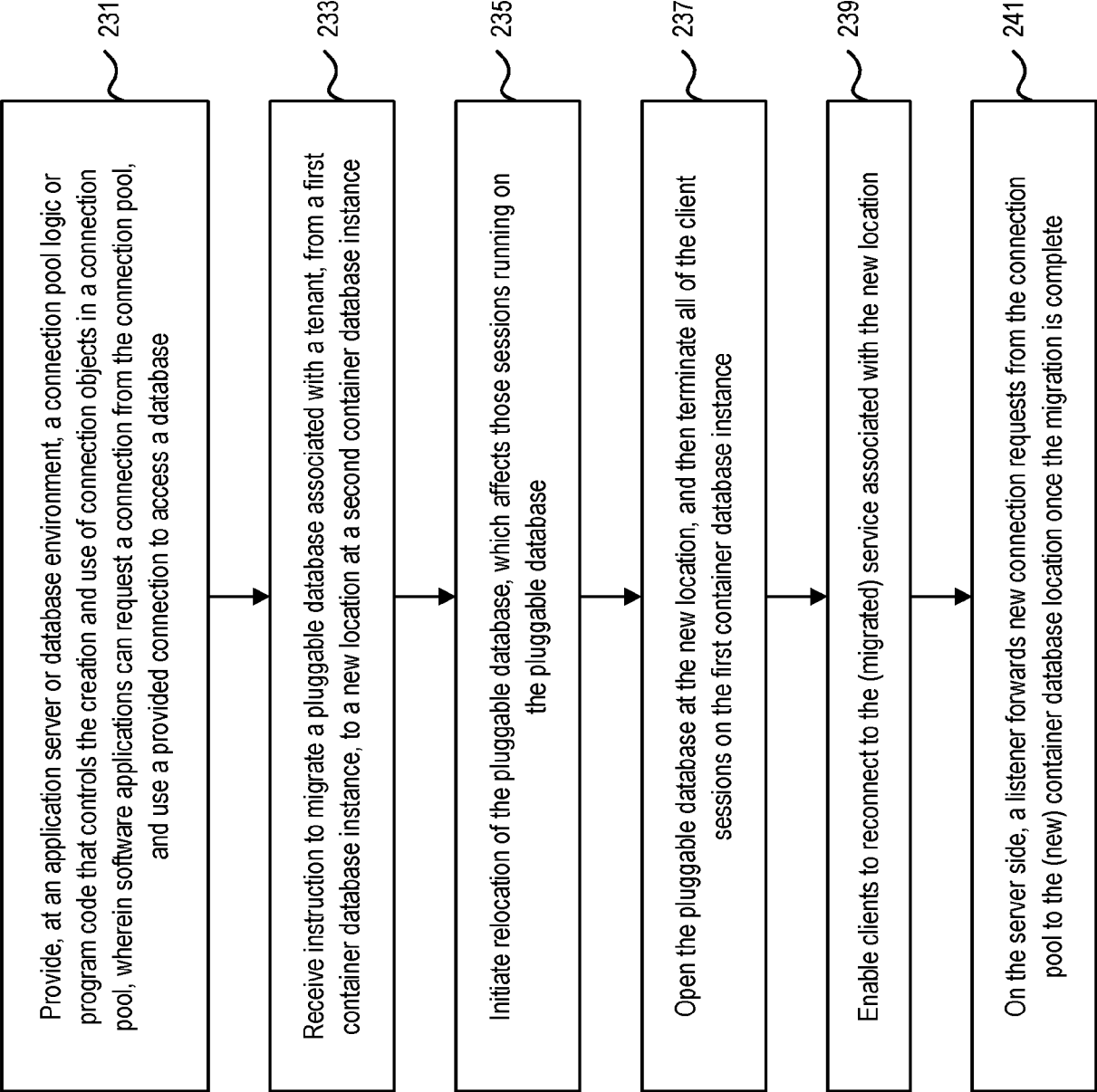


FIGURE 10

INTERNATIONAL SEARCH REPORT

International application No

PCT/US2017/045283

A. CLASSIFICATION OF SUBJECT MATTER
 INV. G06F9/50 H04L29/08
 ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

G06F H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

EPO-Internal, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	Nicolas Michael ET AL: "Downtime-Free Live Migration in a Multitenant Database" In: "Network and Parallel Computing", 1 January 2015 (2015-01-01), Springer International Publishing, Cham 032548, XP055417846, ISSN: 0302-9743 ISBN: 978-3-642-27168-7 vol. 8904, pages 130-155, DOI: 10.1007/978-3-319-15350-6_9, page 6 - page 7 -----	1-21
A	US 6 199 110 B1 (RIZVI HASAN [US] ET AL) 6 March 2001 (2001-03-06) the whole document -----	1-21
A	US 2013/066955 A1 (NEEL KEVIN S [US] ET AL) 14 March 2013 (2013-03-14) the whole document -----	1-21



Further documents are listed in the continuation of Box C.



See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

25 October 2017

Date of mailing of the international search report

07/11/2017

Name and mailing address of the ISA/

European Patent Office, P.B. 5818 Patentlaan 2
 NL - 2280 HV Rijswijk
 Tel. (+31-70) 340-2040,
 Fax: (+31-70) 340-3016

Authorized officer

Alecu, Mihail

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No

PCT/US2017/045283

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 6199110	B1	06-03-2001	NONE

US 2013066955	A1	14-03-2013	CN 103782573 A 07-05-2014
			EP 2754283 A1 16-07-2014
			EP 2903239 A1 05-08-2015
			US 2013066955 A1 14-03-2013
			US 2014229531 A1 14-08-2014
			US 2015326673 A1 12-11-2015
			WO 2013036882 A1 14-03-2013
