Transaction Processing in a Distributed System

A transaction is a logical unit of work constituted by one or more SQL statements executed by a single user. A transaction begins with the user's first executable SQL statement and ends when it is committed or rolled back by that user.

A **remote transaction** contains only statements that access a single remote node. A **distributed transaction** contains statements that access multiple nodes.

The following sections define important concepts in transaction processing and explain how transactions access data in a distributed database:

Remote SQL Statements

A **remote query** statement is a query that selects information from one or more remote tables, all of which reside at the same remote node. For example, the following query accesses data from the `dept` table in the `scott` schema of the remote `sales` database:

```
SELECT * FROM scott.dept@sales.us.americas.example_auto.com;
```

A **remote update** statement is an update that modifies data in one or more tables, all of which are located at the same remote node. For example, the following query updates the `dept` table in the `scott` schema of the remote `sales` database:

```
UPDATE scott.dept@mktng.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
```

> **Note:**
> A remote update can include a subquery that retrieves data from one or more remote nodes, but because the update happens at only a single remote node, the statement is classified as a remote update.

Distributed SQL Statements

A **distributed query** statement retrieves information from two or more nodes. For example, the following query accesses data from the local database as well as the remote `sales` database:

```
SELECT ename, dname
  FROM scott.emp e, scott.dept@sales.us.americas.example_auto.com d
  WHERE e.deptno = d.deptno;
```

A **distributed update** statement modifies data on two or more nodes. A distributed update is possible using a PL/SQL subprogram unit such as a procedure or trigger that includes two or more remote updates that access data on different nodes. For example, the following PL/SQL program unit updates tables on the local database and the remote `sales` database:

```
BEGIN
  UPDATE scott.dept@sales.us.americas.example_auto.com
    SET loc = 'NEW YORK'
    WHERE deptno = 10;
  UPDATE scott.emp
    SET deptno = 11
    WHERE deptno = 10;
END;
COMMIT;
```

The database sends statements in the program to the remote nodes, and their execution succeeds or fails as a unit.

Shared SQL for Remote and Distributed Statements

The mechanics of a remote or distributed statement using shared SQL are essentially the same as those of a local statement. The SQL text must match, and the referenced objects must match. If available, shared SQL areas can be used for the local and remote handling of any statement or decomposed query.

See Also:
Oracle Database Concepts Concepts for more information about shared SQL

Remote Transactions

A remote transaction contains one or more remote statements, all of which reference a single remote node. For example, the following transaction contains two statements, each of which accesses the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp@sales.us.americas.example_auto.com
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

Distributed Transactions

A distributed transaction is a transaction that includes one or more statements that, individually or as a group, update data on two or more distinct nodes of a distributed database. For example, this transaction updates the local database and the remote `sales` database:

```
UPDATE scott.dept@sales.us.americas.example_auto.com
  SET loc = 'NEW YORK'
  WHERE deptno = 10;
UPDATE scott.emp
  SET deptno = 11
  WHERE deptno = 10;
COMMIT;
```

> **Note:**
> If all statements of a transaction reference only a single remote node, the transaction is remote, not distributed.

Two-Phase Commit Mechanism

A database must guarantee that all statements in a transaction, distributed or non-distributed, either commit or roll back as a unit. The effects of an ongoing transaction should be invisible to all other transactions at all nodes; this transparency should be true for transactions that include any type of operation, including queries, updates, or remote procedure calls.

The general mechanisms of transaction control in a non-distributed database are discussed in the *Oracle Database Concepts Concepts*. In a distributed database, the database must coordinate transaction control with the same characteristics over a network and maintain data consistency, even if a network or system failure occurs.

The database **two-phase commit** mechanism guarantees that *all* database servers participating in a distributed transaction either all commit or all roll back the statements in the transaction. A two-phase commit mechanism also protects implicit DML operations performed by integrity constraints, remote procedure calls, and triggers.

> **See Also:**
> Chapter 34, "Distributed Transactions Concepts" for more information about the Oracle Database two-phase commit mechanism

Database Link Name Resolution

A **global object name** is an object specified using a database link. The essential components of a global object name are:

- Object name
- Database name
- Domain

The following table shows the components of an explicitly specified global database object name:

Statement	Object	Database	Domain

Statement	Object	Database	Domain
`SELECT * FROM joan.dept@sales.example.com`	`dept`	`sales`	`example.com`
`SELECT * FROM emp@mktg.us.example.com`	`emp`	`mktg`	`us.example.com`

Whenever a SQL statement includes a reference to a global object name, the database searches for a database link with a name that matches the database name specified in the global object name. For example, if you issue the following statement:

```
SELECT * FROM scott.emp@orders.us.example.com;
```

The database searches for a database link called `orders.us.example.com`. The database performs this operation to determine the path to the specified remote database.

The database always searches for matching database links in the following order:

1. Private database links in the schema of the user who issued the SQL statement.
2. Public database links in the local database.
3. Global database links (only if a directory server is available).

Name Resolution When the Global Database Name Is Complete

Assume that you issue the following SQL statement, which specifies a complete global database name:

```
SELECT * FROM emp@prod1.us.example.com;
```

In this case, both the database name (`prod1`) and domain components (`us.example.com`) are specified, so the database searches for private, public, and global database links. The database searches only for links that match the specified global database name.

Name Resolution When the Global Database Name Is Partial

If any part of the domain is specified, the database assumes that a complete global database name is specified. If a SQL statement specifies a partial global database name (that is, only the database component is specified), the database appends the value in the `DB_DOMAIN` initialization parameter to the value in the `DB_NAME` initialization parameter to construct a complete name. For example, assume you issue the following statements:

```
CONNECT scott@locdb
SELECT * FROM scott.emp@orders;
```

If the network domain for `locdb` is `us.example.com`, then the database appends this domain to `orders` to construct the complete global database name of `orders.us.example.com`. The database searches for database links that match only the constructed global name. If a matching link is not found, the database returns an error and the SQL statement cannot execute.

Name Resolution When No Global Database Name Is Specified

If a global object name references an object in the local database and a database link name is *not* specified using the @ symbol, then the database automatically detects that the object is local and does not search for or

use database links to resolve the object reference. For example, assume that you issue the following statements:

```
CONNECT scott@locdb
SELECT * from scott.emp;
```

Because the second statement does not specify a global database name using a database link connect string, the database does not search for database links.

Terminating the Search for Name Resolution

The database does not necessarily stop searching for matching database links when it finds the first match. The database must search for matching private, public, and network database links until it determines a complete path to the remote database (both a remote account and service name).

The first match determines the remote schema as illustrated in the following table:

User Operation	Database Response	Example
Do *not* specify the `CONNECT` clause	Uses a connected user database link	`CREATE DATABASE LINK k1 USING 'prod'`
Do specify the `CONNECT TO ... IDENTIFIED BY` clause	Uses a fixed user database link	`CREATE DATABASE LINK k2 CONNECT TO scott IDENTIFIED BY` *password* `USING 'prod'`
Specify the `CONNECT TO CURRENT_USER` clause	Uses a current user database link	`CREATE DATABASE LINK k3 CONNECT TO CURRENT_USER USING 'prod'`
Do *not* specify the `USING` clause	Searches until it finds a link specifying a database string. If matching database links are found and a string is never identified, the database returns an error.	`CREATE DATABASE LINK k4 CONNECT TO CURRENT_USER`

After the database determines a complete path, it creates a remote session, assuming that an identical connection is not already open on behalf of the same local session. If a session already exists, the database reuses it.

Schema Object Name Resolution

After the local Oracle Database connects to the specified remote database on behalf of the local user that issued the SQL statement, object resolution continues as if the remote user had issued the associated SQL statement. The first match determines the remote schema according to the following rules:

Type of Link Specified	Location of Object Resolution
A fixed user database link	Schema specified in the link creation statement
A connected user database link	Connected user's remote schema
A current user database link	Current user's schema

If the database cannot find the object, then it checks public objects of the remote database. If it cannot resolve the object, then the established remote session remains but the SQL statement cannot execute and returns an error.

The following are examples of global object name resolution in a distributed database system. For all the following examples, assume that:

Example of Global Object Name Resolution: Complete Object Name

This example illustrates how the database resolves a complete global object name and determines the appropriate path to the remote database using both a private and public database link. For this example, assume the following:

- The remote database is named `sales.division3.example.com`.
- The local database is named `hq.division3.example.com`.
- A directory server (and therefore, global database links) is not available.
- A remote table `emp` is contained in the schema `tsmith`.

Consider the following statements issued by `scott` at the local database:

```
CONNECT scott@hq

CREATE PUBLIC DATABASE LINK sales.division3.example.com
CONNECT TO guest IDENTIFIED BY network
  USING 'dbstring';
```

Later, `JWARD` connects and issues the following statements:

```
CONNECT jward@hq

CREATE DATABASE LINK sales.division3.example.com
  CONNECT TO tsmith IDENTIFIED BY radio;

UPDATE tsmith.emp@sales.division3.example.com
  SET deptno = 40
  WHERE deptno = 10;
```

The database processes the final statement as follows:

1. The database determines that a complete global object name is referenced in `jward`'s `UPDATE` statement. Therefore, the system begins searching in the local database for a database link with a matching name.
2. The database finds a matching private database link in the schema `jward`. Nevertheless, the private database link `jward.sales.division3.example.com` does not indicate a complete path to the remote `sales` database, only a remote account. Therefore, the database now searches for a matching public database link.
3. The database finds the public database link in `scott`'s schema. From this public database link, the database takes the service name `dbstring`.
4. Combined with the remote account taken from the matching private fixed user database link, the database determines a complete path and proceeds to establish a connection to the remote `sales` database as user `tsmith/radio`.
5. The remote database can now resolve the object reference to the `emp` table. The database searches in the `tsmith` schema and finds the referenced `emp` table.
6. The remote database completes the execution of the statement and returns the results to the local database.

Example of Global Object Name Resolution: Partial Object Name

This example illustrates how the database resolves a partial global object name and determines the appropriate path to the remote database using both a private and public database link.

For this example, assume that:

- The remote database is named `sales.division3.example.com`.
- The local database is named `hq.division3.example.com`.
- A directory server (and therefore, global database links) is not available.
- A table `emp` on the remote database `sales` is contained in the schema `tsmith`, but not in schema `scott`.
- A public synonym named `emp` resides at remote database `sales` and points to `tsmith.emp` in the remote database `sales`.
- The public database link in "Example of Global Object Name Resolution: Complete Object Name" is already created on local database `hq`:

  ```
  CREATE PUBLIC DATABASE LINK sales.division3.example.com
    CONNECT TO guest IDENTIFIED BY network
    USING 'dbstring';
  ```

Consider the following statements issued at local database `hq`:

```
CONNECT scott@hq

CREATE DATABASE LINK sales.division3.example.com;

DELETE FROM emp@sales
  WHERE empno = 4299;
```

The database processes the final `DELETE` statement as follows:

1. The database notices that a partial global object name is referenced in `scott`'s `DELETE` statement. It expands it to a complete global object name using the domain of the local database as follows:

   ```
   DELETE FROM emp@sales.division3.example.com
     WHERE empno = 4299;
   ```

2. The database searches the local database for a database link with a matching name.
3. The database finds a matching *private* connected user link in the schema `scott`, but the private database link indicates no path at all. The database uses the connected username/password as the remote account portion of the path and then searches for and finds a matching *public* database link:

   ```
   CREATE PUBLIC DATABASE LINK sales.division3.example.com
     CONNECT TO guest IDENTIFIED BY network
     USING 'dbstring';
   ```

4. The database takes the database net service name `dbstring` from the public database link. At this point, the database has determined a complete path.
5. The database connects to the remote database as `scott/`*`password`* and searches for and does not find an object named `emp` in the schema `scott`.
6. The remote database searches for a public synonym named `emp` and finds it.

7. The remote database executes the statement and returns the results to the local database.

Global Name Resolution in Views, Synonyms, and Procedures

A view, synonym, or PL/SQL program unit (for example, a procedure, function, or trigger) can reference a remote schema object by its global object name. If the global object name is complete, then the database stores the definition of the object without expanding the global object name. If the name is partial, however, the database expands the name using the domain of the local database name.

The following table explains when the database completes the expansion of a partial global object name for views, synonyms, and program units:

User Operation	**Database Response**
Create a view	Does *not* expand partial global names. The data dictionary stores the exact text of the defining query. Instead, the database expands a partial global object name each time a statement that uses the view is parsed.
Create a synonym	Expands partial global names. The definition of the synonym stored in the data dictionary includes the expanded global object name.
Compile a program unit	Expands partial global names.

What Happens When Global Names Change

Global name changes can affect views, synonyms, and procedures that reference remote data using partial global object names. If the global name of the referenced database changes, views and procedures may try to reference a nonexistent or incorrect database. However, synonyms do not expand database link names at run time, so they do not change.

Scenarios for Global Name Changes

For example, consider two databases named `sales.uk.example.com` and `hq.uk.example.com`. Also, assume that the `sales` database contains the following view and synonym:

```
CREATE VIEW employee_names AS
        SELECT ename FROM scott.emp@hr;

CREATE SYNONYM employee FOR scott.emp@hr;
```

The database expands the `employee` synonym definition and stores it as:

```
scott.emp@hr.uk.example.com
```

Scenario 1: Both Databases Change Names

First, consider the situation where both the Sales and Human Resources departments are relocated to the United States. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` becomes `hq.us.example.com`

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.example.com	SELECT * FROM scott.emp@hr.us.example.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.example.com	SELECT * FROM scott.emp@hr.uk.example.com

Scenario 2: One Database Changes Names

Now consider that only the Sales department is moved to the United States; Human Resources remains in the UK. Consequently, the corresponding global database names are both changed as follows:

- `sales.uk.example.com` becomes `sales.us.example.com`
- `hq.uk.example.com` is not changed

The following table describes query expansion before and after the change in global names:

Query on sales	Expansion Before Change	Expansion After Change
SELECT * FROM employee_names	SELECT * FROM scott.emp@hr.uk.example.com	SELECT * FROM scott.emp@hr.us.example.com
SELECT * FROM employee	SELECT * FROM scott.emp@hr.uk.example.com	SELECT * FROM scott.emp@hr.uk.example.com

In this case, the defining query of the `employee_names` view expands to a nonexistent global database name. However, the `employee` synonym continues to reference the correct database, `hq.uk.example.com`.