第16章　分布式数据库管理

分布式数据库结构基于第13章所述的服务器/服务器配置。在一个分布式环境中，不同服务器(主机)上的数据库彼此共享数据。每个服务器既能在物理上分离，又可以保持着彼此间的逻辑关系。

典型的分布式数据库结构是：团体总部的服务器与不同地方的部门服务器相互通信。每个服务器都支持客户应用程序，但同时也能够与网络中的其他服务器通信。图 16-1说明了这种结构。

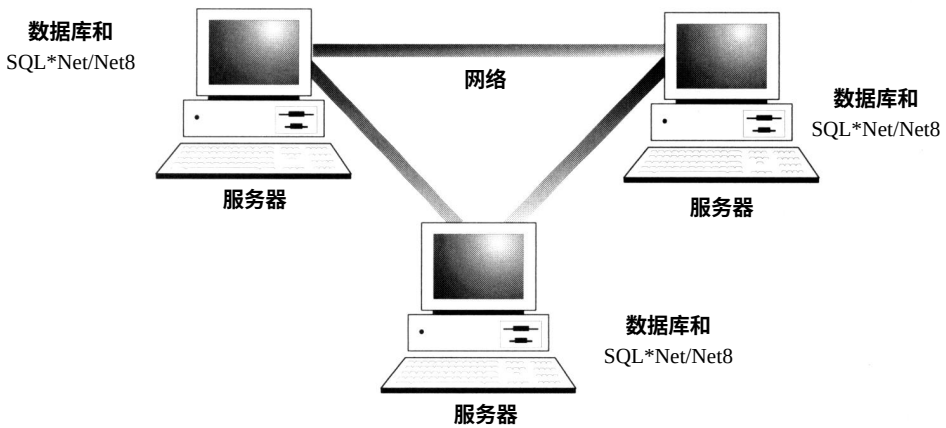


图16-1　服务器/服务器结构

当一个服务器向另一个服务器发送数据库请求时，发送端服务器充当客户机。接收端服务器执行传递给它的SQL语句，并且向发送端返回结果及错误信息。

SQL*Net V2和Net8使这一结构成为现实。当在所有服务器上运行时， SQL*Net和Net8允许一个数据库(或一个应用程序)提出的数据库请求传递到其他服务器的数据库中，并且同时支持分布式查询和分布式更新。

通过这一功能，就可以与能通过网络进行访问的数据库进行通信。然后可以创建同义词，从而使网络对应用程序完全透明，提交查询的用户无需知道用来解决查询的数据的位置。

16.1　远程查询

执行远程查询的能力只是分布式数据库的功能之一，并且它的用途有限。当数据在逻辑和物理上都分离时远程查询才适合需要。也就是说，可以把数据的“拥有权”赋予一个数据库，并且在数据库间不存在数据相关性。

要查询远程数据库，必须创建一个数据库链接。该数据库链接规定要使用的数据库名，也可以规定连接远程数据库的用户名。当一个数据库链接被 SQL语句引用时，Oracle会在远程数据库打开一个会话，并且在那里执行 SQL语句。随后数据被返回，并且远程会话保持打开

状态，以供下次使用。数据被链接可以被指定为公用链接 (由数据库管理人员创建，使该链接对本地数据库中的所有用户可用)或私有链接。

下面是创建一个名为HR_LINK的公用数据库链接的例子：

```
create public database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';
```

例子中所示的create database link命令有下面一些参数：

- 任选的关键字public，它允许数据库管理人员创建对数据库全体用户可用的数据库链接。
- 链接名(本例中为HR_LINK)。
- 要连接的帐户，可以配置数据库链接以便把本地用户名和口令用在远程数据库中。
- 服务名(hq)。

若要使用新创建的链接，只需在命令中将它作为后缀加到表名之后。下面的例子中，通过使用HR_LINK数据库链接来查询一个远程数据表：

```
select * from EMPLOYEE@HR_LINK
 where Office='ANNAPOLIS';
```

执行这个查询时，Oracle通过HR_LINK数据库链接建立一个会话，并且查询这个数据库的EMPLOYEE表。其中的where子句将应用到EMPLOYEE的行中，并且返回符合条件的记录。图16-2示出了这一查询的执行情况。

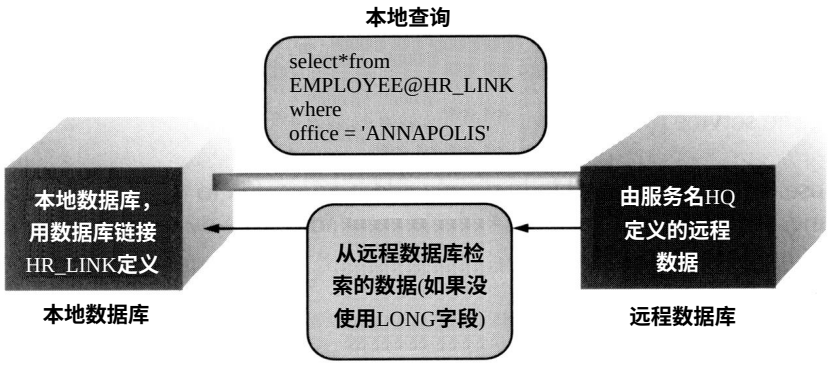


图16-2　远程查询样例

注意　数据库链接不能用于从LONG数据类型的列中返回值。

上面例子中的from子句引用EMPLOYEE@HR_LINK。由于HR_LINK数据库链接规定服务器名、实例名和拥有者名，所以表的全名是已知的。如果在数据库链接中并未指定帐户名，在试图注册远程数据库时，将使用本地数据库中的用户帐户名及口令。

有关数据库链接管理的详细情况，在本章 16.4节“ 分布式数据管理 ”中描述。

16.2　远程数据操作：两阶段提交

实现远程数据操作，需要使用 Two-Phase Commit(2PC,两阶段提交)——这也是Oracle分布式数据库的功能本质。 2PC可以把几个节点间的事务组看作是一个单元；或者是所有事务都提交，或者是它们都回滚。图 16-3示出了一个分布式事务集。图中，执行两个 update(更新)事务。第一个update事务针对本地数据表 (EMPLOYEE）；第二个 update事务则针对远程数据库表

(EMPLOYEE@HR_LINK)。执行完这两个事务之后，便执行一个提交。如果有一个事务不能被提交，则这两个事务都被回滚。

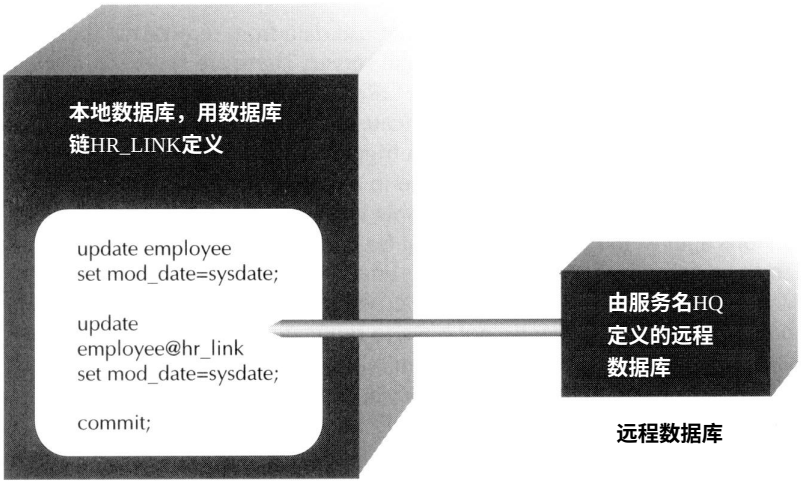


图16-3 分布式事务样例

分布式事务会带来两个重要的好处：其他服务器上的数据库可以被更新，并且这些事务可以与其他事务组成一个逻辑单元。第二个好处在于数据库对 2PC的使用。这两个阶段是：

- 准备阶段：称作全局协调程序 (global coordinator)的初始化节点通知事务所涉及的所有站点做好提交或回滚事务的准备。
- 提交阶段：如果准备阶段没有问题，则所有站点提交它们的事务。如果出现一个网络或节点失败，则所有站点回滚它们的事务。

2PC的使用对用户透明，有关分布式事务的管理细节，在本章后面 16.5节“分布式事务管理”中讨论。

16.3 动态数据复制

若要改善使用远程数据库数据的查询性能，可能需要在本地服务器上复制相关数据。用于这种数据复制的选项有几种，使用哪一种选项取决于所使用的 Oracle功能。

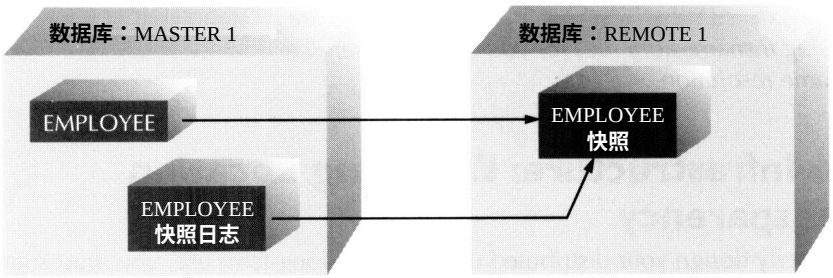
可以使用数据库触发器 (database trigger)从一个表向另一个表复制数据。例如，在每一次对数据表进行 insert(插入)操作之后，就启动触发器在另一个数据表 —— 这个表可能在一个远程数据库中 —— 插入相同的记录。因此，在简单的配置中，可以用触发器强制进行数据复制。如果不能控制针对基表的事务类型，则执行复制需要的触发器代码将十分复杂。

当使用Oracle分布式特性时，可以使用快照 (snapshot)在数据库间复制数据。这样就用不着复制整个表或限定只可从一个表中获取数据。在复制一个表时，可以使用 where子句来限定复制的记录，并且可以对数据执行 group by操作。也可以把表与其他表连结并复制查询的结果。

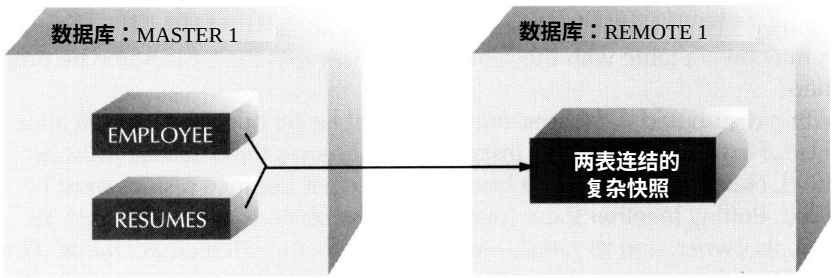
注意 不能用快照复制使用LONG、LONG RAW、BFILE或抽象数据库类型的数据。

远程数据表的本地快照中的数据需要被刷新。可以指定快照的刷新时间周期，并且数据

库会自动管理复制过程。如果快照是远程数据表记录的一对一复制（称作simple snapshot），则数据库可以使用快照日志（snapshot log）来传递事务数据；否则就是复杂快照（complex snapshot），数据库将执行本地快照表的完全刷新。图16-4示出了这种通过快照实现数据动态复制的过程。



a) 简单快照，可以使用快照日志



b) 复杂快照，连结的结果被复制

图16-4　简单快照和复杂的快照

还有其他可以用来复制数据的方法，但它们不由数据库来动态管理。例如，可以使用SQL*Plus中的copy(拷贝)命令在本地数据库中创建远程数据表的拷贝（参见第13章，13.6节“应用样例：copy命令”）。然而copy命令需要在数据改变时重复使用，因此，它只适用于那些大静态表的复制。动态数据要求动态复制。

16.4　分布式数据管理

在考虑远程数据库事务管理之前，必须从那里获取数据，并且使其他数据库能对它进行全局访问。下面几节描述需要的管理工作：用于访问数据的位置透明性、数据库链接管理、触发器管理和快照管理。

注意　本章的例子假定正在使用数据库服务名方案的tnsnames.ora文件。

16.4.1　基础结构：实施位置透明性

要恰当设计长期使用的分布式数据库，必须使数据的物理位置对应用程序是透明的。数据库中的表名在所属的模式中是唯一的。因此，在单个数据库中，拥有者与表名的结合将唯一指定一个表。然而，一个远程数据库有可能拥有一个同名的帐户，并且这个帐户也同样拥有一个同名的表。那么如何正确地限定表名呢？

在分布式数据库中，需要添加两个附加的对象识别层。首先，必须标识访问数据库的实例名。其次，实例所在的主机名也必须标识。将对象名的四个部分——主机、实例、拥有者和名字——放在一起，就形成一个全限定对象名（FQON，full qualified object name)。FQON有时候也称为全局对象名(global object name)。若要访问一个远程表，这个表的FQON必须是已知的。图16-5示出了一个样例。

位置透明性的目标是使FQON的前三个部分——主机、实例和模式——对用户透明。也有可能使对象名本身对用户透明(例如,可以指向由两个表连结成的视图)。本章将原封不动地保持FQON的这部分作为一个引用点。

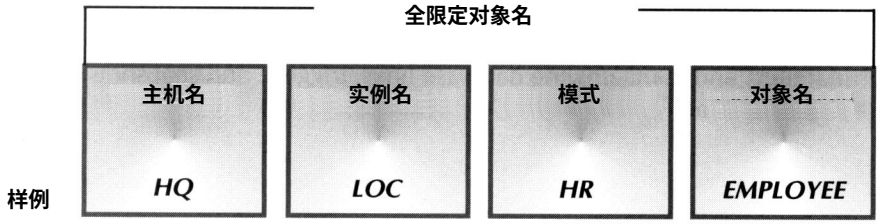


图16-5 全限定对象名

FQON的前三个部分都通过数据库链接限定，因此实现位置透明性的任何工作都应该从这里入手。首先假设一个典型的数据库链接：

```
create public database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';
```

注意 如果init.ora参数GLOBAL_NAMES设置为TRUE，则数据库链接名必须与远程数据库名相同。

通过使用服务名(‘hq’)，使主机和实例名保持透明。它们通过本地主机的tnsnames.ora文件，转变成实际值。下面清单是此文件中有关服务名的部分条目。

```
hq =(DESCRIPTION=
      (ADDRESS=
            (PROTOCOL=TCP)
            (HOST=HQ)
            (PORT=1521))
      (CONNECT DATA=
            (SID=loc))))
```

清单中两行黑体字填充在FQON的两个遗漏部分中：当使用HQ服务名时，主机名是HQ，实例名是LOC。这个tnsnames.ora文件展示了TCP/IP协议的参数；其他协议可能使用不同的关键字，但是它们的用法是相同的。tnsnames.ora条目为服务器名和实例名提供透明性。

如果通过本节前面给出的代码创建HR_LINK数据库链接，则它将为FQON的前个两部分提供透明性。但是，如果数据从HR模式移出或HR帐户的口令发生变化的话，则必须撤消并重新创建数据库链接。如果需要帐户级安全，也同样如此；不过可能要创建并维护多个数据库链接。

若要解决FQON中模式部分的透明性问题，可以修改数据库链接的语法。下面是一个数据库链接的例子：

```
create public database link HR_LINK
connect to current_user
using 'hq';
```

这个数据库链接使用connect to current_user子句。它将使用缺省连接(default connection)。下面是使用这个链接的例子：

```
select * from EMPLOYEE@HR_LINK;
```

当使用HR_LINK时，数据库将以下面方式解决FQON：

1) 搜索本地tnsnames.ora文件，以确定正确的主机名。

2) 搜索本地tnsnames.ora文件，以确定正确的实例名。

3) 搜索数据库链接，以获得connect to说明。若发现connect to current_user子句，它将用当前用户的用户名及口令连接指定的数据库。

4) 搜索查询的from子句，以得到对象名。

缺省连接常用来访问由用户名限定表行的数据表。例如，如果远程数据库具有一个名为HR.EMPLOYEE的表，并且每一个雇员可以查看他们自己的记录。如果使用具有一个特定连接的数据库链接将作为HR帐户(表的拥有者)注册：

```
create public database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';
```

如果使用这种指定连接，就不能限制远程主机中用户的记录视图。然而，如果使用缺省连接且使用用户伪列(User pseudocolumn)在远程主机上创建一个视图，则只从远程主机返回这个用户的数据。下面是这种类型的数据库链接和视图的例子：

```
create public database link HR_LINK
connect to current_user
using 'hq';

create view REMOTE_EMP
as select * from EMPLOYEE@HR_LINK
where Ename=User;
```

用户伪列的值是当前Oracle用户名。如果查询REMOTE_TMP视图，可以使用HR_LINK数据库链接。由于该链接使用一个缺省连接，所以将使用用户名和口令来连接 hq服务名的数据库。因此只能检索EMPLOYEE@HR_LINK中这个表中Ename列与用户名的值相等的那些记录。

无论哪种方式，都可以限制被检索的数据。不同的是，当使用缺省连接时，就可以根据远程数据库中的用户名来限定数据。若使用特定连接，则可以在数据返回本地数据库之后再限定数据。这种缺省连接方法可以减少解决查询所需要的网络通信量并增加数据位置透明性的附加层。

在数据库链接中使用缺省链接会引发一系列不同的维护问题。 tnsnames.ora文件必须在服务器间保证同步，并且在多个数据库中用户名 /口令的组合也必须同步。这些问题在下一节讨论。

使用共享数据库链接

如果把多线程服务器选项用于数据库链接且应用程序应用许多并行数据库链接的连接，则使用共享数据库链接比较有利。共享数据库链接使用共享服务器连接来支持数据库链接的连接。如果有多个数据库链接对一个远程数据库进行访问，可以使用共享数据库链接来减少

所需要的服务器连接数量。

若要创建一个共享数据库链接，可以使用 create database link命令的shared关键字。如下所示，还需要规定远程数据库的模式和口令：

```
create shared database link HR_LINK_SHARED
connect to current_user
authenticated by HR identified by puffinstuff
using 'hq';
```

数据库链接HR_LINK_SHARED使用由connect to current_user子句规定的缺省连接。为了防止非法访问共享链接，共享链接需要使用 authenticated by子句。在这个例子中，用于鉴别的帐户是一个应用程序帐户，但是也可以使用一个空的鉴别模式。鉴别帐户必须拥有CREATE SESSION系统权限。在使用HR_LINK_SHARED链接时，连接活动还应包括对HR链接帐户的鉴别。

如果改变鉴别帐户的口令，就要撤消并重新创建引用这个口令的每一个数据库链接。因此，应创建一个帐户以便对共享数据库链路的连接进行鉴别。该帐户应只有 CREATE SESSION系统权限，并且不应对应用程序表有任何特权。

如果应用程序不常使用数据库链接，应使用传统的数据库链接而不使用 shared子句。如果没有shared子句，每个数据库链路的连接都需要对远程数据库进行单独连接。

16.4.2 数据库链接管理

可以通过DBA_DB_LINKS数据字典视图来检索有关公用数据库链接的信息。有关私有数据库链接的信息则可以通过 USER_DB_LINKS数据字典视图来查看。只要有可能，就要利用应用程序来分离数据库中的用户，以便他们都可以共享相同的公用数据库链接。作为附带好处，这些用户通常也可以共享公用授权及同义词。

DBA_DB_LINKS数据字典视图中的列在下面的表中列出。不能通过 DBA_DB_LINKS数据字典视图查看所使用的链接口令；该口令以未加密的方式存储在 SYS.LINK$表中。

列 名	描 述
OWNER	数据库链接的拥有者
DB_LINK	数据库链接名(例如本章例子中的HR_LINK)
USERNAME	使用指定链接时，在远程数据库中打开会话所使用的帐户名
HOST	用来与远程数据库连接的SQL*Net连接字符串
CREATED	表示数据库链接创建日期的时间标记

注意 一个查询可使用的数据库链接数量由数据库的 init.ora文件中的OPEN_LINKS参数限定，其缺省值是4。

与数据库链接有关的管理任务取决于在数据库中实现位置透明性的程度。也取决于所使用的SQL*Net版本，Net8的用户应参见第13章中关于使用Net 8 Assistant的描述。

在最好情况下，缺省连接与服务名或别名一起使用。在这种情况下，成功维护的唯一要求是：主机间的tnsnames.ora文件必须一致并全局维护用户帐户/口令组合。文件同步可以通过操作系统来实现——例如，使用UNIX rcp(远程拷贝)命令向远程主机拷贝文件。

帐户/口令组合的同步则更困难一些，但可以有几种选项。首先，对用户帐户口令的所有更改都要经过中央管理机构。中央管理机构负责更新网络中所有数据库的帐户口令——这是

个耗时的工作，但值得一做。

第二，通过审查所有 alter user命令来审查所有用户口令的变化。如果一个数据库中的用户口令发生变化，则网络中所有通过缺省连接访问的数据库中的口令也将要变化。如果使用Oracle8引入的口令管理功能，数据库口令变化的同步实现就非常困难。这是因为在 Oracle中可以强迫口令失效并频繁地改变口令。

如果FQON中的任一部分 —— 如用户名 —— 被嵌入数据库链接中，那么对 FQON中该部分的改变都需要撤消数据库链接并重新创建。例如，如果改变 HR用户的口令，则前面定义的具有特定连接的HR_LINK数据库链接将用

```
drop database link HR_LINK;
```

来撤消再重新创建该数据库链接，同时使用新的帐户说明：

```
create public database link HR_LINK
connect to HR identified by NEWPASSWORD
using 'hq';
```

不能在另外的用户帐户中创建数据库链接。如果试图如下这样在 SCOTT帐户中创建数据库链接：

```
create database link SCOTT.HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';
```

则Oracle不会在SCOTT帐户中创建HR_LINK数据库链接。Oracle将在执行create database link命令的帐户中创建一个名为 SCOTT.HR_LINK的数据库链接。若要创建私有数据库链接，则必须登录到将要拥有该链接的数据库帐户。

16.4.3 数据库触发器管理

如果所需的数据复制十分有限，就可以使用数据库触发器在表间复制数据。通常，只有在发送到远程数据库的数据类型是 insert或delete时才使用这种方法。支持 update事务的代码通常比相应的快照要复杂得多。下面几节将对快照的实现情况进行讨论。

当对特定的表执行特定的操作时，会引发数据库触发器。这些触发器可以把每一行事务或整个事务作为一个单元来执行。当进行数据复制时，通常关心的只是每行数据。

在创建触发器之前，必须为要使用的触发器创建一个数据库链接。在这种情况下，数据库链接应当在拥有数据的数据库中创建，并可以对复制表的拥有者进行访问。

```
create public database link TRIGGER_LINK
using ‘remotel’ ;
```

这个数据库链接命名为TRIGGER_LINK，使用一个服务名(remote1)来指定与远程数据库的连接。由于没有规定特定的connect子句，所以使用缺省连接来代替。这个缺省连接将试图注册到remote1数据库中，这个remote1数据库使用的用户名和口令与调用数据库链接的帐户相同。

下面清单中所示的触发器使用了这个数据库链接。在向 EMPLOYEE表插入每一行后都会引发这个触发器。由于这个触发器是在插入行后激发的，所以行数据已经被验证。随后它使用定义的TRIGGER_LINK数据库链接，以同样的结构把同样的行插入到远程数据表中。这个远程数据库表必须已经存在。

```
create trigger COPY_DATA
after insert on EMPLOYEE
for each row
```

```
begin
     insert into EMPLOYEE@TRIGGER_LINK
     values
     (:new.Empno, :new.Ename, :new.Deptno,
     :new.Salary, :new.Birth_Date, :new.Soc_Sec_Num);
end;
/
```

该触发器使用new关键字来引用刚刚插入本地EMPLOYEE表的行数据。

注意 如果使用基于触发器的复制，触发器代码就必须考虑远程站点的潜在错误条件，例如复制关键字值、空间管理问题或数据库故障。

若要列出触发器的有关信息，可以使用DBA_TRIGGERS数据字典视图。下面的查询将列出触发器的“头”信息——触发器的类型、调用触发器的语句和触发器在其中调用的表。这个例子展示了刚刚创建的COPY_DATA触发器的头信息：

```
select Trigger_Type,
       Triggering_Event,
       Table_Name
  from DBA_TRIGGERS
 where Trigger_Name = 'COPY_DATA';
```

下面是这个查询的输出样例：

```
TYPE             TRIGGERING_EVENT        TABLE_NAME
---------------- ----------------------- ------------
AFTER EACH ROW   INSERT                  EMPLOYEE
```

也可以从DBA_TRIGGERS查询触发器文本，如下所示：

```
select Trigger_Body
  from DBA_TRIGGERS
 where Trigger_Name = 'COPY_DATA';
```

下面是这个查询的输出样例：

```
TRIGGER_BODY
------------------------------------------------------
begin
     insert into EMPLOYEE@TRIGGER_LINK
     values
     (:new.Empno, :new.Ename, :new.Deptno,
     :new.Salary, :new.Birth_Date, :new.Soc_Sec_Num);
end;
```

理论上讲，有可能创建一个触发器来复制本地数据库中所有可能的数据复制操作事务，但这样做很快就会变得非常难以管理。对于一个复杂的环境，应该考虑使用快照或手工数据拷贝。然而，对于前面提到的有限环境，触发器倒是一种简捷方案。

注意 如果使用触发器进行数据库复制，则主数据库中事务处理成功与否依赖于远程事务处理是否成功。

16.4.4 快照管理

可以使用快照在分布式数据库之间动态复制数据。主表可以更新，但快照则或是只读，或是可更新。其中只读快照是最常用的快照类型。有两种可用的快照类型：复杂快照(complex snapshot)和简单快照(simple snapshot)。

在一个简单快照中，每一行都基于一个远程数据表中的一个行。而复杂快照的行则基于一个远程数据表的多行——例如通过一个group by操作，或是基于多个表连结的结果。

由于快照将在本地数据库中创建一些对象，因此，创建快照的用户必须要有 CREATE TABLE权限和UNLIMITED TABLESPACE权限或存储快照对象的表空间的定额。快照在本地数据库中创建并从远程主数据库获取数据。

在创建一个快照之前，首先要在本地数据库中创建一个到源数据库的链接。下面的例子创建一个名为HR_LINK的私有数据库链接(这个例子作为全章的例子)。

```
create database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';
```

在下面的程序清单中，展示了在本地服务器上创建快照所使用的句法。在这个例子中，给予快照一个名称(EMP_DEPT_COUNT)并且其存储参数被规定。表空间和存储区参数应用于存储快照数据的本地基表。

除了刷新间隔外，还给出了它的基本查询。在这种情况下，快照被通知立即检索主数据，然后七天(SysDate+7)后再次执行快照操作。

```
create snapshot EMP_DEPT_COUNT
pctfree 5
tablespace SNAP
storage (initial 100K next 100K pctincrease 0)
refresh complete
      start with SysDate
      next SysDate+7
as select Deptno, COUNT(*) Dept_Count
     from HR.EMPLOYEE@HR_LINK
    group by Deptno;
```

注意 由于Oracle在支持快照的数据库对象名中使用快照名，所以快照的名字长度不应超过19个字符。

注意 快照查询不能引用用户SYS所拥有的表或视图。

有关create snapshot命令的全部语法选项，请参见附录A。

由于这个快照中的记录并不与主表中的记录一一对应 (因为该查询含有一个 group by子句)，所以这是一个复杂快照。由于该快照是个复杂快照，因此每次刷新时都需要完全重建。

注意 创建一个快照时，必须引用远程数据库中的整个对象名——在上面的例子中，对象名是HR.EMPLOYEE。

当创建这个快照时，就会在本地数据库中创建一个数据表。 Oracle将创建一个称作“SNAP$_*snapshotname*”的数据表——快照的本地基表——用来存储快照查询返回的记录。尽管此表可以被索引，但它不能以任何方式改变。还将创建这个表的一个只读视图 (以快照命名)。第二个称作“MVIEW$_*snapshotname*的视图，将作为远程主表的视图被创建。此视图将用于刷新过程。

可以使用Oracle Enterprise Manger Schema Manager(Oracle企业管理器模式管理器)来创建和管理数据库链接与快照，并且监控快照日志。图 16-6示出了用填充的信息来创建EMP_DEPT_COUNT快照的create snapshot屏幕。

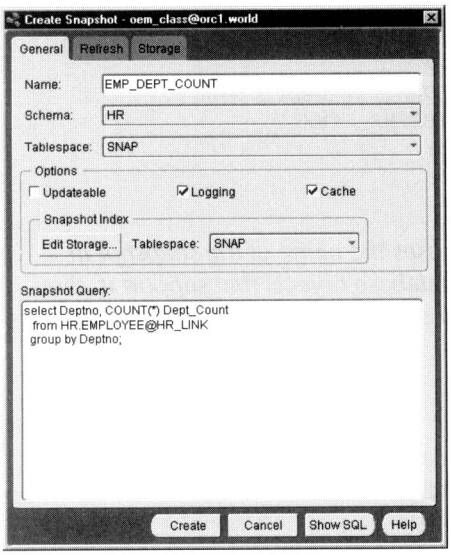


图16-6 用OEM创建快照

若要撤消一个快照，可使用drop snapshot命令。下面列出了撤消快照的例子：

```
drop snapshot EMP_DEPT_COUNT;
```

可以通过alter sanpshot命令来改变快照的存储参数：

```
alter snapshot EMP_DEPT_COUNT pctfree 5;
```

要改善快照的性能，可以给本地基表添加一个索引。这可以通过 create index命令在基本的SNAP$_*snapshotname*表上创建相应的索引来实现。

注意 不要在本地基表上为该快照创建约束条件。

可以查询DB_SANPSHOTS数据字典视图来查看有关快照的数据。下面的清单是对这个数据字典视图进行查询的例子：

```
select
 Name,            /*Name of the view used for the snapshot*/
 Last_Refresh,    /*Timestamp for the last refresh*/
 Type,            /*Type of refresh used for automatic refreshes*/
 Query            /*Query used to create the snapshot*/
from DBA_SNAPSHOTS;
```

这个查询将返回最常用的快照信息。其他信息——如主表名、所使用的数据库链接名——也可以通过DBA_SHAPSHOTS视图被检索。

1. 介质失败的处理

一旦创建一个快照，它的数据就会与主数据链接起来 (逻辑链接而不是物理链接)。如果主数据所在的服务器出现问题，可能需要重新创建或完全刷新此快照。

例如，本章前面几节提到的EMP_DEPT_COUNT快照便是基于远程EMPLOYEE表中的数据。如果EMPLOYEE表所使用的服务器出现介质失败，则可能要执行此服务器的数据库恢复操作。除非可以恢复所有丢失数据，否则主表 EMPLOOYEE将不可能含有快照创建时的全部记录。结果是，基(EMPLOYEE)表中的数据与其快照将不能保持同步。

如果快照是一个简单快照，可以使用 with rowid子句来创建。这个 with rowid子句通知

Oracle使用主表记录中的RowID作为主表中的行与快照(快速刷新)中的行相关联的方式。如果要执行对主表的恢复操作，则应当完全刷新快照。如果需要在快照的数据库上执行恢复，就要在完成恢复操作之后执行快照的完全刷新。

2. 实施快照引用完整性

如果两个相关表对远程数据库都有一个简单快照，它们的快照之间可能没有实施引用完整性。如果表在不同的时间进行刷新，或者在刷新间主表出现事务，则这些表的快照就可能不反映主表中的引用完整性。

例如，如果EMPLOYEE与DEPT通过主键/外键关系而相互关联，则这些表的简单快照可能会违反这种关系。这些违反包括与主键不匹配的外键。在这个例子中，可能指的是一些雇员在EMPLOYEE快照中具有的DEPTNO值在DEPT快照中却不存在。

对这个问题可能有几种解决方案。首先，要在不使用主表时进行刷新。第二，在锁定主表后立即执行手工刷新(参见下节有关这方面的信息)。第三，连结快照涉及的所有表，创建基于主表的复杂快照。

使用刷新组是解决快照引用完整性问题的第四个方案。可以把相关的快照汇集到刷新组(refresh group)中。刷新组的目的是协调成员间的刷新调度。其主表与其他快照主表有关系的快照应考虑使用刷新组。协调调度快照的刷新，也能够维护主表在快照中的引用完整性。就主表的引用完整性而论，如果不使用刷新组，快照中的数据就可能不一致。

可以通过DBMS_REFRESH软件包来实现刷新组的所有操作。如下面的例子所示，DBMS_REFRESH软件包中的过程包括MAKE、ADD、SUBTRACT、CHANGE、DESTROY和REFRESH。有关现有刷新组的信息可以从USER_REFRESH和USER_REFRESH_CHILDREN数据字典视图中查询。

注意 属于刷新组的快照不必属于相同的模式，但它们都应当存储在同一个数据库中。

通过执行DBMS_REFRESH软件包中的MAKE过程来创建一个刷新组，其结构如下：

```
DBMS_REFRESH.MAKE
(name IN VARCHAR2
{ list IN VARCHAR2, |
  tab IN DBMS_UTILITY.UNCL_ARRAY,}
 next_date IN DATE,
 interval IN VARCHAR2,
 implicit_destroy IN BOOLEAN := FALSE,
 lax IN BOOLEAN := FALSE,
 job IN BINARY INTEGER := 0,
rollback_seg IN VARCHAR2 := NULL,
push_deferred_rpc IN BOOLEAN := TRUE,
refresh_after_errors IN BOOLEAN := FALSE,
purge_option IN BINARY_INTEGER := NULL,
parallelism IN BINARY_INTEGER := NULL,
heap_size IN BINARY_INTEGER := NULL);
```

除了前4个参数外，此过程的其他参数都有通常可接受的缺省值。可以使用下面的命令来为名为LOCAL_EMP和LOCAL_DEPT的快照创建一个刷新组：

```
execute DBMS_REFRESH.MAKE
(name => 'emp_group',
 list => 'local_emp, local_dept',
 next_date => SysDate,
 interval => 'SysDate+7');
```

注意 快照的list参数，也就是清单中的第二个参数，在开头和结尾具有一对单引号，中间则没有。在这个例子中，两个快照 LOCAL_EMP和LOCAL_DEPT均通过一个参数传递给这一过程。

前面的命令将创建一个名为 EMP_GROUP的刷新组，并含有两个快照作为其成员。刷新组名用一对单引号括起来。

如果刷新组要包含一个已属于其他刷新组的快照 (例如，把一个旧刷新组中的快照转移到新创建的刷新组中)，则必须将lax参数设置为TRUE。一个快照一次只能属于一个刷新组。

若要向现有的刷新组中增加快照，可以使用 DBMS_REFRESH软件包中的ADD过程，其结构如下：

```
DBMS_REFRESH.ADD
(name IN VARCHAR2,
{ list IN VARCHAR2, |
  tab IN DBMS_UTILITY.UNCL_ARRAY, }
 lax IN BOOLEAN := FALSE);
```

和MAKE过程的情况一样，除非一个快照正在两个刷新组之间移动，否则就不必规定ADD过程的lax参数。当执行此过程且 lax参数为TRUE时，快照就移到新的刷新组中，并自动从旧的刷新组中删除。

可以用DBMS_REFRESH软件包中的SUBTRACT过程移去一个现有刷新组中的快照，例如:

```
DBMS_REFRESH.SUBTRACT
(name IN VARCHAR2,
{ list IN VARCHAR2, |
  tab IN DBMS_UTILITY.UNCL_ARRAY, }
 lax IN BOOLEAN := FALSE);
```

与MAKE和ADD过程的情况一样，一个快照或快照列表 (用逗号隔开)可以用作对SUBTRACT过程的输入。刷新组的刷新调度可以通过 DBMS_REFRESH软件包中的CHANGE过程来改变：

```
DBMS_REFRESH.CHANGE
(name IN VARCHAR2,
 next_date IN DATE := NULL,
 interval IN VARCHAR2 := NULL,
 implicit_destroy IN BOOLEAN := NULL,
 rollback_seg IN VARCHAR2 := NULL,
 push_deferred_rpc IN BOOLEAN := NULL,
 refresh_after_errors IN BOOLEAN := NULL,
 purge_option IN BINARY_INTEGER := NULL,
 parallelism IN BINARY_INTEGER := NULL,
 heap_size IN BINARY_INTEGER := NULL);
```

next_date参数与create snapshot中的start with子句类似。 interval(间隔)参数与create snashot命令中的next子句类似。例如，若要改变EMP_GROUP的刷新调度以便每3天复制一次，可以执行下述命令(这个命令为next_date参数规定一个NULL值，使这个值保持不变)：

```
execute DBMS_REFRESH.CHANGE
(name => 'emp_group',
 next_date => null,
 interval => 'SysDate+3');
```

在这个命令执行后，EMP_GROUP刷新组的刷新周期每3天变化一次。

如下面例子所示，若要删除一个刷新组，可以使用 DBMS_REFRESH软件包中的DESTROY过程来实现。它的唯一参数是刷新组的名字。

```
execute DBMS_REFRESH.DESTROY(name => 'emp_group');
```

也可以隐式删除刷新组。如果在用MAKE过程创建这个组时把implicit_destroy参数设置为TRUE，则在刷新组的最后一个成员从该组中删除时就删除这个组。

注意 对快照组的刷新操作可能比类似的快照刷新操作要长。快照组刷新还可能需要较大的回滚段空间，以便在刷新时维持数据的一致性。

3. 快照的尺寸调整及存储

快照的正确存储参数由所选择的数据确定。在数据复制后，就可以知道源数据并能够进行大小调整。可以用第5章介绍的空间计算方法来调整快照本地基表的大小。

如果使用复杂快照，所需的存储区就会不同。例如，对于前节提到的EMP_DEPT_COUNT快照，应使用比它的主表少的空间，这是因为它对表执行了 group by操作并且只返回两列。对于其他复杂快照 —— 如那些可能要涉及到表连结或从多个表中返回列的快照 —— 快照所需的空间则可能超过相关的任何主表。因此，往往是计算本地快照的空间，而不依赖主表的存储参数。

由于快照数据和对象的特点，可能要创建一个用于支持它们的表空间。而重点则应当放在提供足够的连续空间上，这样才不会发生因为空间不足而导致的快照刷新失败。

4. 快照的自动与手工刷新

前面定义的EMP_DEPT_COUNT快照具有下例指定的刷新间隔：

```
refresh complete
      start with SysDate
      next SysDate+7
```

其中的refresh complete子句指明快照每次被刷新时它都应被彻底重新创建。下表列出了可用的refresh选项。

刷新选项	描 述
COMPLETE	每次刷新快照时，都使用快照的查询和主表完全重新生成快照表
FAST	如果使用简单快照，就使用一个快照日志来只将变化发送到快照表
FORCE	缺省值。如果可能，就执行一个FAST刷新操作；否则执行一个COMPLETE刷新操作

start with子句通知数据库何时应刷新快照。在这个例子中，它被指定为 SysDate，因此数据库将在创建快照时复制数据。

next子句设置刷新间隔。无论刷新是通过数据库自动进行还是由数据库管理人员手工完成，这个时间周期都是从上一次刷新时开始计算。在这个例子中，刷新将在上一次快照完成之后七天进行。

对于自动快照刷新，则必须通知Oracle创建SNP后台进程，由这些后台进程执行快照刷新。SNP后台进程的数量由init.ora参数JOB_QUEUE_PROCESSES来确定。如果不为这个参数设置一个值，则缺省为0且不会出现自动刷新。除非有许多快照需要同时刷新，否则一般只需要一个这样的后台进程(SNP0)。不过，可以设置多达36个SNP进程。

SNP*n*进程的“唤醒调用”之间的时间间隔 (以秒为单位)，由init.ora参数文件中的

JOB_QUEUE_INTERVAL参数进行设定。缺省时间间隔为60秒。

注意 JOB_QUEUE_PROCESSES和JOB_QUEUE_INTERVAL参数创建的后台进程用于作业队列的管理及快照的刷新。作业队列的管理将在本章后面描述。

可以使用DBMS_SNAPSHOT软件包中的REFRESH过程来手工刷新一个快照。下面列出了这个命令用途的一个例子。在这个例子中，用户使用 execute命令来执行这个过程。传送给此过程的参数由这个例子描述。

```
execute DBMS_SNAPSHOT.REFRESH('emp_dept_count','?');
```

如上所示，DBMS_SNAPSHOT软件包的REFRESH过程使用两个参数。头一个是快照名，并以快照拥有者的名称作为前缀(如果由其他用户执行这个命令)。第二个参数是手工刷新选项。下面的表中列出了手工刷新选项参数可采用的值。

手工刷新选项	描 述
F	快速刷新
f	快速刷新
C	完全刷新
c	完全刷新
?	应使用快照的缺省刷新选项

可以使用DBMS_SNAPSHOT软件包中的另一个过程来刷新被安排为自动刷新的所有快照。此过程名为REFRESH_ALL，它将分别刷新每一个快照。它不含任何参数。下面示出了一个执行的例子：

```
execute DBMS_SNAPSHOT.REFRESH_ALL;
```

由于快照将通过REFRESH_ALL来连续刷新，所以快照不会同时被刷新。因此，如果在这个执行过程中，出现数据库或服务器失败，则可能导致本地快照之间的不同步。如果这样，则只需在数据库恢复之后重新执行此过程。

可以通过DBMS_REFRESH软件包的REFRESH过程来手工刷新一个刷新组。REFRESH过程以一个刷新组名作为其唯一参数。下面所示的命令将刷新名为 EMP_GROUP的刷新组：

```
execute DBMS_REFRESH.REFRESH('emp_group');
```

5. 快照日志管理

快照日志是一个表，保存对快照的主表的修改记录。它与主表存储在同一个数据库中，并且只能被简单快照使用。快照日志中的数据在快照快速刷新时使用的。若要使用这种方式，需要在创建快照之前创建快照日志。

要创建一个快照日志，必须能够在表上创建一个 AFTER ROW触发器。这意味着必须具有CREATE TRIGGER和CREATE TABLE权限。不能为快照日志规定名字。

注意 由于Oracle在支持快照日志的数据库对象名中使用主表名，因此主表名应少于19个字符。

由于快照日志是一个表，它可利用表存储子句的全部参数，下面显示了在 EMPLOYEE表上创建一个快照日志的例子。该日志以指定的存储参数存储在 DATA_2表空间中。

```
create snapshot log on EMPLOYEE
tablespace DATA_2
storage(initial 100K next 50K pctincrease 0)
pctfree 5 pctused 90;
```

如这个例子所示，该表的pctfree值可以设得很低，而pctused值应设得很高。快照日志的大小取决于刷新时要处理的修改数量。快照刷新越频繁，快照日志所需的空间就越小。

就像快照创建底层表一样，快照日志创建一组数据库结构。在主表的数据库中，快照日志创建一个名为“MLOG$_*tablename*”的表来存储主表中行的RowID和时间标记。这个表用来识别上次刷新后修改过的行。一个内部触发器将填充MLOG$_*tablename*表。不能修改快照日志表。如果快照是基于主键而不是RowID，那么快照日志将包含主键的值而不是RowID值。

可通过alter snapshot log命令修改快照日志的存储参数。当使用该命令时，要指定主表名而不是其快照日志表名。下面是修改EMPLOYEE表的快照日志的例子：

```
alter snapshot log EMPLOYEE
pctfree 10;
```

可以通过DBA_SNAPSHOT_LOGS数据字典视图来查询快照日志的信息。这个视图列出了快照日志的拥有者、主表、快照日志表和使用的触发器。由于快照日志表是数据库中的一个段，且名字已知(MLOG$_*tablename*表)，因此其空间使用可以通过第6章提供的盘区监控脚本文件来跟踪。

要撤消快照日志，可使用drop snapshot log命令。例如：

```
drop snapshot log on EMPLOYEE;
```

这个命令将从数据库中撤消快照日志及其相关对象。

16.4.5　选择刷新类型

哪种刷新类型最适合于数据库？正确答案会随数据库中快照的不同而改变。其决定因素为：

- 网络流量　完全刷新要在网络上传输大量数据。
- 事务大小　完全刷新会生成一个很大的事务(基于快照基本查询的insert as select命令)。快速刷新一般生成较小的事务。
- 数据的变更率　如果超过25%的行被修改，那么完全刷新通常比快速刷新执行得更好。
- 快照本地基表上的索引数　一个完全刷新会截断本地库登记表并对其执行一个insert as select命令。在完全刷新时所有本地基表的索引都保留下来。如果本地基表上有许多索引，就会影响刷新性能。
- 对快照日志的存储需求　为了使用快速刷新，必须创建和维护一个快照日志。这个快照日志将逐渐增大并且不会自动释放自由空间。

对不常变化的数据，快速刷新是最佳的使用方法。由于多个快照可以使用同一个快照日志，所以快照日志数据的空间需求可以由许多快照共享。如果使用快速刷新，请参见本章后面16.4.7节“清除快照日志”中对快照日志空间管理的描述。

16.4.6　快照的脱机实例化

当创建一个快照时，Oracle发出一个create table as select命令来创建和提供快照。如果有大量的数据要复制，这可能就是一个不可取的解决方法，因为它会产生很大的事务和很大的网络流量。如果一个回滚段装不下这个事务(见第7章)，那么快照创建就会失败。

为解决这个问题，可以用Import实用程序把数据装入快照的数据库中。为使用这种方法(叫作快照实例化，offline instantiation)，首先要在主数据库中创建合适的对象，然后将其导

出。这种方式要求主数据库的回滚段大得足以支持快照的创建。

首先在主站点上创建一个帐户，它具有主表上的权限、创建数据库链接的能力和创建快照的能力。在这个帐户中创建一个数据库链接，这个数据库链接所具有的名字与访问主数据库时在远程数据库中所使用的名字相同。然后用刚创建的数据库链接在这个新建的主数据库帐户内创建一个快照。

例如，下面的命令是在主数据库的一个帐户中创建HR_LINK数据库链接和EMP_DEPT_COUNT快照。

```
create database link HR_LINK
connect to HR identified by PUFFINSTUFF
using 'hq';

create snapshot EMP_DEPT_COUNT
refresh complete
      start with SYSDATE
      next SYSDATE+7
as select Deptno, COUNT(*) Dept_Count
     from HR.EMPLOYEE@HR_LINK
    group by Deptno;
```

这些命令完成后，帐户就具有HR_LINK数据库链接、EMP_DEPT_COUNT快照(以及其相关对象)和快照的数据。

接着，导出拥有主数据库中快照的用户。这时就可以把导出转储文件传送到远程服务器，并把该文件的数据导入到远程数据库中。在导入过程中，将创建数据库链接和快照，并提供快照的本地基表。导入期间，创建多大的事务以及需要多大的回滚段可通过 Import实用程序的COMMIT和BUFFER参数来控制。有关Import和Export的使用情况，请参见第10章。

16.4.7 清除快照日志

快照日志包含一些短暂存留的数据，记录被插入到快照日志中，在刷新时使用，然后又删掉。因此，可通过创建快照时为pctused设置一个高值来促进快照日志块的重复使用。

如果多个快照使用同一个主表，那么它们共享同一个快照日志。如果一个快照很长时间不刷新，快照日志也决不会删除其记录。这就引起快照空间需求的增长。

若要减少快照日志条目使用的空间，可使用 DBMS_SNAPSHOT软件包的PURGE_LOG过程。PURGE_LOG有三个参数：一个主表名、一个num变量和一个DELETE标志。num变量指定其行要从快照日志中删去的最近最少刷新的快照数量。例如，如果有三个快照使用快照日志并且其中一个已经久未刷新，那么就使num值为1。

下面的程序示出一个PURGE_LOG过程的例子。在这个例子中，EMPLOYEE表的快照日志将被清除掉最近最少使用的快照所需要的条目。

```
execute DBMS_SNAPSHOT.PURGE_LOG
(master => 'EMPLOYEE',
    num => 1,
   flag => 'DELETE');
```

只要保证管理快照日志时不把行写入主表，就可以像管理其他表的行一样管理快照日志的行。例如，可以导出快照日志的数据、截断日志的表，然后把数据块导入回快照日志以减少其空间需求。

为进一步支持快照维护，Oracle为truncate命令提供了两个快照专用选项。如果想在截断

主表时不丢失快照日志，可输入命令：

```
truncate table EMPLOYEE preserve snapshot log;
```

如果EMPLOYEE表的快照基于主键值，那么该快照日志值在 EMPLOYEE表的导出/导入后仍然合法。但是若 EMPLOYEE表的快照基于Row ID值，那么日志快照在基表的导出 /导入后已失效(由于在导入期间会赋予不同的 RowID)。这种情况下，在截断基表时应截断快照日志。

```
truncate table EMPLOYEE purge snapshot log;
```

16.5 分布式事务管理

一个逻辑工作单元可以包括针对多个数据库的事务。前面图 16-3 的例子说明了这一点：分离的数据库中的两个表更新后就提出一个提交。 Oracle将通过确保全部事务作为一个组提交或回滚来透明地维护两个数据库间的完整性。这可以通过 Oracle的2PC(两阶段提交)机制自动完成。

2PC的第一阶段是准备阶段。在这个阶段，一个事务中的每个节点都准备提交或回滚所需的数据。准备好后，一个节点被称为 in doubt(不确定)。节点将它们的状态通知事务的初始节点(即global coordiantor，全局协调程序)。

一旦所有节点都准备好，事务就进入提交阶段，所有节点根据指令来提交其逻辑事务部分。数据库在同一逻辑时间都提交数据，保证了分布数据的完整性。

解决悬而未决的事务

独立数据库的事务会因为数据库服务器的问题而出现故障，譬如一个介质失败。分布式数据库操作增加了可能出现的故障数量。例如，对于一个远程数据库的事务，就要求用于访问这个数据库和远程主机的网络是可用的。

当一个分布式事务悬而未决时，该事务的条目将出现在 DBA_2PC_PENDING数据字典视图中。当这个事务完成时，就删除其 DBA_2PC_PENDING记录。如果该事务悬而未决，但又不能完成，那么其记录就留在DBA_2PC_PENDING中。

RECO(Recoverer)后台进程定期检查DBA_2PC_PENDING视图，检查未完成的分布式事务。利用检查的信息，一个节点上的RECO进程会自动尝试恢复不确定事务的本地部分。然后尝试建立与那个事务涉及的其他数据库的连接，解决操作事务的分布式部分。这时各数据库中的DBA_2PC_PENDING视图中相关的行被删去。

> 注意 除非启动前把实例的init.ora文件中的DISTRIBUTED_TRANSACTIONS参数设置为非零值，否则RECO后台进程就不会启动。这个参数应设置成并发分布式事务的预期最大数。

分布式事务的恢复由RECO进程自动完成。可以人工恢复一个分布式事务的本地部分，但通常这会在分布式数据库之间造成数据的不一致。由于分布式事务用于实施本地和远程数据的关系，所以数据的不一致性违背了分布式操作的目的。如果执行一个本地恢复，远程数据就会不同步。

要使需要的分布式恢复数量达到最小化，可以影响分布式事务的处理方式。可通过使用

提交点强度(commit point strength)通知数据库如何影响事务进程。

提交点强度

各个分布式事务集都引用多个主机和数据库。当然，通常会选出一个主机和数据库作为最可靠的主机和数据库，或者拥有最关键性的数据。这个数据库称作提交点站点 (commit point site)，如果数据在这里提交，则向所有数据库提交。如果提交点站点的事务失败，那么其他节点的事务就要回滚。这个站点也保存有关分布式事务状态的信息。

Oracle根据各数据库的提交点强度 (commit point strength)来选择提交点站点。如下所示，这种选择通过init.ora文件来设置：

```
COMMIT_POINT_STRENGTH=100
```

为COMMIT_POINT_STRENGTH参数设置的值是一个相对值，而不是绝对值。在上面的例子中设为100。如果另一个数据库的提交点强度为 200，那么该数据库将是涉及这两个数据库的分布式事务的提交点站点。 COMMIT_POINT_STRENGTH值不能超过255。

由于值是相对的，所以要建立一个站点特定的值。把最可靠数据库的提交点设置为 200，其他服务器和数据库相对于此逐步降级。例如，若一个数据库相对于最可靠数据库的可靠度为80%，那么就指定其提交点强度为160(200的80%)。在确定点(这里为200)上固定一个数据库，让其余数据库均匀地分等级。这种方法使得用于各个不同事务的提交点站点都适当。

16.6 数据库域和群

本章中的所有例子都使用一种标准方法来决定一个对象的 FQON —— 在查询中使用对象名并通过一个数据库链接来解决对象名的其他部分。这种方式要跨所有平台和网络选项工作。不过，使用域名服务(DNS)来命名主机的网络可利用Oracle中附加的网络特性。

域名服务允许分层组织一个网络中的主机。这种结构中的每个节点称为一个域 (domain)，并且各个域按功能来进行标记。这些功能可以包括表示公司的“ COM ”和表示学校的“ EDU ”。每个域可以有许多子域。因此，每个主机在网络中只有一个唯一的名字，该名字包含关于如何适应网络层次的信息。网络中的主机名最多有可达四个部分，最左边部分是主机名，其余部分显示该主机所属的域。

例如，一个主机可命名为 HQ.MYCORP.COM。在这个例子中，主机名叫 HQ，它被标识为COM域中MYCORP子域的一部分。

域结构的意义在于两个方面：第一，主机名是 FQON的一部分。第二， Oracle允许在数据库链接名中规定主机名的DNS版本，这样就简化了分布式数据库链接的管理。

若要在数据库链接中使用 DNS名字，首先需要给数据库的 init.ora文件添加两个参数。第一个参数是DB_NAME，应设置为实例名。第二个参数是 DB_DOMAIN，设置为数据库主机的DNS名。DB_DOMAIN规定主机所在的网络域。如果 LOC数据库是建在HQ.MYCORP.COM服务器上，那么其init.ora条目将如下表示：

```
DB_NAME = loc
DB_DOMAIN = hq.mycorp.com
```

若要使用数据库域名，必须在 init.ora文件中将GLOBAL_NAMES参数设置为TRUE，如下所示：

```
GLOBAL_NAMES = true
```

一旦设置好这些参数，必须关闭数据库并使用该 init.ora文件重新启动数据库以使这些设置生效。

注意 如果把GLOBAL_NAMES设置为TRUE，那么所有数据库链接名都必须遵循本节描述的规则。

当使用这种方式创建全局数据库名时，创建的数据库链接名与其指定的数据库相同。因此，一个指向LOC数据库的数据库链接将被命名为 LOC.HQ.MYCORP.COM。如下所示：

```
CREATE PUBLIC DATABASE LINK loc.hq.mycorp.com
USING 'service name';
```

在这种配置中，所创建的数据库链接仍然可能不包含所指定的数据库的全局数据库名。在这些情况中， Oracle要在数据库链接名上附加本地数据库的 DB_DOMAIN值。例如，如果数据库在“ HQ.MYCORP.COM ”域内且数据库链接叫做 LOC，那么使用时该数据库链接名被自动地扩展为LOC.HQ.MYCORP.COM。

使用全局数据库名在数据库名、数据库域和数据库链接名之间建立起一个链接，从而更容易识别和管理数据库链接。例如，可以在各数据库中创建一个公用数据库链接 (如前面例子中所示，没有连接字符串)分别指向其他每一个数据库。数据库的用户不再需要假设使用某个数据库链接才正确；如果知道全局数据库名，也就知道了数据库链接名。如果把一个表从一个数据库移到另一个数据库，或把一个数据库从一个主机移到另一个主机，那么就容易确定哪个旧的数据库链接必须撤消并重建。使用全局数据库名是从独立数据库到真正的数据库网络转移的一部分。

数据库域与群经常被混淆。群 (community)被SQL*Net V2用来识别一个通过相同通信协议进行通信的服务器组。例如，一个群可以被命名为 TCP.HQ.MYCORP.COM。这个名可以把群标识为在“ HQ.MYCORP.COM ”网络域中使用TCP/IP协议进行通信的服务器群。因此，网络域用来帮助实施 SQL*Net群名的唯一性。群名的 TCP部分不是指一个主机或数据库名，而是指主机共享的通信协议。

由于通常同一域中的主机共享同一个通信协议，所以把它作为 SQL*Net群名中的网络域部分是非常有用的。由于群一般不跨越网络层的这一级，所以网络域名的 COM部分通常不留在群名中。因此，本例中的 TCP/IP群将被命名为TCP.HQ.MYCORP，使用一个三部分的群名也有助于减少群名和网络域名间的混淆。

16.7 分布式数据库监控

像第6章描述的Command Center数据库一样，大多数的数据库级监控系统在分析数据库性能时不需要考虑其环境。不过，必须考虑数据库的下面这些关键性能：

- 主机的性能。
- 磁盘和控制器间的I/O分布。
- 内存的利用。

对于分布式数据库，还必须考虑以下几点：

- 网络及其硬件的容量。
- 网络段上的负载。
- 主机间不同物理访问路径的使用。

这些都不是数据库能测量的，分布式数据库监控的焦点从中心数据库移到中心网络。数

据库成了监控环境的一部分，而不是检查的唯一部分。

还要监控那些对成功起关键作用的数据库方面——例如段的扩展和表空间的自由空间。然而，除了支持分布式数据库的部分网络性能外，分布式数据库的性能是不能测量的。因此所有与性能相关的测试(如应力测试)，必须与网络管理人员合作。网络管理人员也可以检验试图在网络上减少数据库负载的效率。

各个主机的性能通常可通过网络监控软件包来监控。这种监控采用自顶而下的方式——网络到主机到数据库的顺序。第6章描述的监控系统用作网络和主机监控器的一个延伸。

16.8 分布式数据库调整

当调整一个独立数据库时，其目的是要减少查找数据的时间。如第8章所述，可以使用一系列的数据库结构和选项来增加在内存或数据库的第一查找位置找到数据的可能性。

处理分布式数据库时，存在一些附加条件。因为现在不仅要找到数据，而且还要通过网络传输数据。执行一个查询现在等于是执行两步操作。因此必须考虑数据在网络上的传输方法，目的是缓解网络拥挤现象。

减轻网络通信流量的一个简单方法是从一节点向另一个节点复制数据。这可以人工进行(通过SQL*Plus copy命令)或由数据库自动完成(通过快照)。通常在本地主机的一个慢周期期间，通过把数据一次输入到网络上的方法来复制数据，可以改进对远程数据库查询的性能。本地查询可以使用数据的本地拷贝，消除网络流量需求。

复制数据带来两个问题：第一，本地数据会与远程数据不同步。这是获取数据方面的历史问题；它限制这个选项对静止数据表有用。即使是使用有快照日志的简单快照，也只是在安排好的时刻进行刷新。

复制数据方法的第二个问题是表的拷贝不能通过update返回到主表。也就是说，如果用一个只读快照来做一个远程表的本地拷贝，那么该快照就不能更新。用SQL*Plus copy命令创建的表也同样如此。它们不是主表，应看作是只读表。

因此，对这些表的任何更新都要对主表执行。如果表要频繁地更新，除非使用Oracle的高级复制选项(Advanced Replication Option)，否则复制数据也不会改进性能。这个选项支持数据的多站点拥有权，用户可以在指定为数据拥有者的任一数据库中进行修改。Oracle的高级复制选项的管理非常复杂，需要创建支持数据双向复制的数据库环境(数据库链接等)。要了解高级复制先项的细节，参见Oracle manuals。

使用快照的类型随应用程序的性质而定。简单快照不涉及任何查询操作——它们是远程表中行的简单拷贝。复杂快照执行group by、connect by或远程表的连结等操作。只有知道本地数据库使用数据的方法，才知道使用何种操作。

如果使用简单快照，可以使用快照日志。当刷新表的快照时，只是快照日志上的事务通过网络传送。若远程表频繁修改，只有不到25%的表行需要频繁刷新，那么使用简单快照会改进快照刷新处理的性能。

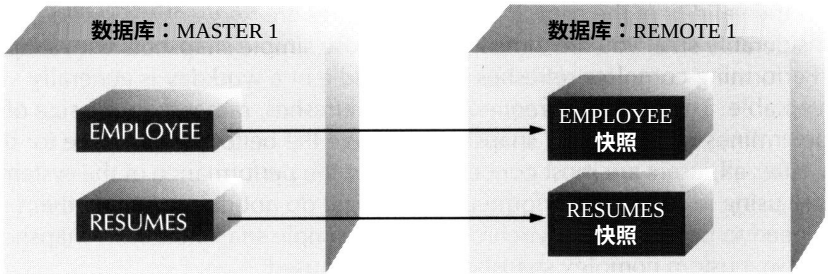
当刷新一个复杂快照时，必须完全重建快照表。咋一看来，这好像是一个加在系统上的巨大负担——为什么不使用简单快照来代替呢？不过，复杂快照有许多优点：

- 选择复制的表一般不常修改，因此通常把刷新安排在本地数据库使用最少的时间内进行，以减少全部刷新的影响。

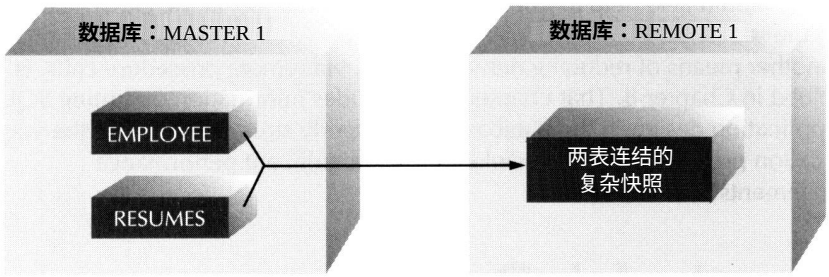
• 复杂快照可以复制比简单快照少的数据。

第二点看起来含义有点模糊。毕竟复杂快照比简单快照涉及更多的处理，因此难道不应该要求网络做更多工作吗？实际上，它可能要求网络做的工作更少，这是因为大量的工作都由数据库完成了。

现在设想通过快照复制两个表的情况。本地数据库的用户总是通过一个连结一起查询两个表。如图16-7所示，有两个选项。可以使用两个简单快照 (图16-7a)或通过一个复杂快照来执行连结(图16-7b)。这两者之间在性能上有什么不同呢？



a) 多个简单快照，可以使用快照日志及快速刷新



b) 一个复杂快照，已经执行了连结

图16-7　连结的数据复制选项

如果表已正确连结，那么即使复杂快照从两个表返回所有列，它在网络上传输的数据也不应比第一次创建的两个简单快照所发送的数据多。事实上，在创建期间很可能传输较少的数据。当根据性能在这两种方案之间进行选择时，要考虑两个因素：

• 刷新性能。

• 对快照的查询性能。

第二个因素通常比较重要，毕竟复制数据是为了改进查询性能。如果用户仅是通过一个特定连结访问表，那么复杂快照已为它们完成了这个连结，而两个简单快照完成这个连结则需要更长的时间。在用户要使用的访问路径未完全定义好之前，不能确定哪个选项更好。如果表有时需要分别查询或是通过不同的连接路径查询，那么就需要使用简单快照或多个复杂快照。

刷新的性能不会影响用户。影响它们的是数据的有效性。如果远程表频繁修改并且相当大，那么就要强制使用带快照日志的简单快照。在一个工作时间内执行完全刷新一般是不可能的。因此是刷新的频率而不是其大小决定哪个快照类型对用户有更好的性能。毕竟用户在使用时最关心的是系统的性能，在夜晚执行刷新并不会直接影响他们。如果表需要频繁同步，

就使用带快照日志的简单快照，否则使用复杂快照。

如本章前面所述，可以检索由本地数据库中快照创建的基本 SNAP$_*tablename*表。这也有助于改进查询性能，但以减慢刷新为代价。

如第8章中所述，减少网络流量的另一种方法是通过远程过程调用。第 8章还包括了有关调整SQL和应用程序设计的信息。如果数据库已正确地构造好，那么调整应用程序处理数据的方法将获得最显著的性能改善。

16.9 使用作业队列

为了支持快照刷新和其他复制功能，Oracle管理一个内部作业队列集。如果数据库中启用了作业队列(通过本章前面描述的JOB_QUEUE_PROCESSES和JOB_QUEUE_INTERVAL参数)，就可以把自己的作业提交给队列。可以用这些队列来替代操作系统的作业队列。

要管理内部作业队列，可使用 DBMS_JOB软件包中的SUBMIT、REMOVE、CHANGE、WHAT、NEXT_DATE、INTERVAL、BROKEN和RUN过程。最重要的是 SUBMIT、REMOVE和RUN过程。

SUBMIT过程有以下五个参数：

```
PROCEDURE SUBMIT
  ( job       OUT BINARY_INTEGER,
    what      IN  VARCHAR2,
    next_date IN  DATE DEFAULT sysdate,
    interval  IN  VARCHAR2 DEFAULT 'null',
    no_parse  IN  BOOLEAN DEFAULT FALSE);
```

job参数是一个输出参数，Oracle通过SYS.JOBSEQ序列为作业生成一个作业号。当提交一个作业时，首先要定义一个把作业号当作输出的变量。例如，下列程序定义一个变量并且每天提交一个作业来执行‘ myproc ’：

```
variable jobno number;
begin
  DBMS_JOB.SUBMIT(:jobno,'myproc',SysDate,SysDate+1);
  commit;
end;
/

print jobno

JOBNO
-----------
       8791
```

提交作业者可以修改作业 (通过BROKEN、CHANGE、INTERVAL、NEXT_DATE和WHAT过程)，从队列中删除作业(REMOVE过程)或强制运行(RUN过程)。对于这些过程，需要知道作业号。

如果在提交作业时没有记录该作业号，可以在数据库中查询DBMS_JOBS来查看提交的作业。

作业管理

可以通过BROKEN过程把作业改变为“ broken ”状态。如果将作业标记为“ broken ”，那么在下次安排它运行时将不再运行。 BROKEN过程的结构为：

```
PROCEDURE BROKEN
( job       IN  BINARY_INTEGER,
  broken    IN  BOOLEAN,
  next_date IN  DATE DEFAULT SYSDATE );
```

如果在前面已将作业设为 broken状态(通过设置 broken变量为 TRUE)，就可以通过将broken变量设置为FALSE使其不再是broken状态。

使用CHANGE过程就可以修改通过 what参数执行的代码、作业运行的下一个日期和作业运行的时间间隔。其结构是：

```
PROCEDURE CHANGE
( job       IN  BINARY_INTEGER,
  what      IN  VARCHAR2,
  next_date IN  DATE,
  interval  IN  VARCHAR2);
```

如果不为CHANGE变量规定一个值(或者将它们设置为NULL)，它们将保留原来的设置值。因此，可通过CHANGE过程修改作业的部分说明而不必重新设置作业的全部设置值。

如果只想修改作业执行的时间间隔，可以使用 INTERVAL过程，它的两个参数是作业号和新的时间间隔表达式。可以使用 NEXT_DATE过程修改作业要运行的下一个日期，它的两个参数是作业号和作业应运行的日期。

如果要改变执行的 PL/SQL代码，可以使用 WHAT过程。它的两个参数是作业号和执行的PL/SQL代码。INTERVAL、NEXT_DATE和WHAT过程并不提供任何 CHANGE过程不提供的功能。使用CHANGE时，把不想修改的变量设置为 NULL。

可以通过 REMOVE过程从作业队列中删除一个作业，这个过程以作业号作为它的唯一参数。如果有许多作业的特性要更改，可能要从作业队列中将其删除并重新提交。

若要在任何时刻强制作业运行，可使用 RUN过程。RUN过程是运行broken状态的作业的唯一方法，它以作业号作为输入变量。

作业队列一般用于管理数据库的内部功能 (例如分析数据库对象)。如果作业是数据库常规产品维护的一部分，那么，应通过数据库所驻留系统的日常调度机制来运行。尽管可以通过Oracle作业队列运行这些作业，但通过在数据库外面维护的中央控制作业管理工具来运行它们更为合适。对于不属于数据库产品控制部分的小型作业，可使用 Oracle内部作业队列。

要进一步详细了解 DBMS_JOB的过程，请查看 dbmsjob.sql文件。这个文件在 Oracle软件主目录下的/rdbms/admin子目录中。