

# 关于 Advanced Replication 的初步研究

Author: Kamus, yangtingkun, eygle

Mail: [kamus@itpub.net](mailto:kamus@itpub.net)

Date: 2004 年 3 月

一、	概述.....	1
二、	MR 的概念和构架.....	3
三、	冲突解决方案的概念和构架.....	9
四、	冲突解决机制的研究.....	10
五、	解决数据冲突—dbms_rectifier_diff 包.....	18
六、	MVR 的概念和构架.....	22
附录一。	多主体复制站点的配置步骤.....	34
附录二。	物化视图复制站点的配置步骤.....	38
附录三。	一些高级复制相关的包使用方法.....	46
附录四。	FAQ.....	50

## 一、 概述

1. Replication 使用分布式数据库技术在多个站点之间共享数据。
2. Replicated Database 和 Distributed Database 并不一样，在分布式数据库系统中数据在多个站点同时有效，但是一个表只会存在于一个站点中，而对于 Replication 来说相同的数据将同时存在于多个站点中。
3. 使用 replication 的原因：
  - 1) **Availability:** 也就是提供了优秀的 failover 保护
  - 2) **Performance:** 由于有多个 server，所以可以将用户业务分布在不同的 server 上
  - 3) **Disconnected computing:** 实体化视图允许用户在和 master 断开后使用数据库的子集，在重新连接上 master 之后再进行两者的同步。
  - 4) **Network load reduction:** 由于有多个 server，所以可以减少 master 的网络请求
  - 5) **Mass deployment:** 通过变量产生自定义的实体化视图以满足多种需求
4. 在不同的 Oracle 发行版本之间以及不同操作系统的 Oracle 之间都可以使用 Advanced

Replication。

5. Replication 中的几个概念:

- 1) **replication object**: 复制对象, 指需要作复制的对象 (object), 包括表, 索引, 存储过程等等。复制对象的更新遵循事务一致性规则 (transactionally consistent manner)。
- 2) **replication groups**: 复制组, 是复制对象 (replication object) 的集合称为 group, oracle 以 replication group 的形式来管理复制。一个组可以包含多个模式的 object, 一个模式也可以有多个组中的 object, 但是每个 replication object 都只能属于一个 replication group。
- 3) **replication sites**: 复制站点, 包含两种类型, 主体站点 (**master sites**) 和实体化视图站点 (**materialized view sites**)。一个 site 可以担任一个 replication group 中的 mater site 同时又担任另外一个 replication group 中的 materialized view site, 注意必须是另外一个组, 而不能是同一个 replication group。
- 4) **scheduled links**: 一个数据库链接 (database link), 包含一个由用户定义的计划, 来将需要更新的事务推到其它的 master sites, 当创建 scheduled link 的时候, oracle 将在本地任务队列中创建一个任务。
- 5) **master definition site**: 主体定义站点, 大部分的高级复制配置都需要在一个站点上作, 这个站点就是 maserdef site。

6. Replication 环境的几种类型

- 1) Multimaster Replication
- 2) Materialized View Replication (也可以称为是 Single Master Replication)
- 3) Multimaster and Materialized View Hybrid Configurations

Multimaster Replication 和 Materialized View Replication 的区别在于:

前者必须是全表复制而后者可以是 master 表的一部分

前者允许在每一个 transaction 之后都进行复制, 而后者是属于批处理复制

两者都使用 scheduled links 进行数据同步操作。

Materialized View Replication 中的 materialized view 可能有以下几种类型:

- 1) Read-Only Materialized Views: 只读的实体化视图
- 2) Updatable Materialized Views: 允许更新, 同时允许将更新复制到 master site
- 3) Writeable Materialized Views: 允许更新, 但是每次 refresh 的时候, 更新都会丢失

7. 介绍 Multimaster Replication 中的复制方式

- 1) Asynchronous replication  
在一个 master 上发生的变化将在推后的时间内更新到其他的 master 上
- 2) Synchronous replication  
在一个 master 上发生的变化将立刻更新到其他的 master 上
- 3) Procedural replication  
必须给每个 site 上的包都生成一个 wrapper, 所有的数据变化应该通过包中的存储过程完成, 当某个 master 上的 procedure 被调用, wapper 将保证其他 site 中的存储过程也被调用 (同步或者不同步)。将大量的数据操作放到一个 procedure 中,

然后对于 procedure 的调用将被同步, 用处在于在于有大数据量操作的时候可以减少网络负载。

## 二、 MR 的概念和构架

MR 是 Master Replication, 也就是主体站点复制的概念, 是高级复制区别于普通复制的一个重要的功能。本章节对于 MR 中出现的种种概念作详细解释。

MR 分为 single master 和 multi master 两种。single master 指一个 master site 支持多个 materialized view site, 而 multi master 则包含多个 master site。

Multimaster Replication 也被称为 peer-to-peer 或者  $n$ -way replication, 任何一个 master 上发生的变化都将被送到其它的 master 上。

### 1. 为什么使用 Multimaster Replication。

**Failover:** 当主数据库发生问题的时候, 可以通过配置 Oracle Net 来实现 automatic connect-time failover, 需要将客户端的 tnsnames.ora 中的 FAILOVER\_MODE 参数设置为 ON。同时主数据库正常的时候, 其它的 master site 仍然可以作为一个具有完全功能的数据库来支持其它业务, 比如报表等。

**Load Balancing:** 提供读动态平衡以及更本地化的数据存取。

### 2. 比较 RAC (Oracle Real Application Clusters) 和 Advanced Replication

**Load Balancing:** 高级复制提供读动态平衡, 而 RAC 则提供读写动态平衡。因为每一次写操作都会在所有的 replication site 上体现, 所以高级复制不能提供写动态平衡。

**Survivability:** 高级复制提供更加强有力的灾难恢复功能, 因为高级复制环境中的各个 site 可以位于物理上的不同地点, 而 RAC 因为使用的是磁盘阵列或其他类型的并行系统, 所以通常在同一个物理地点。

**Interoperability:** 高级复制可以在不同平台和操作系统的 Oracle 之间实现, 而 RAC 环境则必须运行在相同的平台上。

### 3. Multimaster Replication Process

#### **Asynchronous Replication:**

说明: 使用非同步复制能够减少网络资源和硬件资源的消耗, 但是不同的 master sites 之间会有一段时间不同步, 并且可能会造成数据冲突。

以下描述非同步复制的过程:

- 1) 用户执行 DML 操作或者执行 replicated procedure 的 wrapper, 当一个 table 被设定为需要复制, 那么对于此表的任何 DML 操作都会被捕获并且复制到其他 master site。对于每一行被插入, 更新或者删除的数据都将由一个内部触发器来创建一个 deferred remote procedure call (RPC) 并且放在 deferred transaction queue 中, 如果一个存储过程被设置为需要复制并且它的 wrapper 被执行, 则这个 procedure call 被放置在 transaction queue 中。由于内部触发器是由 Oracle 本身内部维护的, 所以可以以最小的系统资源消耗来很快地获取需要复制的对象的变化。

- 2) deferred transaction queue 中保存着所有的 deferred RPCs。每个 site 都有一个事务队列，这个队列可以被多个 replication group 共用。
- 3) 在指定的间隔之后或者被手工调用，事务将被传递到其他的 site，每个 site 都可能有不同的间隔。
- 4) 事务在这些 site 上被应用，如果出错该事务将被放置到一个错误队列中，以备 DBA 检查处理，如果出现数据冲突，冲突解决方法将被调用，如果冲突无法解决那么将被记录在错误队列中
- 5) 当事务在所有的 remote master sites 上被成功执行之后，并不会从源 site 的事务队列中立刻删除，删除工作将由另外的 purge job 来执行，此 job 的执行间隔可以由用户来定义。

### Synchronous Replication:

说明：同步复制始终在同一个 transaction 中完成，如果整个环境中的任何一个 site 没有成功执行事务，那么整个 transaction 将被回滚，包括源 site。这就保证了数据一致性。

以下描述同步复制的过程：

- 1) 用户执行 DML 操作或者执行 replicated procedure 的 wrapper，操作被内部触发器立刻捕获。
- 2) 事务被传递到其他 site 并且立刻执行，任何一个 site 出错，就回滚整个事务。

#### 4. 冲突解决方案的概念

冲突的类型：更新冲突，唯一性冲突，删除冲突。

当发生冲突的时候，冲突解决方法将被调用以解决冲突，如果无法解决，则被记录到目标站点的错误队列中。记录到错误队列中的冲突只能由数据库管理员手动解决。

为了实现冲突解决方案，可能会需要修改表结构，比如如果使用最新时间戳的解决方案，那么就应该在表中添加一个 timestamp 列。

#### 5. 配置高级复制的工具

- 1) 图形界面：Oracle Enterprise Manager 提供了一个友好的 GUI 界面用以配置高级复制。
- 2) 命令行方式：Oracle 提供了一套 replication management application programming interface (API)来支持用户编写自定义的脚本用以配置高级复制，这些 API 是一系列的 PL/SQL packages。实际上 GUI 界面的高级配置工具也是调用这些 API 来完成配置的。

注意点：

在高级复制环境中的对于需要复制的对象作任何 DDL 操作，都应该使用高级复制配置工具来作，比如利用 DBMS\_REPCAT 包中的相应存储过程。在有些场合下也可以用导出导入（EXP/IMP）来创建复制对象。在 SQL\*PLUS 中直接执行的任何 DDL 操作都不会被复制到其它的 site 上。

#### 6. 高级复制中的几个角色

**Replication Administrator:** 默认名称是 repadmin，也可以修改。

**Propagator:** 一个高级复制环境中可能有多个 RA 来管理不同的 schema，但是只能有一个 propagator 将延迟处理事务队列中的事务传递到目标站点。

**Receiver:** 负责接收和处理从 propagator 处传来的延迟处理事务。可以通过

DBMS\_REPCAT\_ADMIN 包中 REGISTER\_USER\_REPGROUP 存储过程来注册一个 receiver。

## 7. Database Links

数据库链接在高级复制环境中提供了数据传送的通道，在一个 MMR 环境中，如果有 N 个 Master Site，就会有 N-1 个数据库链接。在 MVR 环境中，则只需要从实体化视图站点上到主站点的数据库链接。如果使用设置向导来创建 dblink，则会在 USING 后面使用连接描述字串，而不是连接服务名，这样在对方数据库发生变化的时候，就必须删除重建现有的 dblink，所以我们应该手动设定 dnlink，从而在 USING 后面使用连接服务名。这样即使对方数据库改变，我们也只需要修改 tnsnames.ora 中的配置即可，而不需要重建 dblink。

## 8. 可以进行复制的对象

- **Tables**

当一个对象被复制到目标站点上时，复制支持不会自动生成。利用这个特点可以快速地发布一个标准的数据库环境到另外的站点上。

- **Indexes**

作为约束的索引，当表在主站点上被创建的时候，会自动在复制站点上创建，但是对于提高性能的索引则不会被自动创建而必须手动指定。对于索引被复制到目标站点之后，等同于本地的索引，不需要再添加复制支持。

- **Packages and Package Bodies**

存储过程中的所有参数必须是 IN 的，OUT 和 INOUT 不被支持。存储过程和函数也必须定义在包中，单独的存储过程和函数无法进行复制。

- **Procedures and Functions**

虽然单独的存储过程和函数无法进行复制，但是仍然可以在高级复制环境中利用复制来将单独的存储过程和函数发布到远程的站点上，就像在远程站点本地创建的一样。

- **User-Defined Types and Type Bodies**

所有的用户定义类型在所有的复制站点上都必须存在而且必须完全相同。

- **Triggers**

一个比较重要的应用就是在 DML 操作的时候在表的 timestmap 列中插入当前的系统时间。为了防止触发器被重复调用，必须要使用 API 来判断 DML 操作是在本地发起的还是通过高级复制传递过来的。如下例：

```
CREATE OR REPLACE TRIGGER hr.insert_time
  BEFORE
    INSERT OR UPDATE ON hr.employees FOR EACH ROW
  BEGIN
    IF DBMS_REPUTIL.FROM_REMOTE = FALSE THEN
      :NEW.TIMESTAMP := SYSDATE;
    END IF;
  END;
```

- **Views, Object Views, and Synonyms**

只是简单地复制到其它站点，不会产生任何内部触发器或者包来监控这些被复制的对象的改变。由于是复制对象，所以仍然可以使用高级复制工具或者 API 来进行修改和删除。

- **Indextypes**

必须手工指定复制。可以用高级复制工具或者 CREATE\_MASTER\_REOBJECT 存储过程。

- **User-Defined Operators**

复制情况跟视图，同义词等相同，只是简单的复制而已。

注意：高级复制不支持 `sequence`。如果想实现高级复制环境中的序列唯一性，可以有以下几种方法：

1. 使用 `SELECT SYS_GUID() OID FROM DUAL`;这样将会产生全球唯一的 GUID
2. 在序列前面添加站点名称，比如唯一的 `GLOBAL_NAME`
3. 在各个复制站点规划不会重复的序列，比如站点 A 的序列从 1 开始，以 10 递增，站点 B 的序列从 3 开始，以 10 递增。

## 9. 高级复制环境中的队列

Oracle 利用 `Internal Triggers` 来捕获对象变化，并且生成 `RPCs`(remote procedure calls)，`RPCs` 中包含目标站点上的 `internal procedure` 的执行命令以及需要复制的数据，`RPCs` 存储在 `deferred transaction queue` 中，当一个 `RPC` 到达目标站点，该站点上相应的 `internal procedure` 将被运行以应用 `RPC` 来完成复制操作。

高级复制环境中的队列包含 `Deferred Transaction Queue`，`Error Queue` 和 `Job Queue`。`Job Queue` 中包含的作业有三种：将延迟事务推到远程主站点的作业，将已经应用过的事务从延迟事务处理队列中删除的作业，刷新实体化视图更新组的作业。

## 10. 管理请求 (Administrative Request) — 管理机制

什么是管理请求？

在 `DBA_REPCATLOG` 视图中查看 `Administrative Requests` 的状态。当管理请求在所有的主站点上成功执行以后，管理请求将从所有的主站点包括主体定义站点中的管理请求队列中，也就是 `DBA_REPCATLOG` 视图中删除。

`DBA_REPCATLOG` 视图中管理请求的几种状态：

- 1) **READY**: 表示请求准备被执行。如果长时间处于该状态，可以手动执行 `DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN` 存储过程来执行请求。
- 2) **AWAIT\_CALLBACK**: 这种状态只会出现在主体定义站点上，表示正在等待其它的主站点执行请求并且返回结果。
- 3) **ERROR**: 表示请求执行错误
- 4) **DO\_CALLBACK**: 这种状态只会出现在非主体定义站点上，表示要通知主体定义站点请求执行的结果。

## 11. 主体组 (Master Group) — 组织机制

在高级复制环境中，Oracle 用复制组来管理复制对象。而在多主体复制 (`multimaster replication`) 环境中复制组就被称为主体组 (`Master Group`)。在不同的复制站点上的相应主体组中必须包含相同的复制对象。

## 12. Column Groups — 组织机制

`Column Group` 是在冲突解决方案中扮演角色的多个字段的集合。如果组中的某个字段引发了冲突，那么其余的字段可以用来作解决这个冲突。打个比方，如果一个表的 `column group` 中包含 `price` 和 `timestamp` 字段，那么当启用时间戳冲突解决方案 (`timestamp conflict resolution routine`) 时，`timestamp` 字段就可以用来解决 `price` 字段中发生的冲突。

可能刚开始的时候会想把表中的所有字段都放入一个 Column Group 中，这样确实使配置和管理都更简单了些，但是却会降低复制的性能并且可能会引发潜在的数据冲突。在后面的性能机制部分，大家将会看到如果一个 column group 中发生了冲突，那么 oracle 的最小化通信功能（minimum communication feature）将不会从其它的 column group 中传递来数据。所以将所有的字段全部放入一个 column group 将减弱最小化通信功能带来的好处，除非使用了 DBMS\_REPCAT 包中的 SEND\_OLD\_VALUES 和 COMPARE\_OLD\_VALUES 存储过程。在后面的冲突解决方案的概念和体系结构章节中将有更详细的描述。

### 13. 传播类型 — 传播机制

异步数据复制通常也被称为：**store-and-forward data replication**

同步数据复制通常也被称为：**real-time data replication**

由于同步复制采取的锁机制，所以当同时更新同一行数据时，会产生死锁的现象。当同步更新一个复制表时，Oracle 首先锁住本地行，然后使用一个 AFTER ROW 触发器来锁住远端的行。当事务在所有的站点都提交之后，Oracle 才会解锁。同步数据复制极为依赖系统和网络的可用性，因为只要当复制环境中的所有站点都可用时，事务才能正常进行。

混合模式的数据复制：

- 1) 假设创建了 A 是 masterdef site，然后添加了 B 为同步数据复制，再添加 C 为非同步复制，那么此时 AB 之间是同步，AC 和 BC 之间都是非同步。
- 2) 假设创建了 A 是 masterdef site，然后添加了 C 为非同步数据复制，再添加 B 为同步复制，那么此时 AB 和 BC 之间是同步，AC 之间是非同步。

### 14. Initiating 方法 — 展开机制

当使用同步复制的时候，DML 传播被立刻处理并且被自动展开。

如果是使用异步复制，那么可以用下面的方法传播延迟事务：

计划作业：大部分场合，都是利用计划作业在指定的时间间隔后自动传播延迟事务。

手动传播：如果不想等待计划作业的自动传播，也可以利用存储过程或者复制管理工具来手动传播改动。

### 15. 并行传播（Parallel Propagation） — 性能机制

### 16. 最小化通信（Minimum Communication） — 性能机制

### 17. 延迟秒数（Delay Seconds） — 性能机制

上面三个参数（15，16，17 值得好好研究并且进行调整，以后会补齐这部分内容）

### 18. 复制保护机制：

在多主体复制环境中，Oracle 将保证就算发生错误的时候，事务传播也不会丢失，同样同一个事务也不可能传播两次。

正确的传播并不以为着延迟事务在远程站点就正确执行了，可能因为无法解决的冲突或者说远程站点磁盘空间不足等原因延迟事务执行失败，那么这样的错误将会记录在远程站点的错误队列中。

## 19. 数据传播的依赖性维护

非并行传播中，Oracle 按照本站点的事务 commit 顺序来在远程站点应用事务。而并行传播中，Oracle 则会记录最新事务产生的 SCN，如果存在事务之间的依赖性，那么 Oracle 将先在远程站点应用比这个 SCN 小或者等于这个 SCN 的事务，然后再应用这个最新的事务，这是一个非并行的应用，只有当不存在事务依赖性的时候，才会真正利用并行来应用延迟事务。

记录 SCN 有两种方式，一种是数据块级别的，一种是行级别的。

当创建表的时候，如果使用了：

**NOROWDEPENDENCIES**，这是默认属性，那么 Oracle 将会以数据块级别方式来记录 SCN。这样存储在同一个数据块中的多行记录都只会有一个最新的 SCN，旧的会被新生成的覆盖。

**ROWDEPENDENCIES**，那么将对表中的每一行都记录 SCN。在同一个数据块中的多行记录将分别保留自己的 SCN，这样每一行记录都需要额外的 6 个字节的存储空间。但是这将提高并行传播时应用延迟事务的效率。（使用这个特性，要求数据库的初始化参数中 COMPATIBLE=9.0.1 或者更高）。可以使用以下 SQL 来检查那些表启用了这个特性：

```
SQL> SELECT OWNER, TABLE_NAME FROM DBA_TABLES
      WHERE DEPENDENCIES = 'ENABLED';
```

如果没有使用 ROWDEPENDENCIES，那么我们可以设法让事务依赖性最小，这样来达到提高复制环境应用效率的目的。比如我们可以创建多个 freelist，这样可以在大量 insert 的时候将不同事务更新的数据放置到不同的数据块中。

另外在程序设计的时候，我们也应该尽量避免大量的事务同时更新同一张小表的现象出现，比如说有些应用会设计一张小表来模拟序列（Sequence），用以生成唯一的主键。这样就会迫使多个事务同时更新同一个数据块。对于这种情况，我们应该改为使用 Sequence 并且缓存 Sequence 的生成。

## 20. 冲突解决机制

为了正确地侦测复制冲突，Oracle 必须能够找到在不同的站点之间对于相关行的唯一标示。这就要求在复制环境中，每个表都必须有主键，如果没有主键，那么也必须指定多个字段的组合来作为唯一标示。

Oracle 自己提供了以下几种冲突解决方案：

- 1) Latest and Earliest Timestamp
- 2) Overwrite and Discard
- 3) Maximum and Minimum
- 4) Additive and Average
- 5) Timestamp
- 6) Priority Group
- 7) Site Priority

如果上述 Oracle 提供的解决方案无法满足应用的需求，那么也可以利用 PL/SQL 来编写自定义的冲突解决方案。

下面章节我们将进一步研究冲突解决方案。

### 三、冲突解决方案的概念和构架

基本上我们在设计系统的时候，应该尽量避免产生冲突，但是如果必须允许在多个主体站点同时对复制对象的更改，那么我们就一定要考虑冲突解决方案。

对于普通的插入（主键冲突），更新，删除产生的冲突很容易理解，但是还有一些其它的情况也会产生冲突，比如在 3 个或者 3 个以上的复制主站点环境中，可能会产生下面例子中的这种 Ordering Conflicts，其实也是更新冲突的一个例子。

这个复制环境中 A,B,C 三个主站点，每个站点都设置了优先级，A 是 30，B 是 25，C 是 10，而 x 则是被分配了 **site-priority** 冲突解决方案的 column group 中的一列。

Time	Action	Site A	Site B	Site C
1	所有的主体站点上 x = 2.	2	2	2
2	站点 A 更新 x = 5.	5	2	2
3	站点 C 由于故障宕机了，或者网络出现故障.	5	2	down
4	站点 A 将更新推到站点 B. 站点 A 和站点 B 上 x = 5.  站点 C 仍然不可用. 这个更新事务仍然保留在站点 A 的队列中.	5	5	down
5	站点 C 修复了，此时站点 C 上 x = 2. 而站点 A 和站点 B 上 x = 5.	5	5	2
6	站点 B 将 x = 5 更新为 x = 7.	5	7	2
7	站点 B 将更新推到站点 A. 站点 A 和站点 B 上 x = 7. 站点 C 上 x = 2.	7	7	2
8	站点 B 将更新推到站点 C. 站点 C 认为旧值 x = 2; 站点 B 推过去的旧值 x = 5. Oracle 检测到冲突，然后应用站点 B 上的更新来解决冲突。因为站点 B 上设置了优先级 25 比站点 C 上设置的优先级 10 要高。 所有站点上 x = 7.	7	7	7
9	站点 A 将延迟事务 (x = 5) 推到站点 C. Oracle 检测到冲突，因为站点 C 上的当前值(x = 7) 和站点 A 上的旧值(x = 2)不符合。  站点 A 比站点 C 有更高的优先级(30). Oracle 用这个过时的更新解决冲突，这样站点 C 上的 x = 5.			

Time	Action	Site A	Site B	Site C
	因为这样的 ordering conflict, 整个复制环境就不再一致了.			

## 四、冲突解决机制的研究

实际上 Oracle 的 dbms\_rectifier\_diff.DIFFERENCES 过程，内部操作就是执行连个 minus 操作把两边的差异记录下来，作为冲突解决的数据。

这部分后台操作可以通过跟踪 Oracle 进程得到：

```
SQL> alter session set events '10046 trace name context forever,level 12';
```

Session altered.

Elapsed: 00:00:00.02

```
SQL> begin dbms_rectifier_diff.DIFFERENCES(
2 SNAME1 =>'HAWA',
3 ONAME1 =>'TEST',
4 REFERENCE_SITE =>'AVATAR.COOLYOUNG.COM.CN',
4 SNAME2 =>'HAWA',
6 ONAME2 =>'TEST',
7 COMPARISON_SITE =>'AUTHAA.COOLYOUNG.COM.CN',
8 WHERE_CLAUSE =>NULL,
9 COLUMN_LIST =>NULL,
10 MISSING_ROWS_SNAME =>'HAWA',
11 MISSING_ROWS_ONAME1 =>'MISSING_ROWS_TEST',
12 MISSING_ROWS_ONAME2 =>'MISSING_LOCATION_TEST',
13 MISSING_ROWS_SITE =>'AVATAR.COOLYOUNG.COM.CN',
14 MAX_MISSING =>500,
15 COMMIT_ROWS =>100
16 );
17 end;
18 /
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:01.97

```
SQL> alter session set events '10046 trace name context off';
```

从跟踪文件中我们可以清晰的看到(注意你所定义的所有参数在此都会有所体现)：

1.首先是一个正向 Minus

```
DECLARE
row_count BINARY_INTEGER := 0;
missing_rows BINARY_INTEGER := 0;
arowid ROWID;

CURSOR c
IS
SELECT "DATLOGONTIME", "NUMGENDER", "NUMSTATUS", "NUMUSERID", "VC2IP",
"VC2USERNAME"
FROM "HAWA"."TEST"
MINUS
SELECT "DATLOGONTIME", "NUMGENDER", "NUMSTATUS", "NUMUSERID", "VC2IP",
"VC2USERNAME"
FROM "HAWA"."TEST"@authaa.coolyoung.com.cn;
BEGIN
FOR r IN c
LOOP
missing_rows := missing_rows + 1;

IF missing_rows > 500
THEN
COMMIT;
EXIT;
END IF;

INSERT INTO "HAWA"."MISSING_ROWS_TEST"
("DATLOGONTIME", "NUMGENDER", "NUMSTATUS",
"NUMUSERID", "VC2IP", "VC2USERNAME"
)
VALUES (r."DATLOGONTIME", r."NUMGENDER", r."NUMSTATUS",
r."NUMUSERID", r."VC2IP", r."VC2USERNAME"
);

SELECT ROWID
INTO arowid
FROM "HAWA"."MISSING_ROWS_TEST"
WHERE ( datlogontime = r."DATLOGONTIME"
OR (datlogontime IS NULL AND r."DATLOGONTIME" IS NULL)
)
AND ( numgender = r."NUMGENDER"
OR (numgender IS NULL AND r."NUMGENDER" IS NULL)
)
AND ( numstatus = r."NUMSTATUS"
```

```

OR (numstatus IS NULL AND r."NUMSTATUS" IS NULL)
)
AND (numuserid = r."NUMUSERID")
AND (vc2ip = r."VC2IP" OR (vc2ip IS NULL AND r."VC2IP" IS NULL))
AND ( vc2username = r."VC2USERNAME"
OR (vc2username IS NULL AND r."VC2USERNAME" IS NULL)
);

INSERT INTO "HAWA"."MISSING_LOCATION_TEST"
(present, absent, r_id
)
VALUES ('AVATAR.COOLYOUNG.COM.CN', 'AUTHAA.COOLYOUNG.COM.CN',
arowid
);

row_count := row_count + 1;

IF row_count >= 100
THEN
COMMIT;
row_count := 0;
END IF;
END LOOP;

COMMIT;
END;

```

2.其次是一个反向 Minus

```

DECLARE
row_count BINARY_INTEGER := 0;
missing_rows BINARY_INTEGER := 0;
arowid ROWID;

CURSOR c
IS
SELECT "DATLOGONTIME", "NUMGENDER", "NUMSTATUS", "NUMUSERID", "VC2IP",
"VC2USERNAME"
FROM "HAWA"."TEST"@authaa.coolyoung.com.cn
MINUS
SELECT "DATLOGONTIME", "NUMGENDER", "NUMSTATUS", "NUMUSERID", "VC2IP",
"VC2USERNAME"
FROM "HAWA"."TEST";
BEGIN
FOR r IN c

```

```

LOOP
missing_rows := missing_rows + 1;

IF missing_rows > 500
THEN
COMMIT;
EXIT;
END IF;

INSERT INTO "HAWA"."MISSING_ROWS_TEST"
("DATLOGONTIME", "NUMGENDER", "NUMSTATUS",
"NUMUSERID", "VC2IP", "VC2USERNAME"
)
VALUES (r."DATLOGONTIME", r."NUMGENDER", r."NUMSTATUS",
r."NUMUSERID", r."VC2IP", r."VC2USERNAME"
);

SELECT ROWID
INTO arowid
FROM "HAWA"."MISSING_ROWS_TEST"
WHERE ( datlogontime = r."DATLOGONTIME"
OR (datlogontime IS NULL AND r."DATLOGONTIME" IS NULL)
)
AND ( numgender = r."NUMGENDER"
OR (numgender IS NULL AND r."NUMGENDER" IS NULL)
)
AND ( numstatus = r."NUMSTATUS"
OR (numstatus IS NULL AND r."NUMSTATUS" IS NULL)
)
AND (numuserid = r."NUMUSERID")
AND (vc2ip = r."VC2IP" OR (vc2ip IS NULL AND r."VC2IP" IS NULL))
AND ( vc2username = r."VC2USERNAME"
OR (vc2username IS NULL AND r."VC2USERNAME" IS NULL)
);

INSERT INTO "HAWA"."MISSING_LOCATION_TEST"
(present, absent,
r_id
)
VALUES ('AUTHAA.COOLYOUNG.COM.CN', 'AVATAR.COOLYOUNG.COM.CN',
arowid
);

row_count := row_count + 1;

```

```

IF row_count >= 100
THEN
COMMIT;
row_count := 0;
END IF;
END LOOP;

COMMIT;
END;

```

经过这两个步骤的操作，Oracle 定位了冲突数据。

可是注意，如果在解决这个问题时你没有挂起复制，Oracle 得到的数据可能是存在问题的。而且，如果你不指定 `column list`,那么两边的数据可能会因为某些特殊字段(如时间字段)的特殊处理而存在差异。

那么这时候手工介入不可避免。

我们首先先把两个重要参数的用法说明一下。  
一个是 `WHERE_CLAUSE`，另外一个 `COLUMN_LIST`。

`WHERE_CLAUSE` 用于限定进行差异比较的范围，这可以极大的缩减结果集的数量，使用索引加快访问速度等。

比如我这里使用 `NUMGENDER=1`，只比较性别为"女"这一部分用户数据。

`COLUMN_LIST` 用于限定比较字段，如果你能通过某个字段，如主键等确定数据差异，那么你完全可以只比较单个字段。

而且显然可以轻易通过全索引扫描来完成比较，加快比较速度。

我这里使用 `NUMUSERID`，用户 ID 来比较。

但是注意，这样的比较结果中将只包含 `NUMUSERID` 信息，当然我们可以轻易通过 `NUMUSERID` 和原表的比较补全 `MISSING_ROWS_TEST` 表的信息。

```

begin dbms_rectifier_diff.DIFFERENCES(
SNAME1 =>'HAWA',
ONAME1 =>'TEST',
REFERENCE_SITE =>'AVATAR.COOLYOUNG.COM.CN',
SNAME2 =>'HAWA',
ONAME2 =>'TEST',
COMPARISON_SITE =>'AUTHAA.COOLYOUNG.COM.CN',
WHERE_CLAUSE =>'NUMGENDER=1',
COLUMN_LIST =>'NUMUSERID',
MISSING_ROWS_SNAME =>'HAWA',
MISSING_ROWS_ONAME1 =>'MISSING_ROWS_TEST',

```

```
MISSING_ROWS_ONAME2 =>'MISSING_LOCATION_TEST',
MISSING_ROWS_SITE =>'AVATAR.COOLYOUNG.COM.CN',
MAX_MISSING =>500,
COMMIT_ROWS =>100
);
end;
/
```

这段代码供参考。

Ok, 我们继续前面的讨论。

我们提到, 如果存在差异, 通常需要手工介入。

清楚了 DIFFERENCES 的原理, 实际上我们完全可以手工来完成这个过程。

以下是我的手工操作步骤, 目的是为了准确性及减轻数据库压力:

### 1. 首先创建一个 ID 差异表

这个表不是必须的, 这里是为了清晰

```
SQL> create table hawa.prof as select NUMUSERID from hawa.hw_user where 1=0;
```

Table created.

Elapsed: 00:00:00.16

### 2. 根据主键找到差异记录

注意这里取决于你的数据库产生差异的原因, 我的差异由于初始数据不同步, 即 A 全包含 B 并且,  $A > B$ 。

```
SQL> insert into hawa.prof
```

```
2 select * from
```

```
3 (
```

```
4 select NUMUSERID from hawa.HW_USERPROFILE
```

```
5 minus
```

```
6 select NUMUSERID from hawa.HW_USERPROFILE@authaa)
```

```
7 /
```

263 rows created.

Elapsed: 00:00:32.49

### 3. 创建记录表

```
SQL> create table hawa.missing_rows_hw_userprofile
```

```
2 as
3 select * from hawa.hw_userprofile where 1=0;
```

Table created.

Elapsed: 00:00:00.12

#### 4.创建位置(Location)表

注意这里 Oracle 需要记录缺失方向，和具体记录的 ROWID，这个 ROWID 来自 missing\_rows\_hw\_userprofile。

```
SQL> create table hawa.MISSING_LOC_hw_userprofile (
2 present VARCHAR2(128),
3 absent VARCHAR2(128),
4 r_id ROWID);
```

Table created.

Elapsed: 00:00:00.04

#### 4.根据差异信息查询到完整信息

```
SQL> insert into hawa.missing_rows_hw_userprofile
2 select * from hawa.hw_userprofile where NUMUSERID in
3 (select * from hawa.prof);
```

263 rows created.

Elapsed: 00:00:00.06

```
SQL> commit;
```

Commit complete.

Elapsed: 00:00:00.02

#### 5.构造位置信息

注意这里的方向信息及 ROWID 信息。

```
SQL> insert into hawa.MISSING_LOC_hw_userprofile
2 select 'AVATAR.COOLYOUNG.COM.CN','AUTHAA.COOLYOUNG.COM.CN',rowid from
hawa.missing_rows_hw_userprofile;
```

263 rows created.

Elapsed: 00:00:00.00

```
SQL> commit;
```

Commit complete.

Elapsed: 00:00:00.06

## 6.纠正数据冲突

```
SQL> BEGIN DBMS_RECTIFIER_DIFF.RECTIFY(  
2 SNAME1 =>'HAWA',  
3 ONAME1 =>'HW_USERPROFILE',  
4 REFERENCE_SITE =>'AVATAR.COOLYOUNG.COM.CN',  
5 SNAME2 =>'HAWA',  
6 ONAME2 =>'HW_USERPROFILE',  
7 COMPARISON_SITE =>'AUTHAA.COOLYOUNG.COM.CN',  
8 COLUMN_LIST =>NULL,  
9 MISSING_ROWS_SNAME =>'HAWA',  
10 MISSING_ROWS_ONAME1 =>'MISSING_ROWS_HW_USERPROFILE',  
11 MISSING_ROWS_ONAME2 =>'MISSING_LOC_HW_USERPROFILE',  
12 MISSING_ROWS_SITE =>'AVATAR.COOLYOUNG.COM.CN',  
13 COMMIT_ROWS =>100  
14 );  
15 END;  
16 /
```

PL/SQL procedure successfully completed.

Elapsed: 00:00:03.53

## 7.验证结果

```
SQL> select count(*) from hawa.HW_USERPROFILE;
```

```
COUNT(*)
```

```
-----  
1746300
```

Elapsed: 00:00:02.22

```
SQL> select count(*) from hawa.HW_USERPROFILE@authaa;
```

```
COUNT(*)
```

```
-----  
1746300
```

Elapsed: 00:00:00.21

```
SQL> select count(*) from hawa.HW_USERPROFILE;
```

```
COUNT(*)
```

```
-----  
1746300
```

Elapsed: 00:00:00.59

```
SQL>select count(*) from hawa.HW_USERPROFILE@authaa;
```

```
COUNT(*)
```

```
-----  
1746300
```

Elapsed: 00:00:00.20

```
SQL> select NUMUSERID from hawa.HW_USERPROFILE
```

```
2 minus
```

```
3 select NUMUSERID from hawa.HW_USERPROFILE@authaa ;
```

```
no rows selected
```

Elapsed: 00:00:23.51

```
SQL>
```

## 五、 解决数据冲突—dbms\_rectifier\_diff 包

很多时候在高级复制中可能存在数据冲突和不一致现象。

Oracle 提供的 dbms\_rectifier\_diff 包可以用于解决该冲突。

以下通过实例来说明一下该 Package 的用法。

### 1.创建复制组及复制对象

```
SQL> execute dbms_repcat.create_master_repgroup('rep_tt');
```

```
PL/SQL procedure successfully completed
```

```
SQL> select gname, master, status from dba_repgroup;
```

```
GNAME                                MASTER STATUS
```

```
-----  
REP_TT                                Y          QUIESCED
```

```
SQL> execute dbms_repcat.create_master_repobject(sname=>'hawa',oname=>'test',
type=>'table',use_existing_object=>true,gname=>'rep_tt',copy_rows=>false);
```

PL/SQL procedure successfully completed

```
SQL>
```

```
SQL> execute dbms_repcat.generate_replication_support('hawa','test','table');
```

PL/SQL procedure successfully completed

```
SQL> select gname, master, status from dba_repgroup;
```

GNAME MASTER STATUS

```
-----
REP_TT Y QUIESCED
```

```
SQL> select * from dba_repobject;
```

```
SNAME ONAME TYPE STATUS GENERATION_STATUS ID OBJECT_COMMENT GNAME
MIN_COMMUNICATION REPLICATION_TRIGGER_EXISTS INTERNAL_PACKAGE_EXISTS
GROUP_OWNER NESTED_TABLE
```

```
-----
-----
HAWA TEST TABLE VALID GENERATED 8620 REP_TT Y Y Y PUBLIC N
HAWA TEST$RP PACKAGE VALID 8641 SYSTEM-GENERATED: REPLICATION REP_TT
PUBLIC
HAWA TEST$RP PACKAGE BODY VALID 8677 SYSTEM-GENERATED: REPLICATION
REP_TT PUBLIC
```

3 rows selected

```
SQL>
```

```
SQL> execute
dbms_repcat.add_master_database(gname=>'rep_tt',master=>'AUTHAA.COOLYOUNG.C
OM.CN',use_existing_objects=>true, copy_rows=>false, propagation_mode =>
'synchronous');
```

PL/SQL procedure successfully completed

```
SQL> execute dbms_repcat.resume_master_activity('rep_tt',true);
```

PL/SQL procedure successfully completed

```
SQL> select * from dba_repgroup;
```

SNAME	MASTER	STATUS	SCHEMA_COMMENT	GNAME	FNAME
RPC_PROCESSING_DISABLED	OWNER				
-----					
REP_TT	Y	NORMAL	REP_TT	N	PUBLIC

2.创建保存冲突数据的数据表

a.missing\_rows 表用以保存冲突行

```
SQL> create table hawa.missing_rows_test
```

```
2 as
```

```
3 select * from hawa.test where 1=0;
```

Table created

b.用于保存缺失行位置及 rowid

```
SQL> create table hawa.MISSING_LOCATION_TEST (
```

```
2 present VARCHAR2(128),
```

```
3 absent VARCHAR2(128),
```

```
4 r_id ROWID);
```

Table created

3.使用 dbms\_rectifier\_diff.DIFFERENCES 查找缺失记录

```
SQL> begin dbms_rectifier_diff.DIFFERENCES(
```

```
2  SNAME1                =>'HAWA',
3  ONAME1                =>'TEST',
4  REFERENCE_SITE        =>'AVATAR.COOLYOUNG.COM.CN',
5  SNAME2                =>'HAWA',
6  ONAME2                =>'TEST',
7  COMPARISON_SITE       =>'AUTHAA.COOLYOUNG.COM.CN',
8  WHERE_CLAUSE          =>NULL,
9  COLUMN_LIST           =>NULL,
10 MISSING_ROWS_SNAME    =>'HAWA',
11 MISSING_ROWS_ONAME1   =>'MISSING_ROWS_TEST',
12 MISSING_ROWS_ONAME2   =>'MISSING_LOCATION_TEST',
13 MISSING_ROWS_SITE     =>'AVATAR.COOLYOUNG.COM.CN',
14 MAX_MISSING           =>500,
15 COMMIT_ROWS           =>100
```

```
16 );  
17 end;  
18 /
```

PL/SQL procedure successfully completed

冲突记录被保存在我们创建的指定表中

```
SQL> select count(*) from hawa.missing_rows_test;
```

```
COUNT(*)
```

```
-----
```

```
172
```

共有 172 条差异记录

```
SQL> select count(*) from hawa.test;
```

```
COUNT(*)
```

```
-----
```

```
548
```

```
SQL> select count(*) from hawa.test@authaa;
```

```
COUNT(*)
```

```
-----
```

```
376
```

```
SQL> select count(*) from hawa.missing_location_test;
```

```
COUNT(*)
```

```
-----
```

```
172
```

#### 4.使用 DBMS\_RECTIFIER\_DIFF.RECTIFY 进行数据整合

首先需要注意的是:

RECTIFY 过程使用 DIFFERENCES 产生的数据进行数据调整。

在第一个表中存在, 在第二个表中不存在的数据将被插入第二张表。

在第二个表中存在, 在第一个表中不存在的数据将被从第二张表中删除。

另外, 在这个数据纠正过程中, 你可以使用 `dbms_repcat.suspend_master_activity` 将复制组暂时挂起。

这样便于保证数据完整性。

但这不是必须的，如果复制一直激活，可能会有新的冲突出现。

```
SQL> BEGIN DBMS_RECTIFIER_DIFF.RECTIFY(  
 2  SNAME1                =>'HAWA',  
 3  ONAME1                =>'TEST',  
 4  REFERENCE_SITE       =>'AVATAR.COOLYOUNG.COM.CN',  
 5  SNAME2                =>'HAWA',  
 6  ONAME2                =>'TEST',  
 7  COMPARISON_SITE      =>'AUTHAA.COOLYOUNG.COM.CN',  
 8  COLUMN_LIST          =>NULL,  
 9  MISSING_ROWS_SNAME   =>'HAWA',  
10  MISSING_ROWS_ONAME1  =>'MISSING_ROWS_TEST',  
11  MISSING_ROWS_ONAME2  =>'MISSING_LOCATION_TEST',  
12  MISSING_ROWS_SITE    =>'AVATAR.COOLYOUNG.COM.CN',  
13  COMMIT_ROWS          =>100  
14 );  
15 END;  
16 /
```

PL/SQL procedure successfully completed

```
SQL> select count(*) from hawa.test@authaa;  
COUNT(*)
```

```
-----  
548
```

```
SQL> select count(*) from hawa.test;  
COUNT(*)
```

```
-----  
548
```

数据矫正完成以后，数据会自动从 missing\_rows 表中删除。

```
SQL> select count(*) from hawa.missing_rows_test;
```

```
COUNT(*)
```

```
-----  
0
```

```
SQL>
```

## 六、 MVR 的概念和构架

Oracle 提供两种不同的复制方法：多主复制（multimaster replication）和物化视图复制

(materialized view replication)。还可以通过两种复制的组合构成混合复制。

本文主要描述物化视图复制，也就是 MVR，由于物化视图复制中的主站点就是多主复制中的站点，因此也会对相应的多主复制中涉及到的内容作相应的说明。

## 一、物化视图的概念和体系结构

Oracle 的物化视图主要用在两个方面：高级复制和数据仓库。在高级复制环境中，物化视图用于复制数据到非主体站点。在数据仓库环境中，物化视图用于对代价昂贵的查询进行缓存。下面讨论物化视图在高级复制环境中的使用。

### 1. 物化视图是什么

物化视图 (materialized view) 是主体对象在某一时间点上的复制品。这个主体对象即可以是主体站点 (master site) 上的一个主表，也可以是物化视图站点 (materialized view site) 上的一个主物化视图。在多主复制中，一个站点上的表被其他主体站点连续不断的更新 (这个也是要分同步和异步的，异步情况下也是类似于 refresh 这样的定时 push)，而物化视图则是从一个主体站点或主物化视图站点批量的进行更新 (也叫做刷新 refresh)。

当物化视图进行快速刷新时 (fast refresh)，Oracle 会检查主表 (master table) 或主物化视图 (master materialized view) 自上次刷新以来的所有改变，并将其应用到物化视图上。因此，如果主体对象自上次刷新以来存在一些改变，则刷新操作则会花费一定的时间把这些改变应用到物化视图上。如果自上次刷新以来没有发生任何变化，则物化视图刷新操作会迅速的完成。

### 2. 为什么使用物化视图

你可以使用物化视图来完成以下目标：

减轻网络负载；

创建一个 Mass Deployment 环境；

数据子集；

Disconnected Computing。

#### (1) 减轻网络负载：

你可以通过物化视图将数据分布到许多站点，所有用户不需要再访问一个数据库服务器，负载被分散到多个数据库服务器上。和多主复制不同的是，你可以根据需要，只复制表中的一部分字段或者表中的一部分数据，从而降低了每次复制的数据量。

多主复制也可以分布网络负载，但与物化视图相比它对网络的要求要严格得多。由于多主复制各个站点间采用的是网状连接，每个站点和其他所有的站点都有通信，而且多主复制一般用于提供实时或接近实时的复制，这会导致很高的网络流量，对于网络状况要求比较严格。物化视图采用高效的批量更新方式，从一个主体站点或一个主物化视图站点获得更新。和多主复制的连续通信不一样，物化视图复制只需要周期性的刷新，从而对网络的要求大大降低。

#### (2) 创建 Mass Deployment 环境：

展开模板 (Deployment templete) 允许你在本地预先建立物化视图环境。你可以利用展开模板快速简便的展开物化视图环境。你可以不用修改展开模板，而是利用参数来建立不同用户的客户化数据集。

#### (3) 数据子集 (Data subsetting)：

物化视图允许你的复制建立在列 (column) 或者行 (row-level) 的基础上，而多主复制需

要复制整张表。通过使用 Data subsetting, 对于每个站点你可以仅复制满足本站点需要的数据。

#### (4) Disconnected Deployment:

物化视图不需要专用网络连接。你可以利用 job 的调度机制完成物化视图的定时自动刷新, 你也可以在需要的时候手工刷新物化视图。而这第二种方法是在笔记本上运行应用程序的一种理想解决方案。

### 3. 物化视图的分类

物化视图分为只读、可更新和可写三类。不能对只读物化视图进行 DML 操作 (INSERT/UPDATE/DELETE), 对于可更新和可写物化视图则可以进行 DML 操作。

注意: 对于只读、可更新和可写物化视图, 定义物化视图的查询语句必须包含主体对象中的所有主键列。

#### (1) 只读物化视图

在建立物化视图时, 省略 FOR UPDATE 语句建立只读物化视图。除了不需要属于一个物化视图组之外, 只读物化视图的许多机制都和可更新物化视图相同。

使用只读物化视图可以消除在主体站点或者主物化视图站点上由物化视图引入的数据冲突, 这个优点的代价是只读物化视图不能进行 dml 操作。

建立只读物化视图的例子如下:

```
CREATE MATERIALIZED VIEW hr.employees AS SELECT * FROM hr.employees@orc1.world;
```

注意: 使用只读物化视图只能消除由物化视图站点引入的冲突, 并不意味着使用只读物化视图就不会有冲突产生, 后面会举例详细说明。

#### (2) 可更新物化视图

在建立物化视图时, 指明 FOR UPDATE 语句建立可更新物化视图。为了可更新物化视图的修改在刷新时可以被“推回”(push pack) 主体对象, 可更新物化视图必须属于一个物化视图组。

由于可更新物化视图允许数据的修改, 因此可以降低主体站点的负载。

下面是建立可更新物化视图的例子:

```
CREATE MATERIALIZED VIEW hr.departments FOR UPDATE AS SELECT * FROM hr.departments@orc1.world;
```

下面的语句建立一个物化视图组:

```
BEGIN
DBMS_REPCAT.CREATE_MVIEW_REPGROUP (
  gname => 'hr_repg',
  master => 'orc1.world',
  propagation_mode => 'ASYNCHRONOUS');
END;
/
```

下面的语句将物化视图 hr.departments 加入到物化视图组 hr\_repg 中, 使得物化视图 hr.departments 可更新。

```
BEGIN
DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
```

```
gname => 'hr_repg',
sname => 'hr',
oname => 'departments',
type => 'SNAPSHOT',
min_communication => TRUE);
END;
/
```

注意:

1. 不要在建立可更新物化视图时使用列的别名，否则，在将物化视图加入到物化视图组的时候会发生错误。
2. 主表或主物化视图列上的默认值不会自动应用到可更新物化视图上。
3. 可更新物化视图不支持 DELETE CASCADE 操作。

### (3) 可写物化视图

可写物化视图指出 FOR UPDATE 语句，但是没有加入到物化视图组。用户可以对可写物化视图执行 DML 操作，但是在执行刷新操作时，修改不会被“推回”，因此所有的修改在刷新后全部丢失。所有允许只读物化视图的情况也同样允许可写物化视图。

由于可写物化视图很少使用，因此以后大部分内容都只涉及只读物化视图和可更新物化视图。

## 4. 物化视图可用性

Oracle 提供几种不同类型的物化视图，以满足各种复制环境的需要。

介绍下列物化视图以及它们使用的环境:

主键物化视图 (Primary Key Materialized Views);

对象物化视图 (Object Materialized Views);

ROWID 物化视图 (ROWID Materialized Views);

复杂物化视图 (Complex Materialized Views)。

当建立物化视图时，不管物化视图属于何种类型，总是给出方案名 (schema)，也就是查询语句中表的所有者名称。例如:

```
CREATE MATERIALIZED VIEW hr.employees
AS SELECT * FROM hr.employees@orc1.world;
```

这个例子中，方案名 hr 被明确指出。

### (1) 主键物化视图

主键物化视图是默认的物化视图。如果主键物化视图是作为物化视图组的一部分建立的，且指定了 FOR UPDATE 语句，那么这个物化视图是可更新的，且这个物化视图组必须和主站点中复制组的同名。另外，可更新物化视图必须和主复制组在不同的数据库中。

当修改发生后，修改的数据以行级为单位被传播，每行数据由主键确定 (而不是 ROWID)。下面是一个创建可更新的主键物化视图的例子:

```
CREATE MATERIALIZED VIEW oe.customers FOR UPDATE AS
SELECT * FROM oe.customers@orc1.world;
```

主键物化视图可以包含一个子查询，因此你可以在建立物化视图时，建立所有数据的一个子集，也就是说，建立物化视图时可以只选取你需要的数据行。子查询是嵌入在主查询中的查询，因此你可以在建立物化视图时有超过一个的 SELECT 语句。子查询可以是简单的 WHERE 语句也可以是复杂的多层 WHERE EXISTS 语句嵌套。如果主站点中的主对象建立了物化视

图日志表 (materialized view log), 那么一些包含特定类型子查询的主键物化视图仍然可以快速 (增量) 刷新。快速刷新利用 materialized view logs 只更新自上次刷新后被修改的记录。

下面的物化视图包含一个 WHERE 语句的子查询:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
SELECT * FROM oe.orders@orc1.world o
WHERE EXISTS
(SELECT * FROM oe.customers@orc1.world c
WHERE o.customer_id = c.customer_id AND c.credit_limit > 10000);
```

这种类型的物化视图又叫做子查询物化视图。

## (2) 对象物化视图

如果物化视图是基于对象表, 并且在建立是指定了 OF TYPE 语句, 那么这个物化视图叫做对象物化视图。对象物化视图的结构和对象表相同——对象物化视图由行对象(row objects) 组成, 每一个行对象由一个对象标识列 OID (object identifier) 标识。

## (3) ROWID 物化视图

为了后向兼容性, Oracle 除了默认的主键物化视图外, 还支持 ROWID 物化视图。ROWID 物化视图基于主对象中行记录的物理标识 ROWID (physical row identifiers)。ROWID 物化视图只被用在基于 Oracle7 版本的主对象的物化视图, 它不能被用于建立基于 Oracle8 或更高版本主站点的物化视图。

下面是一个建立 ROWID 物化视图的例子:

```
CREATE MATERIALIZED VIEW oe.orders REFRESH WITH ROWID AS
SELECT * FROM oe.orders@orc1.world;
```

## (4) 复杂物化视图

物化视图的定义必须满足某种约束, 才能执行快速刷新。如果你需要的物化视图的定义查询语句更为一般化, 不能满足限制条件, 那么这个物化视图是复杂的, 并且不能执行快速刷新。一般来说, 如果一个物化视图的定义查询包含下列语句, 则被认为是复杂物化视图:

CONNECT BY 语句;

例如:

```
CREATE MATERIALIZED VIEW hr.emp_hierarchy AS
SELECT LPAD(' ', 4*(LEVEL-1))||email USERNAME
FROM hr.employees@orc1.world START WITH manager_id IS NULL
CONNECT BY PRIOR employee_id = manager_id;
```

INTERSECT, MINUS 或 UNION ALL 操作;

例如:

```
CREATE MATERIALIZED VIEW hr.mview_employees AS
SELECT employees.employee_id, employees.email
FROM hr.employees@orc1.world
UNION ALL
SELECT new_employees.employee_id, new_employees.email
FROM hr.new_employees@orc1.world;
```

在某些情况下的 DISTINCT 和 UNIQUE 关键字；

包含 DISTINCT 和 UNIQUE 关键字的并不都是复杂物化视图，简单物化视图（simple materialized view）中也可以包含这两个关键字。下面的例子是建立一个包含 DISTINCT 关键字的复杂物化视图：

```
CREATE MATERIALIZED VIEW hr.employee_depts AS
SELECT DISTINCT department_id FROM hr.employees@orc1.world
ORDER BY department_id;
```

聚集操作；

如下例：

```
CREATE MATERIALIZED VIEW hr.average_sal AS
SELECT AVG(salary) "Average" FROM hr.employees@orc1.world;
```

连接不在子查询中的对象；

例如：

```
CREATE MATERIALIZED VIEW hr.emp_join_dep AS
SELECT last_name
FROM hr.employees@orc1.world e, hr.departments@orc1.world d
WHERE e.department_id = d.department_id;
```

某种情况下的 UNION 操作。

例如：

```
CREATE MATERIALIZED VIEW oe.orders AS
SELECT order_total
FROM oe.orders@orc1.world o
WHERE EXISTS
(SELECT cust_first_name, cust_last_name
FROM oe.customers@orc1.world c
WHERE o.customer_id = c.customer_id
AND c.credit_limit > 50)
UNION
SELECT customer_id
FROM oe.orders@orc1.world o
WHERE EXISTS
(SELECT cust_first_name, cust_last_name
FROM oe.customers@orc1.world c
WHERE o.customer_id = c.customer_id
AND c.account_mgr_id = 30);
```

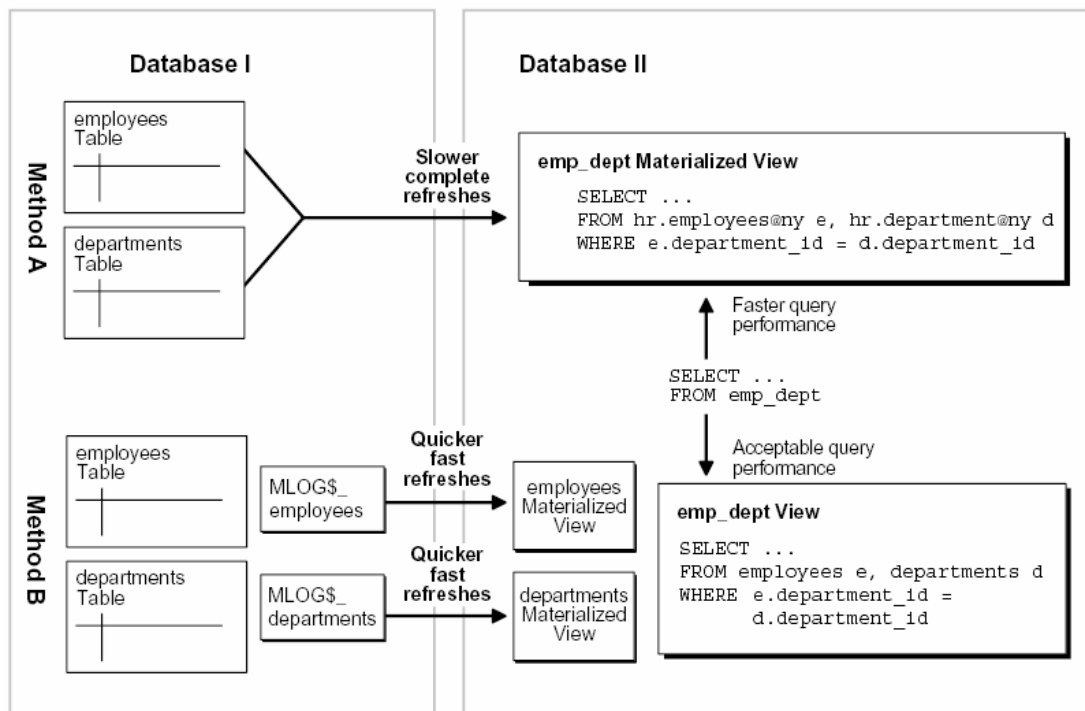
以及其他任何不满足特定约束条件的子查询。具体条件见“物化视图中的数据子集”（Data Subsetting with Materialized Views）的物化视图子查询约束（Restrictions for Materialized Views with Subqueries）

注意：如果可能，尽量避免使用复杂物化视图，因为复杂物化视图不能快速刷新，将会降低网络性能。

对比简单物化视图和复杂物化视图：

为了某种应用，你可能需要考虑使用一个复杂物化视图。你有两种方法可以选择，它们各有利弊，具体如下：

Figure 3-2 Comparison of Simple and Complex Materialized Views



图一：复杂物化视图和简单物化视图对比

复杂物化视图：上图中方法 A 展示了一个复杂物化视图。这个复杂物化视图在数据库 II 中展示出高效的查询性能，因为连接操作在物化视图刷新时已经完成。然而，由于是复杂物化视图，必须执行完全刷新，这将极为可能比执行快速刷新要慢得多。

简单物化视图通过视图连接：上图中方法 B 在 DATABASE II 展示了两个简单物化视图，它们通过一个视图执行连接操作。通过视图查询不可能有方法 A 中查询复杂物化视图那样的性能。然而，简单物化视图可以更加有效的使用快速刷新和物化视图日志（materialized view logs）。

总的来说：

如果你很少刷新，且需要比较高的查询性能，则使用方法 A。（complex materialized view）

如果你经常刷新，且可以牺牲查询性能，则使用方法 B。（simple materialized view）

## 5. 物化视图操作所需权限

三种不同类型的用户对物化视图执行操作：

建立者（creator）：建立物化视图的用户。

刷新者（refresher）：刷新物化视图的用户。

所有者（owner）：拥有物化视图的用户。物化视图存在于所有者的方案（schema）中。

一个用户对物化视图可以执行所有的操作。然而，在一些复制环境中，不同的用户对物化视图执行不同的操作。执行这些操作需要的权限取决于操作由同一个用户执行还是由不同的用户执行。下面详细解释所需权限。

注意：下文没有包括用重写查询（query rewrite）选项来创建物化视图时所需要的权限。

#### （1）创建者是所有者

如果一个物化视图的创建者同时也是一个物化视图的所有者，那么这个用户可以通过明确授权或通过角色拥有下列权限来创建一个物化视图。

**CREATE MATERIALIZED VIEW 或者 CREATE ANY MATERIALIZED VIEW 权限；**

**CREATE TABLE 或者 CREATE ANY TABLE 权限；**

如果数据库兼容性在 8.1.0 以下，需要 **CREATE VIEW 或者 CREATE ANY VIEW 权限；**

对主站点上对象和物化视图日志的 **SELECT 权限或者 SELECT ANY TABLE 系统权限**。如果主站点不是本地数据库，则 **SELECT 权限** 必须授权给一个主站点用户，这个用户就是物化视图站点通过数据库链（database link）连接到主站点的用户（数据库链创建语句中 **CONNECT TO** 关键字后面跟的用户）。

#### （2）创建者不是所有者

如果物化视图的创建者不是所有者，必须授予创建者和所有者某种权限才能创建物化视图。创建者的权限可以直接授权或通过角色授权，但是所有者的权限必须通过明确授权获得。也就是说，所有者的权限不能通过角色获得。

创建者：

**CREATE ANY MATERIALIZED VIEW 系统权限**

所有者：

**CREATE TABLE 或者 CREATE ANY TABLE 权限；**

如果数据库兼容性在 8.1.0 以下，需要 **CREATE VIEW 或者 CREATE ANY VIEW 权限；**

对主站点上对象和物化视图日志的 **SELECT 权限或者 SELECT ANY TABLE 系统权限**。如果主站点不是本地数据库，则 **SELECT 权限** 必须授权给一个主站点用户，这个用户就是物化视图站点通过数据库链（database link）连到主站点的用户（数据库链创建语句中 **CONNECT TO** 关键字后面跟的用户）。

#### （3）刷新者是所有者

如果一个物化视图的刷新者同时也是物化视图的拥有者，这个用户需要主站点上对象和物化视图日志的 **SELECT 权限或者 SELECT ANY TABLE 系统权限**。如果主站点不是本地数据库，则 **SELECT 权限** 必须授权给一个主站点用户，这个用户就是物化视图站点通过数据库链（database link）连到主站点的用户（数据库链创建语句中 **CONNECT TO** 关键字后面跟的用户）。权限可以通过直接授权或通过角色授权。

#### （4）刷新者不是所有者

如果物化视图的刷新者不是所有者，必须授予刷新者和所有者某种权限。这些权限可以直接授权或通过角色授权。

刷新者：

**ALTER ANY MATERIALIZED VIEW 系统权限。**

所有者：

主站点上对象和物化视图日志的 **SELECT 权限或者 SELECT ANY TABLE 系统权限**。如果主站点不是本地数据库，则 **SELECT 权限** 必须授权给一个主站点用户，这个用户就是物化视图站点通过数据库链（database link）连到主站点的用户（数据库链创建语句中 **CONNECT TO** 关键字后面跟的用户）。

## 6. 物化视图中的数据子集

在某些情况下，你可能希望你的物化视图反映主表或者主物化视图中数据的子集。通过使用 **WHERE** 语句，行子集允许你包含主表或者主物化视图中你想要的行记录。列子集允许你从主表或者主物化视图中只包含你所需要的列。在创建物化视图时，通过在 **SELECT** 语句中明确指出所要选取的列来实现列子集。如果你使用展开模板来创建你的物化视图，那么你可以在可更新物化视图上定义列子集。

### (1) 使用数据子集的一些原因

**降低网络流量：**在一个使用列子集的物化视图中，只有满足物化视图定义中 **WHERE** 条件语句的修改才会传播到物化视图站点，因此减少了事务传输数量，降低了网络流量。

**保护敏感数据：**用户只能查看满足物化视图查询定义的数据。

**减少资源需要：**如果物化视图建立在笔记本上，则硬盘与服务器上的硬盘相比要小得多。数据子集可以显著的减少存储空间。

**提高刷新性能：**由于较少的数据传播到物化视图站点，刷新执行的更加迅速。这一点对于那些需要通过拨号连接来刷新物化视图的用户十分重要。

例如：下面语句基于 `oe.orders@orc1.world` 主表创建一个物化视图，而且只包括 `sales_rep_id` 等于 173 的记录。

```
CREATE MATERIALIZED VIEW oe.orders REFRESH FAST AS
SELECT * FROM oe.orders@orc1.world
WHERE sales_rep_id = 173;
```

主表中 `sales_rep_id` 不等于 173 的记录被排除出物化视图。

### (2) 带子查询的物化视图

上面的例子是针对单个表的。如果创建基于多个表的物化视图，则定义和维护这些物化视图相对来说困难得多。主要包括多对一子查询、一对多子查询、多对多子查询以及包含 **UNION** 操作的子查询几种。这些物化视图比较复杂，而且在实际复制中不经常使用，因此这里不再详细描述。如果对这部分有兴趣，请参阅 *Oracle9i Advanced Replication 3-18 Materialized Views with Subqueries* 部分。下面描述一下建立快速刷新子查询物化视图的条件。

**物化视图子查询约束**

带子查询的物化视图为了能达到快速刷新的能力，必须满足许多约束条件，具体如下：

必须是主键物化视图；

物化视图日志必须包括某些在子查询中出现的列；

如果子查询是多对多或一对多查询，连接列中非主键的部分必须包括在物化视图日志中，多对一子查询没有这个约束；

子查询必须是肯定条件，比如，你可以使用 **EXISTS**，但是不能使用 **NOT EXISTS**；

子查询必须使用 **EXISTS** 连接到嵌套层（nested level），不能使用 **IN**；

每张表只允许一个 **EXISTS** 表达式；

连接表达式必须采用精确匹配或等于连接；

在子查询中，每张表只能被连接一次；

在嵌套层（nested level）中，每张表必须有主键存在；

嵌套层（nested level）只能参考比它高的层中的表；

子查询可以包含 **AND** 操作，但是每个 **OR** 操作只能连接“能确定一条记录”的列，子查询中多个 **OR** 运算可以通过 **AND** 连接；

子查询中的所有表必须在同一个主体站点或主物化视图站点中。

## 7. 决定物化视图的快速刷新能力

为了检测所创建的带子查询的物化视图是否满足上面提到的建立快速刷新物化视图的种种约束，在创建时，如果违反任何约束条件，则 Oracle 会返回错误提示。如果在建立物化视图时指明强制刷新（force refresh），则不会收到任何错误信息。因为在强制刷新时，如果不能执行快速刷新的话，Oracle 会自动执行完全刷新（complete refresh）。

你也可以通过 DBMS\_MVIEW 包中的 EXPLAIN\_MVIEW 过程来检测已存在的物化视图甚至是还没有建立的物化视图的一些信息，具体信息如下：

物化视图的各种能力；

对于这个物化视图来说每种能力是否可能；

如果不可能，给出导致这种能力不可能的原因；

这些信息可以存储在 varray 中，也可以存储在 MV\_CAPABILITIES\_TABLE 表中。如果你希望把信息存储到表中，那么你必须在执行 EXPLAIN\_MVIEW 存储过程以前，执行 ORACLE\_HOME/rdbms/admin 目录下的 utlxmv.sql 脚本。

例如：检查 oe.orders 物化视图的能力输入：

```
EXECUTE DBMS_MVIEW.EXPLAIN_MVIEW ('oe.orders');
```

或者如果物化视图不存在，你可以输入你希望建立的物化视图的查询语句：

```
BEGIN
```

```
DBMS_MVIEW.EXPLAIN_MVIEW ('SELECT * FROM oe.orders@orc1.world o
WHERE EXISTS (SELECT * FROM oe.customers@orc1.world c
WHERE o.customer_id = c.customer_id AND c.credit_limit > 500)');
```

```
END;
```

```
/
```

查询 MV\_CAPABILITIES\_TABLE 表得到结果，如下例：

```
SQL> set linesize 110
```

```
SQL> col RELATED_TEXT format a60
```

```
SQL> select capability_name, possible, related_text from mv_capabilities_table;
```

CAPABILITY_NAME	P	RELATED_TEXT
PCT	N	
REFRESH_COMPLETE	Y	
REFRESH_FAST	N	
REWRITE	N	
PCT_TABLE	N	此上下文不支持对象数据类型
REFRESH_FAST_AFTER_INSERT	N	OE.CUSTOMERS
REFRESH_FAST_AFTER_INSERT	N	OE.ORDERS
REFRESH_FAST_AFTER_ONETAB_DML	N	
REFRESH_FAST_AFTER_ANY_DML	N	
REFRESH_FAST_PCT	N	
REWRITE_FULL_TEXT_MATCH	N	OE.ORDERS
REWRITE_FULL_TEXT_MATCH	N	OE.CUSTOMERS
REWRITE_FULL_TEXT_MATCH	N	此上下文不支持对象数据类型
REWRITE_PARTIAL_TEXT_MATCH	N	
REWRITE_GENERAL	N	
REWRITE_PCT	N	

已选择 16 行。

## 8. 多级物化视图 (Multitier Materialized Views)

物化视图的创建可以基于表，也可以基于其他物化视图，这种物化视图叫做多级物化视图 (multitier materialized views)。这种基于其他物化视图的物化视图可以是只读的 (read only) 也可以是可更新的 (updatable)。

当使用多级物化视图时，基于主体表的物化视图叫做第一层物化视图 (level 1 materialized view)。基于第一层物化视图的物化视图叫做第二层物化视图 (level 2 materialized view)。下一层是第三层，依此类推。

作为其他物化视图的主对象的物化视图叫做主物化视图 (master materialized view)。处在任何一层的物化视图都可以成为主物化视图，而且可以存在多个其他的物化视图基于同一个主物化视图。

主物化视图和主站点中的主表起相同的作用。也就是说把第二层物化视图上的改变“推到”第一层物化视图上的操作和把第一次物化视图上的改变“推到”主表上的操作是完全一样的。在主物化视图站点必须注册一个接收者 (receiver)。在多层物化视图站点中接收者负责接受和应用来自主物化视图站点传播者的延迟事务。

多层物化视图 (Multitier materialized views) 在设计复制环境时提供了很高的灵活性。一些物化视图站点不需要复制主表中所有的数据，实际上，这些站点可能根本没有足够的空间存储这些数据。另外，只复制较少的数据意味着在网络上活动的数据也较少。

使用多层物化视图的限制条件：

主物化视图和基于主物化视图的物化视图都必须满足下列条件：

- 1) 必须是主键物化视图；
- 2) 所在数据库兼容性等于或者高于 9.0.1。

主物化视图的一些限制。下列类型的物化视图不能作为可更新物化视图的主物化视图。

- 1) ROWID 物化视图；
- 2) 复杂物化视图；
- 3) 只读物化视图。

不过这三种物化视图可以成为只读物化视图的主物化视图。

基于物化视图的可更新物化视图的额外限制：

- 1) 所属的物化视图组必须和主物化视图站点上的物化视图组同名；
- 2) 必须和主物化视图不在同一个数据库上；
- 3) 必须基于可更新物化视图，不能基于只读物化视图；
- 4) 主站点上的主物化视图组必须存在于 PUBLIC 方案中。

## 9. 包含用户定义类型的物化视图

Oracle 的复制支持用户自定义类型数据。Oracle 不但支持行对象 (row object) 和列对象 (column object) 而且还支持 collections 的复制。Collection 包括基于用户自定义类型的数组 (VARRAY) 和嵌套表 (nested table)。

使用用户自定义类型需要注意几点：

主站点和物化视图站点的数据库兼容性等于或高于 9.0.1；

如果主对象包含用户自定义类型，则不能创建 refresh-on-commit 物化视图；

高级复制不支持对象的继承。

物化视图复制中对用户自定义类型的要求和限制于多主环境中的限制比较类似，这里不再详

细描述, 如果对这部分内容感兴趣, 请参考 Oracle9i Advanced Replication 3-36 Materialized Views with User-Defined Types。

**10. 主站点的物化视图注册**

**11. 主站点和主物化视图站点机制**

**12. 物化视图站点机制**

**13. 组织机制**

**14. 刷新处理**

**三. 展开模板 (Deployment Templates) 的概念和构架**

**四. 多主复制和物化视图复制区别**

## 附录一。多主体复制站点的配置步骤

以下操作如果不是明确指出，均在 master 数据库中运行。

1. 检查安装好的数据库是否支持高级复制：

```
SQL> select value from v$option where parameter='Advanced replication';
```

```
VALUE
```

```
-----  
TRUE
```

确保返回的结果是 TRUE，如果是 FALSE 则表示需要重新安装 oracle 的高级复制部件。

2. 确保数据库的初始化参数中 global\_name=true，同时因为高级复制依靠于 JOB 来实现，所以必须保证

job\_queue\_processes 参数大于 0，我们可以设置为 10。

确保 init.ora 中包含一下初始化参数定义：

```
global_names = true
```

```
open_links = 4 （备注：一个 process 需要 4 个 link，如果我们创建了多个 dblink，并且同时运行，那么可以把此参数设大，比如以下环境中我们应该设置为 open_links = 8）
```

```
job_queue_processes = 10
```

3. 用 sysdba 权限分别登录 master 和 snap 数据库，检查双方的 global\_name，必须保证两边的域名相同才可以建立正确的 dblink。

```
select * from global_name;
```

假设显示结果是 master.com，那么表示该数据库的域名是 com。那么我们可以设置 snap 库的 global\_name 是 snap.com。

使用以下 SQL 设置 global\_name：

```
alter database rename global_name to master.com;
```

4. 创建一个 PUBLIC DBLINK 连接到 snap（此步骤可以省略）

```
CREATE PUBLIC DATABASE LINK "snap.com" USING '(DESCRIPTION = (ADDRESS_LIST = (ADDRESS= (PROTOCOL = TCP)(Host = 10.1.6.124)(Port = 1521)))(CONNECT_DATA = (SID = test1)(SERVER = DEDICATED)))';
```

运行以下 SQL 检查 dblink 创建是否成功，如果结果返回 snap 的 global\_name 则表示成功

```
SQL> select * from global_name@snap.com;
```

```
GLOBAL_NAME
```

-----  
SNAP.COM

5. 创建 repadmin 用户，用于管理高级复制

```
create user repadmin identified by repadmin default tablespace users temporary
tablespace temp;
execute dbms_defer_sys.register_propagator('repadmin');
grant execute any procedure to repadmin;
execute dbms_repcat_admin.grant_admin_any_repgroup('repadmin');
execute dbms_repcat_admin.grant_admin_any_schema(username => "REPADMIN");
grant comment any table to repadmin;
grant lock any table to repadmin;
grant select any dictionary to repadmin;
```

6. 登录 snap 数据库，重复上面的操作，创建 public dblink 以及 repadmin 用户

7. 用 repadmin 用户登录 master，创建私有数据库连接

```
create database link "snap.com" connect to repadmin identified by repadmin;
```

如果第 4 步省略了，没有创建公有数据库连接，则需要如下创建，在创建含有 qualifier 的多个数据库连接时也只能使用下面的方法：

```
create database link "snap.com@perday" connect to repadmin identified by repadmin
USING '(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(Host =
10.1.6.124)(Port = 1521)))(CONNECT_DATA = (SID = test1)(SERVER = DEDICATED)))';
create database link "snap.com@perhour" connect to repadmin identified by repadmin
USING '(DESCRIPTION = (ADDRESS_LIST = (ADDRESS = (PROTOCOL = TCP)(Host =
10.1.6.124)(Port = 1521)))(CONNECT_DATA = (SID = test1)(SERVER = DEDICATED)))';
```

检查是否创建成功

```
SQL> select * from global_name@snap.com@perday;
```

GLOBAL\_NAME  
-----

SNAP.COM

8. 创建主体复制组，添加复制对象，操作的数据库将称为主体定义站点

创建每天复制一次的组

```
execute dbms_repcat.create_master_repgroup(gname => 'rep_gp_day',group_comment
=> 'replcation perday',qualifier => '@PERDAY');
```

创建每小时复制一次的组

```
execute dbms_repcat.create_master_repgroup(gname => 'rep_gp_hour',group_comment
=> 'replcation perhour',qualifier => '@PERHOUR');
```

备注：以下操作只以 rep\_gp\_day 复制组为例，对于 rep\_gp\_hour 复制组则应该作相应更改

再执行下面的操作。

检查执行结果

```
select * from dba_repsites;
```

--用 spool 生成批量执行的 SQL

```
set feedback off;
set pagesize 0;
set heading off;
set verify off;
set linesize 1000;
set trimspool on;
spool filename.sql;
select 'execute dbms_repcat.create_master_repobject(sname=>"test_user",oname=>"
|| table_name || ",type=>"table",use_existing_object=>true,gname=>"rep_gp_day");'
CREATE_SQL from tabs;
select 'dbms_repcat.generate_replication_support("test_user"," || table_name ||
","table");' GEN_SQL from tabs;
spool off;
set feedback on;
set pagesize 9999;
set heading on;
set verify on;
```

检查复制组状态

```
select gname, master, status from dba_repgroup;
```

如果该复制组已经处于 normal 状态，那么在添加复制对象之前必须先停顿复制组，既将同步组的状态由正常(normal)改为停顿(quietced )

```
execute dbms_repcat.suspend_master_activity (gname => 'rep_gp_day');
```

运行上面生成的 spool 文件，批量执行创建复制对象和生成复制支持

如果是单独创建复制对象，则是手工执行下面的 SQL

```
execute dbms_repcat.create_master_repobject(sname=>'test_user',oname=>'account',
type=>'table',use_existing_object=>true,gname=>'rep_gp_day',copy_rows => false);
execute dbms_repcat.generate_replication_support('test_user','account','table');
```

备注：如果所有的主体站点都是在 Oracle815 以上的版本，那么设置 generate\_replication\_support 中的 generate\_80\_compatible 参数为 false，默认是 true。

检查执行结果

```
select * from dba_repobject;
```

9. 添加主体库，这一步操作必须要求 dblink 工作正常

```
execute dbms_repcat.add_master_database(gname=>'rep_gp_day',
```

```
master=>'snap.com@perday',      use_existing_objects=>true,      copy_rows=>false,
propagation_mode => 'asynchronous');
select * from user_jobs;
```

```
execute dbms_repcat.resume_master_activity('rep_gp_day',false);
select gname, master, status from dba_repgroup;
```

如果上述的检查结果显示 status 不是 normal 的，那么运行：

```
execute dbms_repcat.resume_master_activity('rep_gp_day',true);
```

10. 添加 PUSH 的任务（执行间隔为 1 天 1 次），如果是一小时一次，则是 1/24，如果是一分钟一次则是 1/1440

```
begin
dbms_defer_sys.schedule_push (
destination => 'snap.com@perday',
interval => 'sysdate + 1',
next_date => sysdate,
parallelism => 1,
delay_seconds => 50);
end;
/
```

添加 PURGE 的任务（执行间隔为 1 分钟 1 次）

```
begin
dbms_defer_sys.schedule_purge (
next_date => sysdate,
interval => 'sysdate + 1/1440',
delay_seconds => 0,
rollback_segment => "");
end;
/
```

11. 用 repadmin 用户登录 snap，创建私有数据库连接

```
create database link "master.com" connect to repadmin identified by repadmin;
create database link "master.com@perday" connect to repadmin identified by repadmin
USING '(DESCRIPTION = (ADDRESS_LIST = (ADDRESS= (PROTOCOL = TCP)(Host =
10.1.6.120)(Port = 1521)))(CONNECT_DATA = (SID = test1)(SERVER = DEDICATED)))';
create database link "master.com@perhour" connect to repadmin identified by repadmin
USING '(DESCRIPTION = (ADDRESS_LIST = (ADDRESS= (PROTOCOL = TCP)(Host =
10.1.6.120)(Port = 1521)))(CONNECT_DATA = (SID = test1)(SERVER = DEDICATED)))';
```

检查是否创建成功

```
SQL> select * from global_name@master.com@perday;
```

GLOBAL\_NAME

-----  
MASTER.COM

12. 添加 PUSH 和 PURGE 的任务（执行间隔为 1 天 1 次），如果是一小时一次，则是 1/24，如果是一分钟一次则是 1/1440

```
begin
dbms_defer_sys.schedule_push (
destination => 'master.com@perday',
interval => 'sysdate + 1',
next_date => sysdate,
parallelism => 1,
delay_seconds => 50);
end;
/
```

添加 PURGE 的任务（执行间隔为 1 分钟 1 次）

```
begin
dbms_defer_sys.schedule_purge (
next_date => sysdate,
interval => 'sysdate + 1/1440',
delay_seconds => 0,
rollback_segment => '');
end;
/
```

至此，高级复制环境设置完毕。

## 附录二。物化视图复制站点的配置步骤

主站点: rep.yangtingkun

物化视图站点: yangtk.yangtingkun

主机名: yangtingkun

复制用户: yangtk

### 1. 检查初始化参数

复制对数据库的初始化参数限制不多，主要注意两点。

global\_names 为 TRUE 以及 job\_queue\_process 大等 0。

分别在主站点和物化视图站点执行下面两条 sqlplus 命令，检查数据库初始化参数是否符合要求。

```
show parameter global_names
```

```
show parameter job
```

如果初始化参数设置的不满足要求，可以通过下列语句动态修改。

```
alter system set global_names = true;
alter system set job_queue_processes = 20;
```

## 2. 检查全局数据库名称

两个数据库的 `db_domain` 名称应该相同，只有 `db_name` 不同。

通过下列语句检查主站点和物化视图站点的全局数据库名

```
select * from global_name;
```

如果全局数据库名设置不符合规范，可以通过如下语句动态修改。

```
alter database rename global_name to rep.yangtingkun;
alter database rename global_name to yangtk.yangtingkun;
```

## 3. 修改 tnsnames.ora 文件，主站点和物化视图站点的参数文件中都添加下列内容

REP =

```
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = yangtingkun)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = rep)
  )
)
```

YANGTK =

```
(DESCRIPTION =
  (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST = yangtingkun)(PORT = 1521))
  )
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = yangtk)
  )
)
```

## 4. 建立主体站点

--以 system 用户连接到主站点

```
CONN system@rep
```

--建立复制管理用户 repadmin 并授权

```
CREATE USER repadmin IDENTIFIED BY repadmin;
```

```
BEGIN
```

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (username => 'repadmin');
```

```
END;
```

```

/
GRANT COMMENT ANY TABLE TO repadmin;
GRANT LOCK ANY TABLE TO repadmin;
GRANT SELECT ANY DICTIONARY TO repadmin;

--注册传播用户并授权，这里使用了管理用户 repadmin，也可以分别建立用户
BEGIN
DBMS_DEFER_SYS.REGISTER_PROPAGATOR (username => 'repadmin');
END;
/

--注册接收用户，这里使用了管理用户 repadmin
BEGIN
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
username => 'repadmin',
privilege_type => 'receiver',
list_of_gnames => NULL);
END;
/

--建立物化视图站点复制管理员的代理用户，出于简单考虑，这里也使用 repadmin 用户
BEGIN
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (
username => 'repadmin',
privilege_type => 'proxy_snapadmin',
list_of_gnames => NULL);
END;
/

--设置代理刷新用户，并授权，这里仍然使用 repadmin 用户
--对于 repadmin 而言，不需要 create session 权限
--但是这里如果新建用户的话，create session 权限则是必须的
GRANT CREATE SESSION TO repadmin;
GRANT SELECT ANY TABLE TO repadmin;

--设置清除延迟序列的 job
--以复制管理员身份登陆到主站点
CONNECT repadmin/repadmin@rep
BEGIN
DBMS_DEFER_SYS.SCHEDULE_PURGE (
next_date => SYSDATE,
interval => 'SYSDATE + 1/24',
delay_seconds => 0);
END;

```

/

commit;

--多主站点的设置还需要多个站点间建立数据库链并建立调度机制  
--但是对于物化视图复制的主体站点，则这些设置是不需要的

## 5. 设置物化视图站点

--以 system 用户连接到物化视图站点

CONN system@yangtk

--建立物化视图管理员，并授权

```
CREATE USER mvadmin IDENTIFIED BY mvadmin;
```

```
BEGIN
```

```
DBMS_REPCAT_ADMIN.GRANT_ADMIN_ANY_SCHEMA (  
username => 'mvadmin');
```

```
END;
```

/

```
GRANT COMMENT ANY TABLE TO mvadmin;
```

```
GRANT LOCK ANY TABLE TO mvadmin;
```

```
GRANT SELECT ANY DICTIONARY TO mvadmin;
```

--建立传播者，并授权，这里使用 mvadmin 用户，也可以建立单独的用户

```
BEGIN
```

```
DBMS_DEFER_SYS.REGISTER_PROPAGATOR (username => 'mvadmin');
```

```
END;
```

/

--建立刷新者，并授权，这里使用 mvadmin 用户刷新物化视图

--对于 mvadmin 而言，不需要 create session 权限

--但是这里如果新建用户的话，create session 权限则是必须的

```
GRANT CREATE SESSION TO mvadmin;
```

```
GRANT ALTER ANY MATERIALIZED VIEW TO mvadmin;
```

--注册接受者

```
BEGIN
```

```
DBMS_REPCAT_ADMIN.REGISTER_USER_REPGROUP (  
username => 'mvadmin',
```

```
privilege_type => 'receiver',
```

```
list_of_gnames => NULL);
```

```
END;
```

/

--建立 PUBLIC 数据库链

```

CREATE PUBLIC DATABASE LINK rep.yangtingkun USING 'rep';

--建立到主站点上代理物化视图管理员的数据库链
--以物化视图管理员身份连接到物化视图站点
CONNECT mvadmin/mvadmin@yangtk
CREATE DATABASE LINK rep.yangtingkun CONNECT TO repadmin IDENTIFIED BY
repadmin;

--建立到主站点上复制管理员的数据库链
--以传播者身份登陆物化视图站点
--在本例中，这个数据库链与上面的数据库链相同，故省略。

--设置清除延迟序列的 job
--如果物化视图站点只包括只读物化视图，这一步可以省略
BEGIN
DBMS_DEFER_SYS.SCHEDULE_PURGE (
next_date => SYSDATE,
interval => 'SYSDATE + 1/24',
delay_seconds => 0,
rollback_segment => '');
END;
/

--设置将修改推入到主站点的 job
--如果物化视图站点只包括只读物化视图，这一步可以省略
BEGIN
DBMS_DEFER_SYS.SCHEDULE_PUSH (
destination => 'rep.yangtingkun',
interval => 'SYSDATE + 1/24',
next_date => SYSDATE,
stop_on_error => FALSE,
delay_seconds => 0,
parallelism => 0);
END;
/

--如果需要此物化视图站点作为主物化视图站点
--则还需要建立物化视图站点的代理物化视图管理用户以及代理刷新用户
--本例中从略

commit;

```

## 6. 建立主体组

```

--以复制管理员身份登陆复制站点

```

```
CONNECT repadmin/repadmin@rep
```

```
--建立名为 rep_test 的复制组
```

```
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPGROUP (  
gname => 'rep_test');  
END;  
/
```

```
--将复制对象增加到复制组中
```

```
--主键所用的索引自动复制，其他索引需要明确添加到复制组中
```

```
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (  
gname => 'rep_test',  
type => 'TABLE',  
oname => 'test_rep',  
sname => 'yangtk',  
use_existing_object => TRUE,  
copy_rows => FALSE);  
END;  
/
```

```
BEGIN  
DBMS_REPCAT.CREATE_MASTER_REPOBJECT (  
gname => 'rep_test',  
type => 'INDEX',  
oname => 'ind_test_rep_name',  
sname => 'yangtk',  
use_existing_object => TRUE,  
copy_rows => FALSE);  
END;  
/
```

```
--生成复制支持
```

```
BEGIN  
DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (  
sname => 'yangtk',  
oname => 'test_rep',  
type => 'TABLE',  
min_communication => TRUE);  
END;  
/
```

```
--开始复制
```

```

BEGIN
DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
gname => 'rep_test');
END;
/

commit;
7. 建立物化视图
--以复制用户连接到主站点
CONNECT yangtk@rep

--建立物化视图日志表，FAST 刷新方式必须要求建立物化视图日志，COMPLETE 则不需要
CREATE MATERIALIZED VIEW LOG ON yangtk.test_rep;

--如果被复制用户不存在则建立，并授予相应权限
--本例中，用户已存在，此步骤省略
/*
CONNECT system@yangtk

CREATE USER yangtk IDENTIFIED BY yangtk;
ALTER USER yangtk DEFAULT TABLESPACE users QUOTA UNLIMITED ON users;
ALTER USER yangtk TEMPORARY TABLESPACE temp;
GRANT
CREATE SESSION,
CREATE TABLE,
CREATE PROCEDURE,
CREATE SEQUENCE,
CREATE TRIGGER,
CREATE VIEW,
CREATE SYNONYM,
ALTER SESSION,
CREATE MATERIALIZED VIEW,
ALTER ANY MATERIALIZED VIEW,
CREATE DATABASE LINK
TO yangtk;
*/

--建立复制用户到主站点代理刷新者的数据库链
CONNECT yangtk@yangtk
CREATE DATABASE LINK rep.yangtingkun CONNECT TO repadmin IDENTIFIED BY
repadmin;

--建立物化视图组
--以物化视图管理员身份登陆物化视图站点

```

```
CONNECT mvadmin/mvadmin@yangtk
```

```
--物化视图组必须和复制站点上的复制组名称相同
```

```
BEGIN  
DBMS_REPCAT.CREATE_MVIEW_REPGROUP (  
  gname => 'rep_test',  
  master => 'rep.yangtingkun',  
  propagation_mode => 'ASYNCHRONOUS');  
END;  
/
```

```
--创建刷新组
```

```
--对于只包含只读物化视图的站点，不需要此步骤
```

```
BEGIN  
DBMS_REFRESH.MAKE (  
  name => 'mvadmin.rep_refresh',  
  list => '',  
  next_date => SYSDATE,  
  interval => 'SYSDATE + 1/24',  
  implicit_destroy => FALSE,  
  rollback_seg => '',  
  push_deferred_rpc => TRUE,  
  refresh_after_errors => FALSE);  
END;  
/
```

```
--创建物化视图
```

```
--对于只读物化视图，省略 FOR UPDATE 语句
```

```
CREATE MATERIALIZED VIEW yangtk.test_rep  
REFRESH FAST WITH PRIMARY KEY FOR UPDATE  
AS SELECT * FROM yangtk.test_rep@rep.yangtingkun;
```

```
--将物化视图添加到物化视图组
```

```
--对于只读物化视图，此步骤可以省略
```

```
BEGIN  
DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (  
  gname => 'rep_test',  
  sname => 'yangtk',  
  oname => 'test_rep',  
  type => 'SNAPSHOT',  
  min_communication => TRUE);  
END;  
/
```

```
BEGIN
DBMS_REPCAT.CREATE_MVIEW_REPOBJECT (
gname => 'rep_test',
sname => 'yangtk',
oname => 'ind_test_rep_name',
type => 'INDEX',
min_communication => TRUE);
END;
/
```

--将物化视图添加到刷新组

```
BEGIN
DBMS_REFRESH.ADD (
name => 'mvadmin.rep_refresh',
list => 'yangtk.test_rep',
lax => TRUE);
END;
/
```

```
commit;
```

#### 8. 主对象上建立测试对象脚本

```
create table test_rep (id number not null, name varchar2(100));
alter table test_rep add constraint pk_test_rep primary key (id);
create index ind_test_rep_name on test_rep (name);
insert into test_rep values (1, 'ytk');
insert into test_rep values (2, 'zhly');
commit;
```

## 附录三。一些高级复制相关的包使用方法

### 1. 删除复制对象

```
exec DBMS_REPCAT.DROP_MASTER_REPOBJECT(sname => 'test_user',oname =>
't_emp',type => 'table');
```

### 2. 删除复制组

```
exec dbms_repcat.drop_master_repgroup(gname=>'rep_gp',all_sites => true);
```

### 3. 创建新的表或者其它 object 并加入复制支持

```
BEGIN
  DBMS_REPCAT.SUSPEND_MASTER_ACTIVITY (
    gname => 'scott_mg');
END;
/
```

```
BEGIN
DBMS_REPCAT.CREATE_MASTER_REPOBJECT(
sname => 'scott',
oname => 't_emp',
type => 'table',
use_existing_object => false,
ddl_text => 'create table scott.t_emp as select * from scott.emp',
copy_rows => true,
gname => 'scott_mg');
END;
/
```

```
BEGIN
  DBMS_REPCAT.GENERATE_REPLICATION_SUPPORT (
    sname => 'scott',
    oname => 't_emp',
    type => 'TABLE',
    min_communication => TRUE);
END;
/
```

```
BEGIN
  DBMS_REPCAT.RESUME_MASTER_ACTIVITY (
    gname => 'scott_mg');
END;
/
```

4. 执行 DDL 操作，可以创建表，但是不会被加入复制组中

```
EXECUTE DBMS_REPCAT.EXECUTE_DDL(gname => 'scott_mg',ddl_text => 'truncate
table scott.t_emp');
```

5. 检查 dblink

```
COLUMN DEST HEADING 'Destination' FORMAT A45
COLUMN TRANS HEADING 'Def Trans' FORMAT 9999
```

```
SELECT DBLINK DEST, COUNT(*) TRANS
  FROM DEFTRANDEST D
  GROUP BY DBLINK;
```

6. 检查 push 的 JOB

```
COLUMN JOB HEADING 'Job ID' FORMAT 999999
COLUMN PRIV_USER HEADING 'Privilege|Schema' FORMAT A10
COLUMN DBLINK HEADING 'Destination' FORMAT A40
COLUMN BROKEN HEADING 'Broken?' FORMAT A7
```

```
SELECT J.JOB,
       J.PRIV_USER,
       S.DBLINK,
       J.BROKEN
FROM DEFSCCHEDULE S, DBA_JOBS J
WHERE S.DBLINK != (SELECT GLOBAL_NAME FROM GLOBAL_NAME)
AND S.JOB = J.JOB
ORDER BY 1;
```

7. 检查错误的管理请求

```
SELECT * FROM DBA_REPCATLOG WHERE STATUS = 'ERROR';
```

8. 清除错误的管理请求

```
BEGIN
DBMS_REPCAT.PURGE_MASTER_LOG (
  id      IN  BINARY_INTEGER,
  source  IN  VARCHAR2,
  gname   IN  VARCHAR2);
END;
/
```

9. 统计事务总量

```
SELECT COUNT(DISTINCT DEFERRED_TRAN_ID) "Transactions Queued"
FROM DEFTRANDEST;
```

10. 显示所有的延迟事务处理错误

```
COLUMN DEFERRED_TRAN_ID HEADING 'Deferred|Transaction|ID' FORMAT A11
COLUMN ORIGIN_TRAN_DB HEADING 'Origin|Database' FORMAT A15
COLUMN DESTINATION HEADING 'Destination|Database' FORMAT A15
COLUMN TIME_OF_ERROR HEADING 'Time of|Error' FORMAT A22
COLUMN ERROR_NUMBER HEADING 'Oracle|Error|Number' FORMAT 999999
```

```
SELECT DEFERRED_TRAN_ID,
       ORIGIN_TRAN_DB,
       DESTINATION,
       TO_CHAR(START_TIME, 'DD-Mon-YYYY hh24:mi:ss') TIME_OF_ERROR,
       ERROR_NUMBER
```

```
FROM DEFERROR ORDER BY START_TIME; */
```

11. 显示错误百分比

```
SELECT DECODE(TOTAL_TXN_COUNT, 0, 'No Transactions',  
              (TOTAL_ERROR_COUNT/TOTAL_TXN_COUNT)*100) "ERROR PERCENTAGE"  
FROM DEFSCCHEDULE  
WHERE DBLINK = 'ORCL.DB.COM';
```

12. 以 repadmin 身份重新执行错误的事务

```
BEGIN  
  DBMS_DEFER_SYS.EXECUTE_ERROR (  
    deferred_tran_id => '3.26.1613',  
    destination => 'ORA10G.DB.COM');  
END;  
/
```

13. 以实际用户身份重新执行错误的事务

```
BEGIN  
  DBMS_DEFER_SYS.EXECUTE_ERROR_AS_USER (  
    deferred_tran_id => '1.12.2904',  
    destination => 'ORC2.WORLD');  
END;  
/
```

14. 清空错误事务

```
BEGIN  
  DBMS_DEFER_SYS.DELETE_ERROR(  
    deferred_tran_id    IN  VARCHAR2,  
    destination         IN  VARCHAR2);  
END;  
/
```

15. 暂停复制，该操作只是在当前 session 内有效

```
EXECUTE DBMS_REPUTIL.REPLICATION_OFF();
```

16. 恢复复制

```
EXECUTE DBMS_REPUTIL.REPLICATION_ON();
```

17. 暂停复制传播，延迟事务仍然会生成，只是不再传播，注意必须 commit 才能在其它 session 中也生效

```
exec DBMS_DEFER_SYS.SET_DISABLED(destination => 'orcl2003.db.com',disabled =>  
true);  
commit;
```

作上述操作以后，每当 PUSH 的 JOB 定时执行的时候，将会在 alertlog 中产生错误，但是没有关系，等到重新开始复制的时候，自然就不会报错了。

错误信息如下：

Thu Apr 22 01:06:34 2004

Errors in file c:\oracle\admin\orcl\udump\orcl\_j001\_3604.trc:

ORA-12012: 自动执行作业 43 出错

ORA-23354: 对 "ORCL2003.DB.COM" (带 " ") 禁用延迟执行 RPC

ORA-06512: 在"SYS.DBMS\_DEFER\_SYS", line 1716

ORA-06512: 在"SYS.DBMS\_DEFER\_SYS", line 1804

ORA-06512: 在 line 1

#### 18. 重新开始复制传播

```
exec DBMS_DEFER_SYS.SET_DISABLED(destination => 'orcl2003.db.com',disabled =>
false);
commit;
```

#### 19. 删除生成的延迟事务

```
exec DBMS_DEFER_SYS.DELETE_TRAN ( deferred_tran_id => '4.16.166', destination =>
'ORCL2003.DB.COM');
```

```
commit;
```

如果要删除一个准备传播到某个 dest 上的所有事务，则

```
exec DBMS_DEFER_SYS.DELETE_TRAN (deferred_tran_id => NULL, destination =>
'ORCL2003.DB.COM');
```

```
commit;
```

#### 20. 插入测试数据，测试高级复制环境是否正常运行

```
begin
  for i in 1..1000
  loop
    insert into emp(empno,ename,hiredate) values (seq_dept.nextval,'itpub',sysdate);
  end loop;
end;
/
```

## 附录四。FAQ

**Q:** 如果高级复制环境中的主体定义站点损坏，如何将主体定义站点切换到另外的主体站点上？

**A:** 分为两种情况。

**备注:** 每次运行完 repcat 包以后都应该执行一次 commit，因为某些 rep 的存储过程是不会自动 commit 的，同时这也是一个 troubleshooting，一般的 rep 脚本都会较快的返回结果，

如果一条命令之后长时间没有结果返回，那么很可能是上面的命令没有 **commit**，取消掉当前的命令，然后作一次 **commit**，再重新执行，一般都能够解决问题。

一是只有主体定义站点损坏。假设站点 A 是主体定义站点，已经损坏，在复制环境中还有站点 B，想作为新的主体定义站点。

1. 以 repadmin 身份登录站点 B，执行主体站点切换。

```
connect repadmin/repadmin
execute dbms_repcat.relocate_masterdef
(gname => 'groupname',
 old_masterdef => 'oldmaster.world',
 new_masterdef => 'newmaster.world',
 notify_masters => true,
 include_old_masterdef => false);
```

2. 将站点 A 作为主体站点删除

```
execute dbms_repcat.remove_master_databases
(gname => 'groupname',
 master_list => 'oldmasterdef.world');
```

3. 当站点 A 重新可用时，用 repadmin 用户登录站点 A，删除其中的复制组信息

```
connect repadmin/repadmin
execute dbms_repcat.drop_master_repgroup
(gname => 'groupname',
 drop_contents => true,
 all_sites => false);
```

如果要使站点 A 重新成为复制环境中的一个主体站点，继续执行下面的 4, 5 两步，否则切换主体定义站点就已经完成了。

4. 登录站点 B（新的主体定义站点）

```
connect repadmin/repadmin

execute dbms_repcat.suspend_master_activity
(gname => 'groupname')

execute dbms_repcat.add_master_database
(gname => 'groupname',
 master => 'oldmasterdef.world',
 use_existing_objects => true,
 copy_rows => false);
```

5. 重新开始复制

```
execute dbms_repcat.resume_master_activity
(gname => 'groupname')
```

第二种情况是一些主体站点和主体定义站点同时损坏了。

1. 依次登录所有正常运行的主体站点，执行主体定义站点切换

```
execute dbms_repcat.relocate_masterdef
(sname => 'schemaname',
 old_masterdef => 'oldmaster.world',
 new_masterdef => 'newmaster.world',
 notify_masters => false, /*此处是 false，而第一种情况中这个参数是 true */
 include_old_masterdef => false);
```

后面的操作步骤跟情况一相同，依次执行 2—5 就可以了。

**Q:** 如果在高级复制环境中的定义了多个复制组，而且想让不同的复制组在不同的时间间隔后进行复制传播，如何处理？

**A:** 我们可以使用 **Connection Qualifiers** 来定义多个 **dblink** 指向同一个远程主体站点。Connection qualifiers allow several database links pointing to the same remote database to establish connections using different paths. For example, a database named ny can have two public database links named ny.world that connect to the remote database using different paths.

- ny.world@ethernet, a link that connects to ny using an ethernet link
- ny.world@modem, another link that connects to ny using a modem link

For the purposes of replication, connection qualifiers can also enable you to more closely control the propagation characteristics for multiple master groups. Consider, if each master site contains three separate master groups and you are not using connection qualifiers, then the scheduling characteristics for the propagation of the deferred transaction queue is the same for all master groups. This may be costly if one master group propagates deferred transactions once an hour while the other two master groups propagate deferred transactions once a day.

Associating a connection qualifier with a master group gives you the ability to define different scheduling characteristics for the propagation of the deferred transaction queue on a master group level versus on a database level as previously described.

When you create a new master group, you can indicate that you want to use a connection qualifier for all scheduled links that correspond to the group. However, when you use connection qualifiers for a master group, Oracle propagates information only after you have created database links with connection qualifiers at every master site. After a master group is created, you cannot remove, add, or change the connection qualifier for the group.

---

**Caution:**

To preserve transaction integrity in a multimaster environment that uses connection qualified links and multiple master groups, a transaction cannot manipulate replication objects in groups with different connection qualifiers.

---

**Note:**

If you plan to use connection qualifiers, then you probably need to increase the value of the OPEN\_LINKS initialization parameter at all master sites. The default is four open links for each process. Estimate the required value based on your usage.

**Q:** 如何在异步复制环境中，实现模拟持续的实时同步的复制操作？

**A:** 即使设置为异步复制，我们也可以配置 scheduled link 来模拟连续实时的复制。

*在设置 PUSH 作业时需要设置的参数*

SCHEDULE_PUSH 存储过程参数	值
delay_seconds	1200
interval	Lower than the delay_seconds setting
parallelism	1 or higher
execution_seconds	Higher than the delay_seconds setting

With this configuration, Oracle continues to push transactions that enter the deferred transaction queue for the duration of the entire interval. If the deferred transaction queue has no transactions to propagate for the amount of time specified by the delay\_seconds parameter, then Oracle releases the resources used by the job and starts fresh when the next job queue process becomes available.

If you are using serial propagation by setting the parallelism parameter to 0 (zero), then you can simulate continuous push by reducing the settings of the delay\_seconds and interval parameters to an appropriate value for your environment. However, if you are using serial propagation, simulating continuous push is costly when the push job must initiate often.

The following is an example that simulates continual pushes:

```
BEGIN
  DBMS_DEFER_SYS.SCHEDULE_PUSH (
    destination => 'orc2.world',
    interval => 'SYSDATE + (1/144)',
    next_date => SYSDATE,
    parallelism => 1,
    execution_seconds => 1500,
    delay_seconds => 1200);
END;
/
```

To configure continuous purging of a site's deferred transaction queue, you can use the DBMS\_DEFER\_SYS.SCHEDULE\_PURGE procedure and specify the settings.

*在设置 PURGE 作业时需要设置的参数*

SCHEDULE_PURGE 存储过程参数	值
-----------------------	---

SCHEDULE_PURGE 存储过程参数	值
delay_seconds	500000
interval	Lower than the delay_seconds setting
purge_method	dbms_defer_sys.purge_method_quick

You can also use the Replication Management tool to configure continuous purge. To do so, on the Purge sub tab of the Schedule tab on the Administration property sheet, set Delay Seconds to 500,000 and set interval (the "then purge every" control) to a value less than the Delay Seconds setting.

The following is an example that simulates continuous purges:

```
BEGIN
  DBMS_DEFER_SYS.SCHEDULE_PURGE (
    next_date => SYSDATE,
    interval => 'SYSDATE + (1/144)',
    purge_method => dbms_defer_sys.purge_method_quick,
    delay_seconds => 500000);
END;
/
```

但是在实际应用中却发现如果这样来模拟持续 PUSH 或者 PURGE 的话, 会导致 Oracle 数据库无法正常 shutdown 而必须使用 shutdown abort 来关闭数据库, 这是一个比较严重的问题, 基本情况是这样的。

最近几天忽然发现自己的测试数据库出现无法用 shutdown immediate 来关闭, 而只能 shutdown abort 的情况。键入 shutdown immediate, 数据就停在那里, 没有任何反应, 可以确认已经没有任何用户连接, 也不存在任何大的事务正在执行, 等待 1 个小时以上, 也仍然无法关闭数据库。

我的测试数据库有两个, 一个是 windows 下的, 一个是 linux 下的, 都是 9201, 懒得升级了。

今天检查 windows 下数据库的 DBA\_REPCATLOG 视图, 发现有两个高级复制的管理请求状态一直是 AWAIT\_CALLBACK, 解决方法不说了, 在后面有详细叙述, 解决完了之后, 发现这个数据库可以正常关闭了。

晚上检查 linux 下的数据库, 发现仍然不能正常关闭。于是在 shutdown immediate 之后, 查看 alertlog 文件, 发现如下内容。

```
Thu Apr 29 23:58:29 2004
```

```
Shutting down instance: further logons disabled
```

```
Shutting down instance (immediate)
```

```
License high water mark = 9
```

```
Fri Apr 30 00:03:34 2004
```

```
Active call for process 5376 user 'kamus' program 'oracle@defora.db.com (J002)'
```

```
Active call for process 5360 user 'kamus' program 'oracle@defora.db.com (J000)'
```

SHUTDOWN: waiting for active calls to complete.

Fri Apr 30 00:45:29 2004

License high water mark = 9

Instance terminated by USER, pid = 7581 --用 shutdown abort 停了数据库

可以看到 oracle 一直在等待 J000 和 J002 这两个进程完成自己的工作，这是 Job Queue Process，由此我想到了在这个测试库中高级复制的 PUSH 和 PURGE 两个 JOB 的设置，其中用到了模拟持续复制的功能。

于是进一步检查 v\$sqlprocess, v#session, v\$sqlarea 视图，发现确实没错，这两个进程就是 PUSH 和 PURGE 的 JOB 进程。

然后修改了 JOB 的定义，重新 shutdown immediate 成功。

通过对比 shutdown 前后的进程，也发现 oracle 会先停所有 queue 和 job 相关的进程，然后再去停 pmon, dbwr 等进程。

关于这个问题的其它讨论，也可以参见下面的链接：

<http://www.itpub.net/showthread.php?s=&threadid=218773>

**Q: 如何计算延迟事务将占用多少的资源?**

**A: Deferred Transactions**

Oracle forwards data replication information by propagating (that is, sending and executing) the RPCs that are generated by the internal triggers described previously. These RPCs are stored in the deferred transaction queue. In addition to containing the execution command for the internal procedure at the destination site, each RPC also contains the data to be replicated to the target site. Oracle uses distributed transaction protocols to protect global database integrity automatically and ensure data survivability.

#### **Deferred Transaction Queue**

This queue stores the transactions (for example, DML) that are bound for another destination in the master group. Oracle stores RPCs produced by the internal triggers in the deferred transaction queue of a site for later propagation. Oracle also records information about initiating transactions so that all RPCs from a transaction can be propagated and applied remotely as a transaction. Oracle's replication facility implements the deferred transaction queue using Oracle's advanced queuing mechanism.

上面是 Oracle 联机文档中对于延迟事务和延迟事务队列的描述。

开始的时候一直以为延迟事务应该是存在 Oracle 的一个内存结构中，所以总是担心如果复制环境中的网络长时间出现问题，那么会不会导致延迟事务队列占用大量的内存而使数据库的其它操作变慢，或者说超出了延迟事务可以使用的内存大小而产生错误。

因为上面提到延迟事务队列使用的是 Oracle 的高级队列 (Advanced Queue) 算法，所以又查找了高级队列的文档，发现多处提到 Table 这个词，所以忽然明白所谓延迟事务的队列应该是存储在磁盘上的某些表中，这样陡然就解决了心中很多疑问，首先事务多只是占用硬盘空间，其次要想计算事务占用的资源可以通过表的 block 数来计算。

于是通过 SQL Trace，找到了延迟事务相关视图的基表。

deftran 对应 DEF\$\_AQCALL 表，通过执行计划也发现在统计大量延迟事务总数时候速度极为缓慢的原因，因为在作 TABLE ACCESS FULL DEF\$\_AQCALL，同时还会作 TABLE ACCESS

FULL DEF\$\_AQERROR, 还有 UNION ALL 的操作。  
deferror 对应 DEF\$\_ERROR 表。defcall 对应的也是 DEF\$\_AQCALL 和 DEF\$\_AQERROR 表。  
现在我们检查一下 DEF\$\_AQCALL 表的信息。

```
SQL> col owner for a10
SQL> col object_name for a20
SQL> select owner,object_name,object_id,data_object_id,object_type from dba_objects
where object_name='DEF$_AQCALL';
```

OWNER	OBJECT_NAME	OBJECT_ID	DATA_OBJECT_ID	OBJECT_TYPE
SYS	DEF\$_AQCALL	3913		SYNONYM
SYSTEM	DEF\$_AQCALL	3861	3861	TABLE
SYSTEM	DEF\$_AQCALL	3869		QUEUE

Executed in 0.06 seconds

从上面的结果可以看到这个表是属于 SYSTEM 的, 在 SYS 下有一个同义词。  
再检查一下 segment 的情况, 我们可以从 dba\_segments 或者 dba\_extents 视图中查看。

```
SQL> col tablespace_name for a20
SQL> select owner,tablespace_name,bytes,blocks from dba_segments where
segment_name='DEF$_AQCALL';
```

OWNER	TABLESPACE_NAME	BYTES	BLOCKS
SYSTEM	SYSTEM	65536	8

Executed in 0.11 seconds

由此我们已经可以知道作为高级复制中延迟事务存储所占用的资源, 同时由于这是普通的表, 那么当插入记录的时候当然也是会缓存在 buffer cache 中。这里不作讨论。  
也不再讨论 DEF\$\_AQERROR 表, 因为只有在延迟事务产生错误时才会插入, 如果高级复制环境中没有太多错误, 这个表的资源占用可以不考虑。

为了继续验证, 现在测试环境中 REP\_HOME 复制组, 其中的复制对象是 SCOTT.EMP\_2003 表。断开网络连接的情况下, 我们插入 10000 条记录。  
再次检索 DEF\$\_AQCALL 表现在的情况。

```
SQL> select owner,tablespace_name,bytes,blocks from dba_segments where
segment_name='DEF$_AQCALL';
```

OWNER	TABLESPACE_NAME	BYTES	BLOCKS
SYSTEM	SYSTEM	2097152	256

Executed in 0.08 seconds

现在该表的大小已经扩大到 2M，可以认为所有的复制数据和队列信息都存储在这张表中。

```
SQL> select count(*) from DEF$_AQCALL;
```

```
      COUNT(*)  
-----  
      10000
```

Executed in 0.04 seconds

再次证明确实是 10000 条数据。

如果此时我们用 repadmin 用户检索 defcall 和 deftran 视图

```
SQL> select count(*) from defcall;
```

```
      COUNT(*)  
-----  
      10000
```

已用时间: 00: 00: 02.04

```
SQL> select count(*) from deftran;
```

```
      COUNT(*)  
-----  
              1
```

已用时间: 00: 00: 00.00

可以看到 deftran 视图中只有一条记录，因为上面的 10000 条数据的插入是一个事务中完成的，所以在复制环境中作为一个延迟事务处理。

而 defcall 中则是 10000 条记录，详细查看内容，知道所有的 cal 事务号都相同，而 callno 不同，同时我们发现直接检索 DEF\$\_AQCALL 只需要 0.04 秒，而检索 defcall 视图却需要 2 秒，所以如果想要计算到底有多少数据需要处理的时候，我们可以直接从 DEF\$\_AQCALL 检索，这样可以缩短查询时间。

**Q: 如何在高级复制环境中实现自动 Failover?**

**A:** 可以使用 Oracle Net 去配置自动的 connect-time failover，可以实现如果一台主体站点损坏那么 Oracle Net 会将请求自动 fail over 到另一台主体站点上。

方法是在 tnsnames.ora 中配置 FAILOVER 选项为 on，同时指定多个连接描述

具体配置文件如下所示：

测试方法：



```

                                argcnt => argcnt,
                                argsize => argsize,
                                tran_db => tran_db,
                                tran_id => tran_id,
                                date_fmt => 'YYYY-MM-DD HH24:MI:SS',
                                types => v_types,
                                vals => v_vals);

FOR indx IN 1 .. argcnt LOOP
    IF v_types(indx) = 1 THEN
        v_type_desc := 'VARCHAR2';
    END IF;
    IF v_types(indx) = 2 THEN
        v_type_desc := 'NUMBER';
    END IF;
    IF v_types(indx) = 12 THEN
        v_type_desc := 'DATE';
    END IF;
    IF v_types(indx) = 23 THEN
        v_type_desc := 'RAW';
    END IF;
    IF v_types(indx) = 96 THEN
        v_type_desc := 'CHAR';
    END IF;
    IF v_types(indx) = 11 THEN
        v_type_desc := 'ROWID';
    END IF;
    dbms_output.put_line('Arg ' || indx || ': Datatype ' ||
                        v_type_desc || '; Value: ' ||
                        v_vals(indx));
END LOOP;
END GET_CALL;

```

另外我们可以通过查询 DEF\$\_AQCALL 表的 USER\_DATA 字段也可以得到数据，这是一个 BLOB 字段，提取 BLOB 字段内容的方法这里不再讨论，如果有兴趣的可以自己试一下。

**Q: 高级复制环境中出现长时间的网络问题会出现什么情况？**

**A:** 由于高级复制的传播都是通过 JOB 来实现的，而大家知道 Oracle 对于 JOB 的执行有个限制，就是如果一个 JOB 执行失败了 16 次，那么这个 JOB 将会被标志为 BROKEN，以后这个 JOB 再也不会被自动执行，除非是手动设置 BROKEN 为 FALSE 或者手动成功地运行一次 JOB。

这个特性给我们的实际应用中带来了一些麻烦，假设我们的 PUSH JOB 定义的时间间隔是一分钟，那么如果主体站点之间的网络出现长时间的问题，比如说超过了 16 分钟，也就是此时 JOB 已经失败了 16 次，那么 PUSH 的 JOB 就被标志为 BROKEN 了，这样等到网络问题

修复，会发现堆积的延迟事务也不会被 PUSH 到其它的主体站点上。

如果不注意这个问题，往往就会出现严重的问题。

解决方案是另外作一个 JOB，这个 JOB 里面每隔一定时间自动检查那个 PUSH JOB 的状态，如果是 BROKEN 的，那么自动将其 BROKEN 状态重新设置为 FALSE，这样下次又可以重新执行了。

这个 JOB 中执行的存储过程基本上如下：

```
DECLARE
  CURSOR my_broken_jobs IS
    SELECT job FROM user_jobs WHERE broken = 'Y';
BEGIN
  FOR broken_job IN my_broken_jobs LOOP
    BEGIN
      dbms_job.broken(broken_job.job, FALSE);
    EXCEPTION
      WHEN OTHERS THEN
        NULL;
    END;
  END LOOP;
END;
```

**Q: 如果主定义站点的 DBA\_REPCATLOG 视图中的管理请求总是处于 AWAIT\_CALLBACK 状态是什么原因，如何处理？**

**A: AWAIT\_CALLBACK 状态表示管理请求已经从主体定义站点上正常发出，并且已经被其他的主体站点接收，但是其他的主体站点却没有返回消息表明管理请求是否执行成功，造成这种错误基本上是网络的原因。我利用以下环境模拟这种情况的产生。**

机器 A 是主体定义站点，在局域网内用 ADSL 拨号上 internet，机器 B 是另外一个主体站点，通过静态 IP 直接联入 internet。这样的环境中机器 A 可以连通机器 B，但是机器 B 却无法连通机器 A。

在机器 A 上发出停顿复制组的命令：

```
execute dbms_repcat.suspend_master_activity (gname => 'rep_gp_day');
```

然后检查 A 的复制组状态和管理请求状态，发现 A 中的复制组状态已经为 quiesced，但是 DBA\_REPCATLOG 视图中始终显示这个请求处于 AWAIT\_CALLBACK 状态，在这种情况下后续的所有管理请求都将不会执行了。

处理方法是：

首先解决网络问题，当网络重新连通的时候，利用 DBMS\_REPCAT 包重新执行这个管理请求。

```
DBMS_REPCAT.DO_DEFERRED_REPCAT_ADMIN ( gname=>'rep_gp_day', all_sites=>>true);
```

**Q: 如果在添加主体站点的时候碰到 ORA-23375 错误，如何解决？**

**A: 有些时候我们在执行下面命令添加主体站点的时候会碰到 ORA-23375 错误。**

```
SQL>execute dbms_repcat.add_master_database(.....)
```

ERROR 位于第 1 行：

ORA-23375: 特性与数据库版本<GLOBAL\_NAME>不兼容

ORA-06512: 在"SYS.DBMS\_SYS\_ERROR", line 86

ORA-06512: 在"SYS.DBMS\_REPCAT\_MAS", line 2151

ORA-06512: 在"SYS.DBMS\_REPCAT", line 146

ORA-06512: 在 line 1

上面的这个错误，通常表示我们在创建 repadmin 用户的时候缺少了几步授权的命令，请对照本文创建 repadmin 用户的步骤仔细检查。

通常缺少的是下面这两步：

```
grant lock any table to repadmin;
```

```
grant select any dictionary to repadmin
```

**注：**本文第六章和附录二完全由 **yangtingkun** 编写。本文第四，五章完全由 **eygle** 编写。