

# Oracle9i 实体化视图

*Oracle 白皮书*

2001 年 5 月

# Oracle9i 实体化视图

执行概要.....	3
简介.....	3
为何使用概要管理.....	4
概要管理组件.....	4
模式需求.....	5
维度.....	5
定义维度的相关提示.....	6
实体化视图.....	7
创建实体化视图.....	8
使用自己预先建立的实体化视图.....	9
实体化视图的索引选择.....	9
实体化视图可以完成哪些工作？.....	9
实体化视图失效.....	10
安全含义.....	10
加载和刷新实体化视图.....	11
完全刷新.....	11
快速刷新.....	12
刷新和约束条件.....	12
数据可用性.....	13
分区维护操作和实体化视图.....	13
查询重写.....	13
启用/禁用查询重写.....	14
查询重写的类型.....	14
完全匹配.....	14
概要联接返回.....	15
概要汇总或聚合所有.....	15
数据子集.....	15
查询重写完整性方式.....	16
结果是否正确.....	17
解释重写.....	17
概要顾问.....	18
提供负载.....	18
推荐实体化视图.....	19
实现推荐.....	19
过滤负载.....	20
估计实体化视图的大小.....	20
结论.....	20

## 执行概要

今天的数据库，无论是数据仓库、数据中心还是 OLTP 系统，都包含大量的信息等待人们去发现和理解。然而，如何以一种及时的方式查找和表示这些信息是一个重大的问题，尤其是当需要搜索庞大数量信息的时候。

实体化视图能够帮助解决这个问题，因为它提供了一种快速访问和报告数据的方法。

## 简介

实体化视图首先在 Oracle8i 中引入，是称为“概要管理”的组件的一部分。可能您的公司已经在使用实体化视图，但只知道它的其他名字，例如概要或聚合表。在这里我们讨论如何创建和管理实体化视图，还讨论查询重写功能如何透明地重写 SQL 查询，从而使用实体化视图来缩短查询响应时间。这将使数据库用户完全无需知道存在哪些实体化视图。

实体化视图应看作是一种特殊的视图，它物理上存在于数据库内部，可以包括联接和/或聚合。它能够在执行之前预先计算开销大的联接和聚合操作，因此它的存在缩短了查询执行时间。

今天，使用自身概要的公司花费了大量的时间用于手工创建概要、识别将创建哪些概要、对概要进行索引和更新，以及建议用户使用哪些概要。

现在 DBA 将仅须在开始时创建实体化视图，而无论数据源何时发生变化，它都将被自动更新。此外还有一个概要顾问组件，它向 DBA 推荐创建、删除和保留哪些实体化视图。

数据仓库或数据库用户将可以体会到使用实体化视图的最大好处之一，DBA 无须再告诉他们存在哪些实体化视图。他们可以对数据库中的表或视图编写自己的查询。然后 Oracle 服务器的查询重写机制将自动重写 SQL 查询以使用实体化视图。这样就大大缩短了查询响应时间，终端用户无须“了解概要”。

## 为何使用概要管理

当向数据仓库终端用户问起他们希望从中获得什么，大部分人都会回答：快速准确的信息。但是这也给数据仓库设计者出了个大难题：为了回答“在 y 地点我们卖出多少件 x 产品”，同时希望避免读取表中的每一行，必须建立一条到数据的快速路由。

解决此问题最常见的办法之一就是创建概要表，Oracle 将其称为实体化视图。这一工作包括首先要理解典型负荷，然后创建规模非常小的实体化视图，实体化视图中可以包含所需信息的联接和/或聚合。例如，为了回答前面的问题，实体化视图中每种产品对应于一行，指明每个区域的销售量。因此如果一家公司在 5 个地点销售 2000 件产品，则将要读取的最大行数始终为 10000，而无论已经售出多少商品。

很明显，实体化视图必须保证精确，但该技术意味着终端用户现在需要读取的行数很少，因此可以始终快速地接收结果。数据库容量已经增长到兆兆字节，因此使用这样的方法来缩短查询响应时间就显得越来越重要。今天许多站点都创建了自己的概要表，因此使用 Oracle8 概要管理所带来的额外好处是：

- Oracle 中的查询重写机制是透明的并采用实体化视图（即使它仅能部分满足查询的需要）。
- 具有高级的查询重写，可以使用实体化视图对不同聚合级别（例如按照星期、月和年）进行报告。
- 自动化机制刷新实体化视图，单个请求刷新所有实体化视图。
- DBA 不再需要花时间查找应创建哪些实体化视图。系统将基于过去对数据库或数据仓库的查询，向 DBA 提供有关需要哪些概要的信息。

## 概要管理组件

组成概要管理的有五个组件：

- 维度
- 实体化视图
- 刷新
- 查询重写
- 概要顾问

并不需要使用所有组件，但所选用的组件越多，获得的优势就越多。现在我们将详细探讨这些组件。

## 模式需求

用于实体化视图的模式类型或设计没有什么限制。因此在数据仓库环境中，模式可以是雪花式的设计，但这并不是必须的。

对于熟悉产品系统中数据库设计技术的设计者来说，在一个数据仓库中必须使用不同的规则和技术。例如，产品数据库通常是规范化的，因此在这种情况下，时间维的表示方法最好是采用三个表：*日*、*月*、*年*。联接条件应该满足：将每个*日期*行连接到一个（仅一个）*月份*行，每个月份行连接到一个（仅一个）*年份*行。数据仓库实现通常将导致一个完全非规范化的*时间*维表，其中*日期*、*月份*、*年份*栏都处于同一个表中。不过，无论设计使用的是规范化还是非规范化表，都可以使用实体化视图。

## 维度

在创建一个实体化视图之前，第一步应该是回顾模式，指明维度。维度定义了列之间的层次化（父级/子级）关系，所有的列无须来自同一个表。我们强烈推荐定义数据的维度，因为这将有助于查询重写和概要顾问做出更佳决策。

数据库设计者所面临的另一个问题是，频繁查询将不会直接涉及所有的维度列，而仅参考与维度相关的那一列，例如查询仅参考星期二而不是具体日期。因此当定义了维度之后，还必须描述维度列和表中其它列之间的关系。

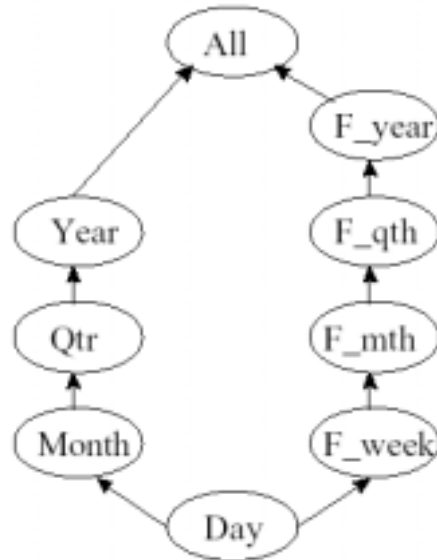
图 1 显示了包含两个层次的时间维。从一个指定日期开始，有一个层次告诉我们该日期涉及哪些财政周、月或年，而另一个层次定义了日、月、季度和年之间的关系。

当定义了一个层次之后，可以指定多个列来描述该层次，例如，如果 *City* 在每个 *State* 之内是唯一的，但是在 *States* 之间不唯一，那么就需要指定一个地理层次，其形式如（*Country, State, <State, City>*），从而满足 *City* 级到 *State* 级之间严格的 1:n 层次关系。

可以使用两种方法来创建维度：

- SQL CREATE DIMENSION 语句，如图 2 所示
- Oracle 企业管理器中的维度向导

图 1 介绍时间维



按照图 1 画出维度，可以帮助 DBA 完成定义过程。每个圆圈代表维度中的一个级别。通过 **LEVEL** 子句来声明。维层次通过 **HERARCHY** 子句来声明。概要管理也同样依赖于 DBA 定义约束条件，保证层次级别中每一级别的列非空。

在图 2 中，我们可以看到创建该维度的 SQL 语句。级别名称对应于维表中的列。然后使用这些级别名称来描述每一层次。最后，使用 **ATTRIBUTE** 子句来定义具有直接关系

的项目。因此属性 `calendar_month_name` 与级别 `month` 有关系。

使用 **JOIN KEY** 子句来声明维度中的 1:n 联接关系。在事实表和维表之间，使用事实表中的 **FOREIGN KEY** 和 **NOT NULL** 约束条件来表示这种联接关系。

### 定义维度的相关提示

为帮助创建维度，请按照下面的简单步骤：

1. 指明模式中的所有维度和维表。如果维度是规范化的，即它存储在多个表中，那么请检查维表之间的联接，确保每个子级行联接到一个（仅一个）父级行。对于非规范化维度，请检查子级列是否唯一确定父级（或属性）列。如果不遵守这些规则，可能会在查询时得到错误的结果。
2. 指明每一维度中的层次。例如，`day` 是 `month` 的子级（我们可以将 `day` 级别聚合到 `month`），`quarter` 是 `year` 的子级。
3. 指明层次中每一级别的属性依赖关系。例如，指明 `calendar_month_name` 是 `month` 的属性。
4. 指明数据仓库中每个事实表到维度之间的联接，检查每个联接，确保每个事实行联接到一个（仅一个）维度行。必须声明该条件，而且还可以选择是否强制执行该条件，其方法是向事实关键列添加 **FOREIGN KEY** 和 **NOT NULL** 约束条件，向父级联接键添加 **PRIMARY KEY** 约束条件。可以通过 **NOVALIDATE** 选项来启用这些约束条件，从而无须花费时间来验证表中的每一行是否满足这些约束条件。对于所有未得到验证的约束条件，还需要新的 **RELY** 子句来使其能够用于查询重写中。

**图 2 创建时间维的 SQL 语句**

```
CREATE DIMENSION times_dim
LEVEL day IS TIMES.TIME_ID
LEVEL month IS TIMES.CALENDAR_MONTH_DESC
LEVEL quarter IS TIMES.CALENDAR_QUARTER_DESC
LEVEL year IS TIMES.CALENDAR_YEAR
LEVEL fis_week IS TIMES.WEEK_ENDING_DAY
LEVEL fis_month IS TIMES.FISCAL_MONTH_DESC
LEVEL fis_quarter IS TIMES.FISCAL_QUARTER_DESC
LEVEL fis_year IS TIMES.FISCAL_YEAR
HIERARCHY cal_rollup
( day CHILD OF
month CHILD OF
quarter CHILD OF
year )
HIERARCHY fis_rollup
( day CHILD OF
fis_week CHILD OF
fis_month CHILD OF
fis_quarter CHILD OF
fis_year )
ATTRIBUTE day DETERMINES
(day_number_in_week, day_name, day_number_in_month,
calendar_week_number)
ATTRIBUTE month DETERMINES
Oracle9i 实体化视图 第8页
(calendar_month_desc, calendar_month_number,
calendar_month_name, days_in_cal_month,
end_of_cal_month)
ATTRIBUTE quarter DETERMINES
(calendar_quarter_desc, calendar_quarter_number,
days_in_cal_quarter, end_of_cal_quarter)
ATTRIBUTE year DETERMINES
(calendar_year, days_in_cal_year, end_of_cal_year)
ATTRIBUTE fis_week DETERMINES
(week_ending_day, fiscal_week_number) ;
```

## 实体化视图

一旦定义了维度，就可以创建实体化视图。现在我们将着重介绍什么是实体化视图，在后面我们将看到建议功能如何推荐创建哪些实体化视图。

*实体化视图*定义可包括聚合，例如 SUM、MIN、MAX、AVG、COUNT(\*)、COUNT(x)、COUNT(DISTINCT)、VARIANCE 或 STDDEV，还可以包括一个或多个联接到一起的表和一个 GROUP BY。可以进行索引和分区，还可以应用基本的 DDL 操作，例如 CREATE、ALTER 和 DROP。

由于实体化视图是数据库中的一个对象，因此在很多方面它更像一个索引，因为

- 实体化视图的目的是提高查询执行性能。
- 实体化视图的存在对于 SQL 应用程序是透明的，因此 DBA 可以在任何时候创建或删除实体化视图而不影响 SQL 应用程序。
- 实体化视图占用存储空间，当底层详情表被修改时应该对其进行更新。

许多站点都已经拥有了数据仓库，并在其中定义自己的概要。因此可以通过查询重写来注册现有概要，而不是强迫用户从零开始重新生成概要。

### 创建实体化视图

使用 `CREATE MATERIALIZED VIEW` 语句创建实体化视图。图 3 介绍了如何创建一个名为 `costs_mv` 的实体化视图，通过 `time` 和 `prod_nam` 来计算 `costs` 的总和。

当定义了一个实体化视图之后，必须遵守一些简单规则。**SELECT** 列必须包含所有 **GROUP BY** 列，而 **GROUP BY** 列必须为简单列。需要聚合的表达式可以是任何的 SQL 值表达式，不包含子查询或网状聚合功能。

**WHERE** 子句仅允许包含基列上的内部同等联接谓词。实体化视图可以拥有自己的存储规范，这样就可以指定它存储在哪个表空间中以及其区域的大小。您还可以将分区子句包括进来，以便将实体化视图的内容存储在多个表空间中。

表和视图都可以用在实体化视图的定义中。因此，参考前面的例子，成本可以作为一个表，而产品可以作为一个视图。如果视图在使用 `SYSDATE` 和 `USER` 这样的功能时没有根据用户变化的数据，那么就可以使用任何视图。

### 图 3 创建实体化视图的 SQL 语句

```
CREATE MATERIALIZED VIEW costs_mv
PCTFREE 0
STORAGE (initial 8k next 8k pctincrease 0)
BUILD IMMEDIATE
REFRESH FAST ON DEMAND
ENABLE QUERY REWRITE
AS
SELECT time_id, prod_name,
SUM( unit_cost) AS sum_units,
COUNT(unit_cost) AS count_units,
COUNT(*) AS cnt
FROM costs c, products p
WHERE c.prod_id = p.prod_id
GROUP BY time_id, prod_name;
```

## 使用自己预先建立的实体化视图

已经拥有数据仓库的站点可能已经具备了概要和过程来维护视图。因此，他们希望利用查询重写功能，而不用重新创建概要。可以使用 CREATE MATERIALIZED VIEW 语句以及 ON PREBUILT TABLE 子句将已经存在的表注册为实体化视图。

实体化视图的名称必须与表的名称相同，同时还必须提供描述创建该表的查询的 SELECT 子句。可能无法始终保证查询的精度与表的精度匹配。为了克服此问题，应该在规范中包含 WITH REDUCED PRECISION 子句。

## 实体化视图的索引选择

根据实体化视图中行的数目和是否进行增量式刷新，可能需要创建实体化视图的索引。因此，应该考虑首先创建一个唯一的本地索引，其中包含所有实体化视图关键字。其他索引可以包含实体化视图关键字列上的单列位图索引。

当创建索引时，不要忘记考虑每个索引对存储空间的要求以及对刷新时间的影响。

## 实体化视图可以完成哪些工作？

在创建实体化视图之前或刚刚创建时，DBA 可能会对实体化视图产生这样的疑问：它究竟能做什么？它可以快速刷新吗？如果不，为什么呢？DBMS\_MVIEW.EXPLAIN\_MVIEW 过程可以提供该信息。

考虑我们在图 3 中创建的实体化视图，如果我们从定义中删除 COUNT(\*)，然后调用图 4 中介绍的过程 DBMS\_MVIEW.EXPLAIN\_MVIEW。它将告诉我们，分区改变跟踪 (PCT) 可用，因为成本表已经被分区，所有类型的查询重写都是可能的。然而，由于实体化视图中没有 COUNT(\*)，因此 DML 之后快速刷新已经不可能了。

## 图 4 解释实体化视图示例

```
TRUNCATE TABLE mv_capabilities_table;

EXEC DBMS_MVIEW.EXPLAIN_MVIEW (' SELECT time_id,
                                prod_name,SUM( unit_cost) AS sum_units,
                                COUNT(unit_cost) AS count_units, COUNT(*)
                                AS cnt FROM costs c, products p
                                WHERE c.prod_id = p.prod_id
                                GROUP BY time_id, prod_name');

SELECT capability_name, possible, related_text, msgtxt
FROM mv_capabilities_table;

PCT_TABLE:Y COSTS:
PCT_TABLE:N
PRODUCTS:      relation is not a partitioned table
REFRESH_COMPLETE:Y
```

```
REFRESH_FAST:Y
REFRESH_FAST_AFTER_ANY_DML:N
see the reason why REFRESH_FAST_AFTER_ONETAB_DML is
disabled
REFRESH_FAST_AFTER_INSERT:Y
REFRESH_FAST_AFTER_ONETAB_DML:N
COUNT(*) is not present in the select list
REFRESH_FAST_PCT:Y
REWRITE:Y
REWRITE_FULL_TEXT_MATCH:Y
REWRITE_GENERAL:Y
REWRITE_PARTIAL_TEXT_MATCH:Y
REWRITE_PCT:Y
```

### 实体化视图失效

实体化视图受到持续监控，确保其中包含的数据是最新的。使实体化视图失效的目的是保证不会返回已经失效的数据。当实体化视图所基于的对象改变时，它将被标记为失效。

可以通过查询表 USER\_MVIEWS 来判定实体化视图的状态。如果对实体化视图的状态有什么疑问，可以发出 ALTER MATERIALIZED VIEW COMPILE 命令来确保了解其最新状态。

### 安全含义

数据库中的某些信息可能需要限制访问，查询重写可以被视为一种迂回安全机制。然而，由于所有的安全确认都是在 Oracle9i 服务器中进行的，所以可以向数据和实体化视图提供更加强大的保护。为了避免对实体化视图或详情表的未授权访问，使用 CREATE MATERIALIZED VIEW 时将需要 CREATE MATERIALIZED VIEW 权限、对详情表的 SELECT WITH GRANT 权限、以及对实体化视图容器对象的 SELECT WITH GRANT 和 INSERT 权限。此外，如果用户在某次请求中对表进行访问，而这些表中已经定义了一个或多个实体化视图，那么该用户将被允许访问实体化视图，而不考虑赋予实体化视图容器表的权限。因此无论查询源自何处，只有通过了安全检查才能够访问数据。

## 加载和刷新实体化视图

在历史上，概要表使用过程中的一个问题就是概要的初始加载和随后更新。这些问题现在得到了解决，因为概要管理提供了一系列机制实现下列目的：

- 对数据进行完全刷新
- 执行快速刷新，仅添加/合并发生变化的内容
- 无论何时作出更改，都自动更新实体化视图。

因此，DBA 必须考虑需要多少时间来创建和维护每个实体化视图，并在花费时间与使用该实体化视图所获得的性能改善之间进行平衡。

Oracle 9i 提供了下面的刷新方法：

- 完全刷新
- 快速刷新（仅适用于发生变化的内容）
- 强制刷新，在可能时进行快速刷新，否则进行完全刷新。

这些操作可以

- 按需刷新，由
  - 特殊实体化视图 (DBMS\_MVIEW.REFRESH)
  - 依赖于某一表的实体化视图 (DBMS\_MVIEW.REFRESH\_DEPENDENT)
  - 所有实体化视图 (DBMS\_MVIEW.REFRESH\_ALL\_MVIEWS)
- 提交刷新，只要定义实体化视图所依据的表发生变化。

按需刷新通过调用上面所列的过程之一来实现，从而当实体化视图发生改变时赋予 DBA 完全的控制权限。

如果选择了提交刷新，只要实体化视图受到源数据变化的影响，它都将自动更新以反映该数据。然而，请记住这一更新是作为基表发生更改的事务中提交处理的一部分而发生的。因此，提交刷新所花费时间会稍微长一些，因为需要对原表和所有定义中包含该表的实体化视图同时作出更改。

### 完全刷新

当实体化视图完全刷新时，首先截断该视图，然后加载所有数据。鉴于实体化视图的大小，这可能会是一个很耗时的过程。完全刷新在下列情况下是一项很好的技术：

- 需要插入的新行的数量比实体化视图所基于的表的基数多 50% 以上。

- 实体化视图没有可用于合并的索引
- 进行快速刷新所需要的时间比完全刷新所需时间更长

### 快速刷新

有一些实体化视图可能会非常大，可能不具有执行例行完全刷新所需的时间。一种替代方法是进行快速刷新，仅将事实表中变化的部分应用到实体化视图中。加载到数据仓库中任何表的新数据将被识别，然后参考该表的任何实体化视图将利用新数据进行自动更新。

为了进行快速刷新操作，必须将数据中的变化记录下来，有两种方法可以实现这一目的。如果曾经使用 SQL\*Loader 直接路径将数据插入到数据库中，那么刷新机制将检测到这一过程，指明将要加载的新数据。然而，大多数数据变化都是通过 SQL 命令 INSERT、UPDATE 和 DELETE 来进行。在这种情况下，实体化视图所基于的每个表都需要一份 MATERIALIZED VIEW LOG。

每个表仅需要一份日志，实体化视图日志在表中，而不在实体化视图中。因此如果数据库中仅有 6 个表发生了变化，则将仅需要 6 份实体化视图日志。但可以让任何数量的实体化视图来使用这些日志。

请注意，并非所有的实体化视图都可以进行快速刷新，可以通过调用 DBMS\_MVIEW.EXPLAIN\_MVIEW 过程来确认其是否可以快速刷新。该过程还建议需要对实体化视图进行哪些操作，从而使其可以快速刷新。

### 刷新和约束条件

前面曾经声明过，理想情况下，约束条件（尤其是外部关键字约束条件）应该在事实表中定义，从而确保事实表中的一行可以与一个维度相匹配。当提到词约束条件时，一些 DBA 或许会挥手示意，宣称由于可能的性能开销，该数据库中将不再有约束条件。

然而，DBA 可以确信，通过使用

```
ALTER TABLE <table name> ENABLE NOVALIDATE  
CONSTRAINT <name> 子句，
```

可以立即启用约束条件而无需检查数据。如果使用 SQL\*Loader 直接路径将数据加载到事实表中，那么默认情况下所有的约束条件都将被禁用。加载事实表之后，发出启用语句 NOVALIDATE 会立即启用约束条件而无须检查数据。因此它对于数据加载时间没有任何影响，启用约束条件不需要花费时间。然而，由于对于加载的数据没有进行验证，因此确保所有加载的数据不会影响到任何完整性约束条件就尤为重要。

## 数据可用性

数据虽然已经刷新，但实体化视图仍然可用，不过可以通过使用命令 `ALTER SYSTEM SET QUERY_REWRITE_ENABLED = FALSE` 禁用查询重写，直至所有实体化视图都已经刷新。另一方面，对于那些不需要实体化视图反映最新加载数据的用户，可以在会话级启用查询重写。

## 分区维护操作和实体化视图

Oracle9i 提供了一个称为分区变化跟踪（PCT）的组件，该组件可以透明地检测到分区发生变化，然后判定该操作是否使得实体化视图中的数据不一致。例如，合并分区或添加分区将不会影响到实体化视图，执行这样的过程不会导致实体化视图被标记为失效。

分区变化跟踪还可以用于指明分区操作影响了哪些实体化视图行。例如，如果一个详情表分区被截断或删除，PCT 将指明实体化视图中受影响的行并将其删除。

DBMS\_MVIEW.EXPLAIN\_MVIEW 过程将建议某一实体化视图是否可以使用 PCT。

## 查询重写

在概要管理的诸多优点中，终端用户真正称赞的主要优点之一是查询重写功能。这是一项查询优化技术，能够转换根据表和视图来编写的用户查询，从而从实体化视图中攫取数据来提高执行速度。由于 Oracle9i 服务器可以自动重写任何合适的 SQL 应用程序以使用实体化视图，因此它对于终端用户是透明的，无需 SQL 应用程序的干预或提示。虽然本文档中的所有引用都指的是 SQL SELECT 子句，但查询重写仍将依赖于包含 SELECT 子句的 INSERT 和 CREATE TABLE 语句。

查询重写可以用于各种查询。请注意，对于维度对象中声明的关系无需强制执行，但假设它们是真的。如果声明的关系与表数据中存在的实际关系不匹配，那么当查询重写使用存在问题的关系声明来重写查询时，重写查询很可能会产生错误结果。不过，通过定义关系和使用约束条件以便系统确保数据的正确性，所生成的报告将是可信的，其中包含正确的结果。当实现了系统完整性后，可以仅投入最小的努力和开销就获得快速、准确的查询结果。

查询的组成不再需要与实体化视图的定义精确匹配，因为这要求 DBA 事先知道应该针对数据执行哪些查询。当然这是不可能的，尤其对数据仓库更是如此，因为数据仓库对于公司的主要好处之一就是立即执行新的查询。因此，即使使用实体化视图仅可满足查询的一部分，查询重写还是会发生。

## 启用/禁用查询重写

当设置了下列参数时，查询重写将发生：

```
ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE
```

或

```
ALTER SYSTEM SET QUERY_REWRITE_ENABLED = TRUE
```

或当定义实体化视图时，通过包括子句 `ENABLE QUERY REWRITE` 以使其符合查询重写条件。

有时可能需要禁用查询重写，这可以通过将上面的参数改为 `FALSE` 来实现，也可以在一个具体实体化视图上使用 `DISABLE QUERY REWRITE` 子句来实现。

## 查询重写的类型

Oracle9i 中可以有各种查询重写类型，下面的例子介绍了使用图 5 中的实体化视图可以实现哪些功能。

### 图 5 实体化视图用于查询重写示例

```
CREATE MATERIALIZED VIEW all_cust_sales_mv
BUILD IMMEDIATE
REFRESH COMPLETE
ENABLE QUERY REWRITE
AS
SELECT    c.cust_id,
          p.prod_id,
          sum(s.amount_sold) AS dollars,
          sum(s.quantity_sold) as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
GROUP BY c.cust_id, p.prod_id;
```

### 完全匹配

当实体化视图定义与查询定义完全匹配时，将会发生最简单的查询重写。也就是说，`FROM` 子句中的表、`WHERE` 子句中的联接和 `GROUP BY` 子句中的关键字在查询和实体化视图之间完全匹配。例如，给出下面的查询：

```
SELECT    c.cust_id,
          sum(s.quantity_sold) as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
GROUP BY c.cust_id, p.prod_id;
```

Oracle9i 重写此查询以使用 `all_cust_sales_mv`

### 概要联接返回

有时候查询可能会参考某一行，该行没有存储在概要表中，但是可以通过将实体化视图联接返回到合适的维表来重新获得。例如，在前面的例子中，假设报告不是根据用户 ID，而是使用用户名。

```
SELECT  c.cust_last_name,
        sum(s.quantity_sold) as quantity
FROM    sales s, customers c, products p
WHERE   c.cust_id = s.cust_id
AND     s.prod_id = p.prod_id
GROUP  BY c.cust_last_name, p.prod_id;
```

此次查询参考 `c.cust_last_name` 列，该列不在实体化视图 `all_cust_sales_mv` 中，但 `c.cust_last_name` 在功能上取决于 `c.cust_id`，这是由它们之间的层次关系决定的。这意味着本次查询可以通过 `all_cust_sales_mv` 来重写，因为 `all_cust_sales_mv` 被联接返回到客户表以获得 `c.cust_last_name` 列。

### 概要汇总或聚合所有

当查询在某一层次中请求聚合（如 SUM（销售）），该聚合的级别比聚合在实体化视图中存储的级别更高时，可以通过使用实体化视图，以及将该聚合汇总到所需级别从而对该查询进行重写。

例如，我们的实体化视图 `all_cust_sales_mv` 是在客户级别上组织数据的，但我们可能更希望在状态级别上仅由客户来报告数据。我们已经创建了客户维，它描述了客户和区域之间的关系。因此下面的查询可以使用我们的实体化视图 `all_cust_sales_mv` 来产生报告，将某一客户的所有数据都聚合起来，并汇总到状态级别。

```
SELECT  c.cust_state_province,
        sum(s.quantity_sold) as quantity
FROM    sales s, customers c, products p
WHERE   c.cust_id = s.cust_id
AND     s.prod_id = p.prod_id
GROUP  BY c.cust_state_province;
```

### 数据子集

迄今为止，我们已经看到的所有实体化视图包含了全部的数据，但这仍会导致一个非常庞大的实体化视图。Oracle9i 允许定义一个实体化视图，它仅包含图 6 中所示的部分数据，即 Dublin, Galway, Hamburg 和 Istanbul 的数据。

**图 6 包含数据子集的实体化视图**

```
CREATE MATERIALIZED VIEW some_cust_sales_mv
BUILD IMMEDIATE
REFRESH COMPLETE
ENABLE QUERY REWRITE
AS
SELECT    c.cust_id,
          p.prod_id,
          sum(s.amount_sold) AS dollars,
          sum(s.quantity_sold) as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
AND c.cust_state_province IN
      ('Dublin','Galway','Hamburg','Istanbul')
GROUP BY c.cust_id, p.prod_id;
```

现在该实体化视图可以用于满足包含范围的查询，

例如下面所示的 IN 和 BETWEEN 子句。

```
SELECT    c.cust_state_province,
          sum(s.quantity_sold) as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
AND c.cust_state_province IN ('Dublin','Galway')
GROUP BY c.cust_state_province;
```

### 查询重写完整性方式

概要管理将尽力识别不一致的实体化视图并打上相应标记，但是为了克服这些问题，可以通过参数 QUERY\_REWRITE\_INTEGRITY 来选择三种完整性级别。

- STALE\_TOLERATED
- TRUSTED
- ENFORCED (default)

在 STALE\_TOLERATED 方式中将始终使用一个实体化视图，即使它已经失效。在 TRUSTED 方式中，优化程序相信实体化视图中的数据是最新的，在维度中声明的关系和 RELY 约束条件是正确。在这一方式中，优化程序还将使用预先建立的实体化视图或基于视图的实体化视图，并使用未执行和已经得到执行的关系。在这种方式中，优化程序还“信任”已经声明但尚未 ENABLED VALIDATED 的主/唯一关键字约束条件和通过维度指明的数据关系。

ENFORCED 方式是默认方式，在这种方式中，优化程序将只使用确知其包含最新数据的实体化视图，以及只使用基于 ENABLED VALIDATED 主/唯一/外部关键字约束条件的关系。因此您将发现，如果一些约束条件尚未生效，那么使用该方法的查询重写将不会发生。但如果使用的是约束性较小的 TRUSTED 或 STALE\_TOLERATED 模式，这些查询重写则会发生。

## 结果是否正确

当 SQL 查询使用实体化视图而非真实的数据源时，有时候会出现返回结果不同的情况。

1. 实体化视图可能与详细数据不同步。发生这种情况的原因通常是刷新过程挂起，同时选择了 STALE\_TOLERATED 完整性方式。
2. 联接列可能影响参考的完整性。在这种情况下，一些子端的行没有正确汇总到父端的行。为了避免发生这种情况，请使用系统强制执行的完整性，这样开销是微不足道的，而获利却很显著。

当实体化视图包含了不再位于详情数据中的行的信息时，可以创建一个 *rolling materialized view*。例如，实体化视图可能包含 18 个月的数据，但详情表中仅包含最近 6 个月的。因此，如果查询将在基表中而不是实体化视图中进行，那么将显示不同的结果。

## 解释重写

当使用查询重写时，询问最频繁的问题是“此查询会重写吗？”或“为什么此查询不重写？” Oracle9i 通过 DBMS\_MVIEW.EXPLAIN\_REWRITE 过程给出了这一问题的解决办法，图 7 中显示了一个使用范例。因此，可以在查询运行之前了解该信息。

查询文本作为一个长字符串传递，该过程存储其在表 REWRITE\_TABLE 中找到的结果，必须查询该表才能查看过程的结果。在下面的例子中，我们将看到 some\_cust\_sales\_mv 用于此次查询。

**图 7 解释重写示例**

```
DECLARE
querytxt VARCHAR2(1500) := 'SELECT c.cust_id,
sum(s.amount_sold) AS dollars, p.prod_id,
sum(s.quantity_sold) as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
AND c.cust_state_province IN
('Dublin','Galway','Hamburg',
'Istanbul')
GROUP BY c.cust_id, p.prod_id';

BEGIN
dbms_mview.Explain_Rewrite(querytxt, NULL, 'ID1');
END;

/

SELECT message FROM rewrite_table;

MESSAGE
-----
QSM-01009:materialized view, SOME_CUST_SALES_MV, matched
query text
QSM-01033:query rewritten with materialized view,
SOME_CUST_SALES_MV
```

## SUMMARY ADVISOR

当首次决定使用实体化视图时，必须定义一个初始集。目前，这对 DBA 而言是一个巨大的挑战，尤其是当他们对商务并非透彻理解，或者应用程序提出的查询根本不可预测时。

为了帮助解决此问题，概要管理包含了一个称为 Summary Advisor 的组件，可以通过调用过程或从 Oracle 企业管理器中调用该组件。Summary Advisor 提供下列信息：

- 基于收集或假定工作量推荐实体化视图
- 估计实体化视图的大小
- 报告基于收集工作量的实体化视图的实际利用率
- 定义用于工作量的过滤器
- 加载和生效工作量
- 清除过滤器、工作量和结果

在使用 Summary Advisor 之前，DBA 应该运行 DBMS\_STATS 过程，以收集数据库中表和实体化视图的基数信息。该信息将用作预测进程的一部分。

### 提供负载

虽然 Summary Advisor 无需负载就能够推荐实体化视图，但有负

载时效果最好，在 Oracle9i 中可以用下面的形式来提供：

- 用户定义 (DBMS\_OLAP.LOAD\_WORKLOAD\_USER)
- SQL 缓存的当前内容 (DBMS\_OLAP.LOAD\_WORKLOAD\_CACHE)
- Oracle Trace 收集的查询 (DBMS\_OLAP.LOAD\_WORKLOAD\_TRACE)

用户定义的负载包括将查询存储在数据库的表中。然后 Summary Advisor 读取该查询，将其作为自己的负载。

另一方面，SQL 缓存中的当前查询可以作为负载，输入到 Summary Advisor 中。

如果可以使用 Oracle Trace，那么将提供一个称为 *Summary Workload* 的事件集。当启用时，它将收集负载统计信息，包括查询重写使用的实体化视图名称、通过使用实体化视图而获得的预计“好处”，以及可能已经使用的理想化实体化视图。

虽然一次仅可以使用一项负载，输入到推荐过程 RECOMMEND\_MVIEW\_STRATEGY，但还是可以在数据库中存储多个负载，然后进行比较，找出哪个负载可以产生最佳推荐。

## 推荐实体化视图

对创建哪些实体化视图进行推荐可以通过两种方式来完成，一种是使用 Oracle 企业管理器中的 Summary Advisor，它将带领逐步完成推荐实体化视图的整个过程并实现推荐。

另一种方法是调用过程 RECOMMEND\_MVIEW\_STRATEGY 来生成推荐。Summary Advisor 将不考虑所选择的方法，而是推荐究竟是删除还是维持现有实体化视图，以及在需要时创建哪些实体化视图。

还可以使用 DBMS\_OLAP.GENERATE\_MVIEW\_REPORT 过程来产生可选报告，该过程提供了advisor 的推荐、实体化视图的使用和所考虑查询的有关信息。

### 实现推荐

使用 Summary Advisor 向导的诸多好处之一是它将自动实现推荐。如果使用了

DBMS\_OLAP.RECOMMEND\_MVIEW\_STRATEGY 过程，它将产生一系列推荐存储在数据库中。然后可以调用 DBMS\_OLAP.GENERATE\_MVIEW\_SCRIPT 过程来创建一个 SQL 文件，其中包含实现这些推荐所需的语句。

## 过滤负载

无须对整个负载进行考虑，可以使用 DBMS\_OLAP.ADD\_FILTER\_ITEM 对其进行过滤。过滤器可以运用于应用程序名、查询中所使用的表、表中的基数、查询频率、上次使用查询的日期、表的属主、查询优先级、查询响应时间或跟踪收集名。然后将这些过滤器中的一个或多个运用到用于推荐实体化视图的负载中。

## 估计实体化视图的大小

DBA 的另一有利优势是，他可以在创建实体化视图之前估计其大小。通过将查询作为一个参数输入到 DBMS\_OLAP.ESTIMATE\_MVIEW\_SIZE 中，能够预测实体化视图中行的数目和其可能大小，如图 9 所示：

图 9 估计实体化视图的大小

```
DECLARE
no_of_rows NUMBER;
mv_size NUMBER;
BEGIN
dbms_olap.estimate_summary_size ('MV 1',
'SELECT c.cust_id, sum(s.amount_sold) AS
dollars,p.prod_id, sum(s.quantity_sold)
as quantity
FROM sales s , customers c, products p
WHERE c.cust_id = s.cust_id
AND s.prod_id = p.prod_id
GROUP BY c.cust_id, p.prod_id' ,
no_of_rows, mv_size );

DBMS_OUTPUT.put_line ( '');
DBMS_OUTPUT.put_line ( 'No of Rows:' || no_of_rows );
DBMS_OUTPUT.put_line ( 'Size of Materialized view (bytes):
' ||
mv_size ); END;

No of Rows: 245504
Size of Materialized view (bytes): 21604352
```

## 结论

对于期望提高数据仓库或数据库查询性能的人来说，如果他们能够预先计算一些查询的结果，那么就on应该好好考虑一下实现实体化视图。创建实体化视图所需要的工作量非常少，Summary Advisor 将向您建议创建哪些视图，甚至还提供执行推荐的脚本。一旦建立起实体化视图，它事实上将可以自我维护。终端用户在不需要更改任何 SQL 代码的情况下，就可以看到查询响应时间有了显著的改善。

# ORACLE

Oracle9i 实体化视图  
2001年5月  
作者：Dr. Lilian Hobbs  
协作者：

Oracle Corporation  
全球总部  
500 Oracle Parkway  
Redwood Shores, CA 94065  
U. S. A.

全球咨询热线：  
电话： +1.650.506.7000  
传真： +1.650.506.7200  
[www.oracle.com](http://www.oracle.com)

Oracle 公司提供该软件  
that powers the Internet.

Oracle 是 Oracle Corporation 的注册商标。  
本文中提及的各种产品和服务的名称可能是  
Oracle Corporation 的商标。其它所有提及  
的产品和服务名称可能是各自所有者的商标。

版权所有 © 2000 Oracle 公司  
保留所有权利。