Building and Optimizing Data Warehouse "Star Schemas" with MySQL

Bert Scalzo, Ph.D.

Bert.Scalzo@Quest.com
About the Author

Oracle DBA for 20+ years, versions 4 through 10g
Been doing MySQL work for past year (4.x and 5.x)
Worked for Oracle Education & Consulting
Holds several Oracle Masters (DBA & CASE)
BS, MS, PhD in Computer Science and also an MBA
LOMA insurance industry designations: FLMI and ACS

Books
- The TOAD Handbook (Feb 2003)
- Oracle DBA Guide to Data Warehousing and Star Schemas (Jun 2003)

Articles
- Oracle Magazine
- Oracle Technology Network (OTN)
- Oracle Informant
- PC Week (now E-Magazine)
- Linux Journal
- www.linux.com
- www.quest-pipelines.com
Oracle DBA
Guide to Data Warehousing and Star Schemas

The definitive, real-world guide to Oracle data warehousing
- Maximizing performance, flexibility, and manageability in production environments
- Industrial-strength data loading and query optimization techniques
- By the world-renowned architect of 7-Eleven's multi-terabyte data warehouse

THE PRENTICE HALL PTR ORACLE SERIES
THE INDEPENDENT VOICE ON ORACLE
“Star schema” approach to dimensional data modeling was pioneered by Ralph Kimball

**Dimensions:** smaller, de-normalized tables containing business descriptive columns that users use to query

**Facts:** very large tables with primary keys formed from the concatenation of related dimension table foreign key columns, and also possessing numerically additive, non-key columns used for calculations during user queries
How much data would a data miner mine, if a data miner could mine data?

**Dimensions:** generally queried selectively to find lookup value matches that are used to query against the fact table

**Facts:** must be selectively queried, since they generally have hundreds of millions to billions of rows – even full table scans utilizing parallel are too big for most systems

*Business Intelligence (BI) tools generally offer end-users the ability to perform projections of group operations on columns from facts using restrictions on columns from dimensions …*
Often, people have expectation that using expensive hardware is only way to obtain optimal performance for a data warehouse.

- **CPU**
  - SMP
  - MPP

- **Disk**
  - 15,000 RPM
  - RAID (EMC)

- **OS**
  - UNIX
  - 64-bit

- **MySQL**
  - 4.x / 5.x
  - 64-bit
In reality, the database design is the key factor to optimal query performance for a data warehouse built as a “Star Schema”

There are certain minimum hardware and software requirements that once met, play a very subordinate role to tuning the database

Golden Rule: get the basic database design and query explain plan correct
Key Tuning Requirements

1. MySQL 5.x
3. MySQL.ini
5. Table Design
7. Index Design
9. Data Loading Architecture
11. Analyze Table
13. Query Style
15. Explain plan
1. MySQL 5.X (help on the way)

- Index Merge Explain
  - Prior to 5.x, only one index used per referenced table
  - This radically effects both index design and explain plans
- Rename Table for MERGE fixed
  - With 4.x, some scenarios could cause table corruption
- New "greedy search" optimizer that can significantly reduce the time spent on query optimization for some many-table joins

- Views
  - Useful for pre-canning/forcing query style or syntax (i.e. hints)
- Stored Procedures
- Rudimentary Triggers
- InnoDB
  - Compact Record Format
  - Fast Truncate Table
2. MySQL.ini

- `query_cache_size` = 0 (or 13% overhead)
- `sort_buffer_size` >= 4MB
- `bulk_insert_buffer_size` >= 16MB
- `key_buffer_size` >= 25-50% RAM
- `myisam_sort_buffer_size` >= 16MB
- `innodb_additional_mem_pool_size` >= 4MB
- `innodb_autoextend_increment` >= 64MB
- `innodb_buffer_pool_size` >= 25-50% RAM
- `innodb_file_per_table` = TRUE
- `innodb_log_file_size` = 1/N of buffer pool
- `innodb_log_buffer_size` = 4-8 MB
3. Table Design

SPEED vs. SPACE vs. MANAGEMENT

64 MB / Million Rows (Avg. Fact)
500 Million Rows

32,000 MB (32 GB)

Primary storage engine options:
- MyISAM
- MyISAM + RAID_TYPE
- MERGE
- InnoDB
CREATE TABLE `ss.pos_day` (  
    PERIOD_ID decimal(10,0) NOT NULL default '0',  
    LOCATION_ID decimal(10,0) NOT NULL default '0',  
    PRODUCT_ID decimal(10,0) NOT NULL default '0',  
    SALES_UNIT decimal(10,0) NOT NULL default '0',  
    SALES_RETAIL decimal(10,0) NOT NULL default '0',  
    GROSS_PROFIT decimal(10,0) NOT NULL default '0'  
)  
PRIMARY KEY (PRODUCT_ID, LOCATION_ID, PERIOD_ID),  
ADD INDEX PERIOD(PERIOD_ID),  
ADD INDEX LOCATION(LOCATION_ID),  
ADD INDEX PRODUCT(PRODUCT_ID)  
) ENGINE=MyISAM  
PACK KEYS  
DATA_DIRECTORY= 'C:\mysql\data'  
INDEX_DIRECTORY= 'D:\mysql\data';

Pros:  
• Non-transactional – faster, lower disk space usage, and less memory

Cons:  
• 2/4GB data file limit on operating systems that don't support big files  
• 2/4GB index file limit on operating systems that don't support big files  
• One big table poses data archival and index maintenance challenges (e.g. drop 1998 data, make 1999 read only, rebuild 2000 indexes, etc)
CREATE TABLE ss.pos_day  
  PERIOD_ID decimal(10,0) NOT NULL default '0',
  LOCATION_ID decimal(10,0) NOT NULL default '0',
  PRODUCT_ID decimal(10,0) NOT NULL default '0',
  SALES_UNIT decimal(10,0) NOT NULL default '0',
  SALES_RETAIL decimal(10,0) NOT NULL default '0',
  GROSS_PROFIT decimal(10,0) NOT NULL default '0'
  PRIMARY KEY (PRODUCT_ID, LOCATION_ID, PERIOD_ID),
  ADD INDEX PERIOD(PERIOD_ID),
  ADD INDEX LOCATION(LOCATION_ID),
  ADD INDEX PRODUCT(PRODUCT_ID)
)  
  ENGINE=MyISAM
  PACK_KEYS
  RAID_TYPE=STRIPED;

Pros:
• Non-transactional – faster, lower disk space usage, and less memory
• Can help you to exceed the 2GB/4GB limit for the MyISAM data file
• Creates up to 255 subdirectories, each with file named table_name.myd
• Distributed IO – put each table subdirectory and file on a different disk

Cons:
• 2/4GB index file limit on operating systems that don't support big files
• One big table poses data archival and index maintenance challenges (e.g. drop 1998 data, make 1999 read only, rebuild 2000 indexes, etc)
CREATE TABLE ss.pos_merge (  
  PERIOD_ID decimal(10,0) NOT NULL default '0',  
  LOCATION_ID decimal(10,0) NOT NULL default '0',  
  PRODUCT_ID decimal(10,0) NOT NULL default '0',  
  SALES_UNIT decimal(10,0) NOT NULL default '0',  
  SALES_RETAIL decimal(10,0) NOT NULL default '0',  
  GROSS_PROFIT decimal(10,0) NOT NULL default '0',  
  INDEX PK(PRODUCT_ID, LOCATION_ID, PERIOD_ID),  
  INDEX PERIOD(PERIOD_ID),  
  INDEX LOCATION(LOCATION_ID),  
  INDEX PRODUCT(PRODUCT_ID)  
) ENGINE=MERGE  

Pros:
• Non-transactional – faster, lower disk space usage, and less memory
• Partitioned tables offer data archival and index maintenance options (e.g. drop 1998 data, make 1999 read only, rebuild 2000 indexes, etc)
• Distributed IO – put individual tables and indexes on different disks

Cons:
• MERGE tables use more file descriptors on database server
• MERGE key lookups are much slower on "eq_ref" searches
• Can use only identical MyISAM tables for a MERGE table
CREATE TABLE `ss.pos_day`(
    `PERIOD_ID` decimal(10,0) NOT NULL default '0',
    `LOCATION_ID` decimal(10,0) NOT NULL default '0',
    `PRODUCT_ID` decimal(10,0) NOT NULL default '0',
    `SALES_UNIT` decimal(10,0) NOT NULL default '0',
    `SALES_RETAIL` decimal(10,0) NOT NULL default '0',
    `GROSS_PROFIT` decimal(10,0) NOT NULL default '0',
    PRIMARY KEY (`PRODUCT_ID`, `LOCATION_ID`, `PERIOD_ID`),
    ADD INDEX `PERIOD`(`PERIOD_ID`),
    ADD INDEX `LOCATION`(`LOCATION_ID`),
    ADD_INDEX `PRODUCT`(`PRODUCT_ID`)
) ENGINE=InnoDB

Pros:

• Simple yet flexible tablespace datafile configuration
  
```sql
innodb_data_file_path=ibdata1:1G:autoextend:max:2G; ibdata2:1G:autoextend:max:2G
```

Cons:

• Uses much more disk space – typically 2.5 times as much disk space as MyISAM!!
• Transaction Safe – not needed, consumes resources (SET AUTOCOMMIT=0)
• Foreign Keys – not needed, consumes resources (SET FOREIGN_KEY_CHECKS=0)
• Prior to MySQL 4.1.1 – no “multiple tablespaces” feature (i.e. one table per tablespace)
• One big table poses data archival and index maintenance challenges
  (e.g. drop 1998 data, make 1999 read only, rebuild 2000 indexes, etc)
# Space Usage 21 Million Records

## MERGE
- 3GB

### Files:
- pos_1998.frm: 9 KB FRM File
- pos_1998.MYD: 460,395 KB MYD File
- pos_1998.MYI: 604,455 KB MYI File
- pos_1999.frm: 9 KB FRM File
- pos_1999.MYD: 460,395 KB MYD File
- pos_1999.MYI: 604,455 KB MYI File
- pos_2000.frm: 9 KB FRM File
- pos_2000.MYD: 460,395 KB MYD File
- pos_2000.MYI: 604,455 KB MYI File
- pos_merge.frm: 9 KB FRM File
- pos_merge.MRG: 1 KB MRG File

## MyISAM
- 3GB

### Files:
- pos_day.frm: 9 KB FRM File
- pos_day.MYD: 1,381,183 KB MYD File
- pos_day.MYI: 1,811,830 KB MYI File

## InnoDB
- 8GB

### Files:
- ib_logfile0: 10,240 KB File
- ib_logfile1: 10,240 KB File
- ibdata1: 8,251,392 KB File
Index Design must be driven by DW users’ nature: you don’t know what they’ll query upon, and the more successful they are data mining – the more they’ll try (which is actually a really good thing) …

Therefore you don’t know which dimension tables they’ll reference and which dimension columns they will restrict upon – so:

- Fact tables should have primary keys – for data load integrity
- Fact table dimension reference (i.e. foreign key) columns should each be individually indexed – for variable fact/dimension joins
- Dimension tables should have primary keys
- Dimension tables should be fully indexed
  - MySQL 4.x – only one index per dimension will be used
    - If you know that one column will always be used in conjunction with others, create concatenated indexes
  - MySQL 5.x – new index merge will use multiple indexes
Note: Make sure to build indexes based off cardinality (i.e. leading portion most selective), so in this case the index was built backwards.
MyISAM Key Cache Magic

1. Utilize two key caches:
   - Default Key Cache – for fact table indexes
   - Hot Key Cache – for dimension key indexes

   command-line option:
   shell> mysqld --hot_cache.key_buffer_size=16M

   option file:
   [mysql]
   hot_cache.key_buffer_size=16M

   CACHE INDEX t1, t2, t3 IN hot_cache;

2. Pre-Load Dimension Indexes:

   LOAD INDEX INTO CACHE t1, t2, t3 IGNORE LEAVES;
5. Data Loading Architecture

Archive:
• ALTER TABLE fact_table UNION=(mt2, mt3, mt4)
• DROP TABLE mt1

Load:
• TRUNCATE TABLE staging_table
• Run nightly/weekly data load into staging_table
• ALTER TABLE merge_table_4 DROP PRIMARY KEY
• ALTER TABLE merge_table_4 DROP INDEX
• INSERT INTO merge_table_4 SELECT * FROM staging_table
• ALTER TABLE merge_table_4 ADD PRIMARY KEY(…)
• ALTER TABLE merge_table_4 ADD INDEX(…)
• ANALYZE TABLE merge_table_4
6. Analyze Table

ANALYZE TABLE ss.pos_merge;

• Analyze Table statement analyzes and stores the key distribution for a table

• MySQL uses the stored key distribution to decide the order in which tables should be joined

• If you have a problem with incorrect index usage, you should run ANALYZE TABLE to update table statistics such as cardinality of keys, which can affect the choices the optimizer makes
Many Business Intelligence (BI) and Report Writing tools offer initialization parameters or global settings which control the style of SQL code they generate.

Options often include:

- Simple N-Way Join
- Sub-Selects
- Derived Tables
- Reporting Engine does Join operations

For now – only the first options make sense...
(see following section regarding explain plans)
8. Explain Plan

The EXPLAIN statement can be used either as a synonym for DESCRIBE or as a way to obtain information about how the MySQL query optimizer will execute a SELECT statement.

You can get a good indication of how good a join is by taking the product of the values in the rows column of the EXPLAIN output. This should tell you roughly how many rows MySQL must examine to execute the query.

We’ll refer to this calculation as the explain plan cost — and use this as our primary comparative measure (along with of course the actual run time) . . .
Method: Derived Tables

```
SELECT prod.category_name, 
       sum(fact.sales_unit),
       sum(fact.sales_retail)
FROM pos_day fact, 
    (select period_id from period where levelx = 'DAY'
       AND period_month = 11 
       AND period_year = '1999') per, 
    (select location_id from location where levelx = 'STORE'
       AND city = 'DALLAS'
       AND state = 'TX') loc, 
    (select product_id, 
       category_name from product where levelx = 'ITEM'
       AND category_name in ('10.01 BEER','23.01 COFFEE') prod
WHERE fact.period_id = per.period_id
   AND fact.location_id = loc.location_id
   AND fact.product_id = prod.product_id
GROUP BY prod.category_name;
```

Huge Explain Cost, and statement ran forever!!!

Cost = $1.8 \times 10^{16}$th
Method: Sub-Selects

SELECT prod.category_name, 
    sum(fact.sales_unit), 
    sum(fact.sales_retail) 
FROM pos_day fact, 
    product prod 
WHERE fact.period_id in (select period_id from period where levelx = 'DAY' 
                           and period_month = 11 
                           and period_year = '1999') 
    and fact.location_id in (select location_id from location where levelx = 'STORE' 
                              and city = 'DALLAS' 
                              and state = 'TX') 
    and prod.levelx = 'ITEM' 
    and prod.category_name in ('10.01 BEER','23.01 COFFEE') 
    and fact.product_id = prod.product_id 
GROUP BY prod.category_name;

Sub-Select better than Derived Table – but not as good as Simple Join

Cost = 2.1 x 10^7
Method: Simple Joins and Single-Column Indexes

```
SELECT prod.category_name, 
      sum(fact.sales_unit), 
      sum(fact.sales_retail) 
FROM pos_day fact, 
     period per, 
     location loc, 
     product prod 
WHERE fact.period_id  = per.period_id 
AND fact.location_id  = loc.location_id 
AND fact.product_id   = prod.product_id 
AND per.levelx       = 'DAY' AND per.period_month = 11 AND per.period_year = '1999' 
AND loc.levelx       = 'STORE' AND loc.city = 'DALLAS' AND loc.state = 'TX' 
AND prod.levelx      = 'ITEM' AND prod.category_name in ('10.01 BEER','23.01 COFFEE') 
GROUP BY prod.category_name;
```

Join better than Sub-Select and much better than Derived Table

Cost = 7.1 x 10^6
Method: Merge Table and Single-Column Indexes

```
SELECT prod.category_name, 
    sum(fact.sales_unit), 
    sum(fact.sales_retail) 
FROM pos_merge fact, 
    period per, 
    location loc, 
    product prod 
WHERE fact.period_id = per.period_id 
AND fact.location_id = loc.location_id 
AND fact.product_id = prod.product_id 
AND per.levelx = 'DAY' AND per.period_month = 11 AND per.period_year = '1999' 
AND loc.levelx = 'STORE' AND loc.city = 'DALLAS' AND loc.state = 'TX' 
AND prod.levelx = 'ITEM' AND prod.category_name in ('10.01 BEER', '23.01 COFFEE') 
GROUP BY prod.category_name;
```

Same Explain Cost, but statement ran 2X faster

Cost = $7.1 \times 10^6$
Method: Merge Table and Multi-Column Indexes

```
SELECT prod.category_name, 
       sum(fact.sales_unit),
       sum(fact.sales_retail)
FROM pos_merge fact, 
    period per, 
    location loc, 
    product prod
WHERE fact.period_id = per.period_id 
AND fact.location_id = loc.location_id 
AND fact.product_id = prod.product_id 
AND per.levelx = 'DAY' AND per.period_month = 11 AND per.period_year = '1999' 
AND loc.levelx = 'STORE' AND loc.city = 'DALLAS' AND loc.state = 'TX' 
AND prod.levelx = 'ITEM' AND prod.category_name in ('10.01 BEER','23.01 COFFEE')
GROUP BY prod.category_name;
```
Method: Merge Table and Merge Indexes

MySQL 5.0 new Index Merge = best run time

Cost = 5.2 \times 10^5
Conclusion ...

• MySQL can easily be used to build large and effective “Star Schema” Data Warehouses

• MySQL Version 5.x will offer even more useful index and join query optimizations

• MySQL can be better configured for DW use through effective mysql.ini option settings

• Table and Index designs are paramount to success

• Query style and resulting explain plans are critical to achieving the fastest query run times