Optimizing Your Data Warehouse Design for Superior Performance

Lester Knutsen, President and Principal Database Consultant—Advanced DataTools Corporation
The Problem

- The database is too complex for users to understand and build queries!
- The numbers from queries do not add up!
- The data is wrong and users don’t trust it!
- When two users build separate reports, the numbers don’t match!
- Users cannot build queries that compare numbers in different data marts – inventory to sales!
- Performance is so slow!
Tips for Optimizing Your Data Warehouse Design

• Build a dimensional model – star schema.
• Put all facts in one table; avoid double-counting.
• Standardize the grain of measurement; avoid facts with different grains of measurement.
• Build standard calculated fields in the database; avoid having users do standard calculations.
• Set defaults in the database schema: no nulls in the database.
• Include a row for “Not Found” in every table: no outer joins.
• Check the data during the loads; it’s too late if an error is found in a report.
• Build a hierarchy of aggregates of facts: summary tables or OLAP cubes.
• Build a consistent hierarchy of dimensions: rollups.
• Build an enterprise architecture for dimensions.
NEW Tips for Optimizing Your Data Warehouse Design

- Know when NOT to use Indexes
- Know when to allow Sequential Scans – and how to avoid buffer turnover problems
- Know how to Partition your data onto different disk drives
- Know when to use Dirty Reads – and how to read data avoiding locks and your database server log files
- Use ½ of your disk drives, leave the rest empty – your database throughput will be faster
- Know how to use Composite Indexes and when to use them, and how to get Index-Only Reads
- Know about Update Statistics and why it is important
Data Warehouse Architecture

Data Source

Data Warehouse Server

Staging Area and Load Database

Data Warehouse Database

Web and Hyperion Performance Suite

Web Users
Data Warehouse Design Considerations

- What is the key question you want to ask of your data?
- Organize and summarize your data to answer this question quickly.
- Allow for unexpected ad hoc queries of the data to provide new insight into your business.
- Focus on solving the business problem: What information do we need to make better business decisions?
How do you organize the data? Keep it simple.

Design Goals:
- Fast data access
- Ease of use

Star Schema:
- Fact tables - The detail data
- Dimension tables - Look-up information
- Summary tables - Presummarized results
Example Data Model

- Financial Detail Fact Table
- Labor Detail Fact Table
- Organization
- Program Codes
- Budget Object Class
- Time
The goal of the summary tables is to answer 90% of all queries.

Summary tables save system resources for handling complex ad hoc queries.
Goals of a Data Warehouse

- The data warehouse provides access to corporate or organizational data.
- Data is consistent.
- Data can be sliced and diced.
- Tools to query, analyze, and present the data are available.
- The data warehouse is a place to publish data.
- The quality of data drives business change (driver of business reengineering).
Goals of the Transactional Model

- Fast data entry
- No redundancy in data by separating data into multiple tables
- Normalization of data relationships to ensure data integrity
- Short, fast queries – for example, current customer status/balance
- Continually changing data
Problems with Querying OLTP Databases

- Relational model is too complex for users.
- Database is not designed for large queries while data is changing. (Deadlocks will occur.)
- Queries slow down transactions.

Hence the need for a separate specialized database for query and analysis –

*The data warehouse*
A design methodology determines the relevant dimensions and facts for each business process of an organization.

Facts are measurable – typically numbers.

Dimensions are reference or descriptive points for examining facts.
Facts: Numbers

- Additive (facts) – Measures in a fact table that can be added across all dimensions
- Semiadditive (facts) – Measures that can be added across some, but not all, dimensions
- Continuously valued facts – A measure that is different every time it is measured
- Fact table – The central table in a star-joined schema, characterized by a composite key made up of foreign keys to each of the dimension tables
Dimension – An independent entry in a model or organization that serves as a point for slicing the facts of an organization

Slowly changing dimensions – Tendency of a dimension to change over time (for example, customer name); three methods to handle this

Conforming dimensions – Dimensions that have exactly the same set of primary keys and same number of records in different data marts or data warehouses
• Degenerate dimension – A dimension key that has no attributes and hence no actual dimension table; for example, ‘order number’ in the stores database. (Degenerate dimensions go in the fact table.)

• Snowflake dimension – A normalized dimension where a single table is decomposed into a tree structure (Do not Snowflake, as it will be harder for users to query!)
• A grain is the meaning of a single record in the fact table.

• The determination of the grain of the fact tables is one of the key steps in the design of a star schema:
  — Snapshot grain
  — Transactional grain
  — Hierarchy consistency
  — Time consistency

• What is the scope of the fact that you are measuring?
Aggregates

- Aggregated facts – The results of an SQL statement that uses SUM, COUNT, MIN, MAX or AVG
- Aggregates – Precalculated and prestored summaries that are stored in the warehouse to improve query performance and make it easier for users to query (summary tables or OLAP cubes)
Key Steps in Design

1. Identify the business process to model: What is the key question you want to ask of the data?
2. Identify the grain: What level of detail is required to answer the question?
3. Identify the dimensions: What are the report column headings to slice the data?
4. Identify the facts: What are the numbers?
Avoid Double-Counting: Put All Facts in One Table

SQL joins tables with one (table A) to many (table B) relationships by creating a result set of all the rows. In this case, any numbers in table A will be double-counted.
Double-Counting Example: One Order Has Many Items
Double-Counting Example: Ship Charge is Double-Counted
Double-Counting: Solutions

- Custom SQL, Unions, etc: Works but is very complicated for users.
- Add a column to the fact table for Ship Charge and divide the charge by the number of items.
- Add a column to the fact table for Ship Charge and add a row with zero-item cost.
- Add a row to the item table for Ship Charge as an item.
Solution 1: Ship Charge Divided by Items

ADD A COLUMN FOR SHIP CHARGE AND DIVIDE BY THE NUMBER OF ITEMS

create table items_fact_1 (  
    item_num smallint,  
    order_num integer,  
    stock_num smallint not null ,  
    manu_code char(3) not null ,  
    quantity smallint,  
    total_price money(8,2),  
    ship_charge money(6,2), -- New Field  
    primary key (item_num,order_num)  
);  
insert into items_fact_1  
Select  
    item_num , order_num , stock_num , manu_code , quantity , total_price ,  
    ( select ( ship_charge / ( select count(*) from items b where b.order_num = items.order_num ) )  
        from orders where orders.order_num = items.order_num  
    )  
from items;
Solution 1: Ship Charge Divided by Items
ADD A COLUMN FOR SHIP CHARGE AND ADD A ROW WITH ZERO ITEM COST

insert into items_fact_2

  ( item_num , order_num , stock_num , manu_code , quantity , total_price , ship_charge )

Select

  item_num , order_num , stock_num , manu_code , quantity , total_price , 0 -- Must default to 0
from items;

-------------------------------------

insert into items_fact_2

  ( item_num , order_num , stock_num , manu_code , quantity , total_price , ship_charge )

Select

  0,
  order_num ,
  0,
  "NONE",
  0,
  0,
  Ship_Charge
from orders;
Solution 2: Add a Row for Ship Charge
Solution 3: Add a Row to the Item Table for Ship Charge as an Item

ADD A ROW TO THE ITEM TABLE FOR SHIP CHARGE AS AN ITEM

-- Insert the items
insert into items_fact_3
    ( item_num , order_num , stock_num , manu_code , quantity , total_price)
Select
    item_num , order_num , stock_num , manu_code , quantity , total_price
from items;

-- Insert the Ship Change
insert into items_fact_3
    ( item_num , order_num , stock_num , manu_code , quantity , total_price)
Select
    0,
    order_num ,
    0,
    "SHP",
    1,
    Ship_Charge
from orders;
Solution 3: Add a Row to the Item Table for Ship Charge as an Item

Ship Charge
Facts (numbers) should have the same grain in order to add up, or users may get the wrong answer:

- Snapshot grain
- Transactional grain
- Hierarchy consistency
- Time consistency
Facts with Different Grains of Measurement

- Transform all facts to a common grain. Pick the fact with the highest grain level and roll up all facts to that level. This loses some of the detail of the lowest level.

- Expand, enhance, transform some facts to a lower level of detail so that all facts have the same lowest level of detail. This may create some misleading (bad) data.

- Carefully fix and match facts in summary tables to make sure everything adds up.
Example: Facts with Different Grains of Measurement

- Agency tracks expenses at the Executing Org level (lowest branch level).
- Agency plans budget at the Budget Org level (office level).
- Hierarchy is as follows: Executing Org -> Budget Org -> State Org -> Agency.
- How do they compare budget to expenses?
Avoid Having Users Do Standard Calculations

- Two users may not agree on the same definition of a term.
- They may get the calculations wrong.
- Having users doing standard calculations runs the risk of making the data warehouse seem unfriendly and complex.
- If the answers are wrong or inconsistent, the data warehouse will be viewed as wrong.
Build Standard Calculations in a Summary Table

Financial Data Warehouse Summary Table

<table>
<thead>
<tr>
<th>Organization</th>
<th>financial_summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization_key: integer</td>
<td>Organization_key: integer</td>
</tr>
<tr>
<td>execute_org: char(5)</td>
<td>Boc_key: integer</td>
</tr>
<tr>
<td>budget_fy: smallint</td>
<td>fiscal_yr: smallint</td>
</tr>
<tr>
<td>execute_org_name: char(30)</td>
<td>project_code: char(4)</td>
</tr>
<tr>
<td>report_off: char(5)</td>
<td>planned_workmonths: decimal(16,2)</td>
</tr>
<tr>
<td>report_off_name: char(30)</td>
<td>used_workmonths: decimal(16,2)</td>
</tr>
<tr>
<td>budget_org: char(5)</td>
<td>accrual_workmonths: decimal(16,2)</td>
</tr>
<tr>
<td>budget_org_name: char(30)</td>
<td>lvc accr_workmonths: decimal(16,2)</td>
</tr>
<tr>
<td>directorate: char(5)</td>
<td>planned_dollars: decimal(16,2)</td>
</tr>
<tr>
<td>directorate_name: char(30)</td>
<td>prior_yr_revenue: decimal(16,2)</td>
</tr>
<tr>
<td>state: char(2)</td>
<td>prior_yr_direct: decimal(16,2)</td>
</tr>
<tr>
<td>state_name: char(30)</td>
<td>prior_yr_accural: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>prior_yr_total: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>current_yr_revenue: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>unliq_obligation: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>unliq_commitment: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>direct_expense: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>indirect_expense: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>payroll_accrual: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>leave_srch_direct: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>leave_srch_accrual: decimal(16,2)</td>
</tr>
<tr>
<td></td>
<td>current_yr_total: decimal(16,2)</td>
</tr>
</tbody>
</table>
• NULL plus a number equals a NULL.
• Totals on a column with one NULL will equal NULL.
• Joins with columns containing NULL are more complex:
  — from tableA, tableB
  — where ( tableA.col1 is NULL and tableB.col1 is NULL )
  — and tableA.col2 = tableB.col2
• Solution:
  — Set the default for all facts to be zero in the schema.
  — Set a default “Not Found” for all dimensions where there might be NULLs.
No Outer Joins: Include a Row for “Not Found”

- Outer joins increase the chance for user error; Result may be wrong answers.

- Solution: Add a record to every dimension table to handle data not found:
  - “Not Found”
  - “None”
  - “NULL”

- When a fact record is missing dimension data, point it to the row “Not Found”.
Check the Data During the Loads

• It’s too late if an error is found in a Hyperion report.
• Audit the load process.
• Have a report from the production system that the load process balances to in the staging area before it loads the data warehouse.
• Summary tables are your best performance tool.

• Build summary tables that will aggregate the facts by 10-100 times.

• Example:
  — Financial Detail
  — Summary by Month
  — Summary by Quarter
  — Summary by Year
Hierarchy of Aggregates of Facts

Base Fact Table
– 100,000,000 Records

Summary Table 1 – 1,000,000

Summary Table 2 – 100,000

Summary Table 3 – 10,000 Records
• Plan for drill-up
• Plan for drill-down
• Example:
  — Program_Code
  — Subactivity
  — Activity
  — Fund
  — Appropriation
NEW Tips for Optimizing Your Data Warehouse Design

- Know when NOT to use Indexes
- Know when to allow Sequential Scans – and how to avoid buffer turnover problems
- Know how to Partition your data onto different disk drives
- Know when to use Dirty Reads – and how to read data avoiding locks and your database server log files
- Use ½ of your disk drives, leave the rest empty – your database throughput will be faster
- Know how to use Composite Indexes and when to use them, and how to get Index-Only Reads
- Know about Update Statistics and why it is important
Know when NOT to use Indexes

- Using an Index requires reading an index header, several index leaf nodes, and then finally the row.
- This can be 3-6 read operations per row, depending on how many levels in the Index.
- If you need to read more than 25% of a table, the index reads may add more overhead than doing a scan of the table.
- Use Optimizer Directives to avoid indexes.
Know when to allow Sequential Scans and how to avoid buffer turnover

- Every database server has some system of Buffers for data when it has been read from disk, so the next user does not read the same data from disk.

- Problem is when you are reading large amounts of data from disk, you may get Buffer trashing.

- Calculate – How many times per hour do your Buffers turnover?

- Look into doing Light Scans – disk reads that do not go into the Buffers.
Know how to Partition your data onto different disk drives

- Disk drives can only read one spot on disk at a time
- Partition your data by some field that is used in the queries
  - Examples:
    - Fiscal Year and Month
    - State Office
- Improves multi-user performance
- Improves availability of data
- Mirror data – Most systems will read 50% from primary and 50% from mirror, thus increase read performance
Know when to use Dirty Reads – and how to read data avoiding locks and your database server log files

- Data Warehouse systems should be designed for fast reads
- Avoid logging – create a save point as part of the nightly loads
- Avoid locks – if the data is static you do not need to lock the row
- Use Dirty Reads
Use ½ of your disk drives, leave the rest empty – your database throughput will be faster

As data is added to a disk, the disk throughput will decrease

Disk Performance

Number of 2GB Chunks Accessed

Performance KB/sec.
Know how to use Composite Indexes and when to use them, and how to get Index-Only Reads

- Composite Indexes – made up of 2 or more fields
- The order of the fields in a composite index is important
  - Examples:
    - First_name, Last_name, Company
    - Company, Last_name, First_name

- Sometimes putting the fact in an index will allow Index-Only Reads
  - Example:
    - Company_code, Company_name, AR_balance
Know about Update Statistics and why it is important

- Database Servers maintain statistics so the Optimizer knows what to do:
  - Which table to read first in a join
  - Which filter to apply first in a where clause

- The better information the Optimizer has, the better job it will do on your queries
• Build a dimensional model: star schema.
• Put all facts in one table; avoid double-counting.
• Standardize the grain of measurement; avoid facts with different grains of measurement.
• Build standard calculated fields in the database; avoid having users do standard calculations.
• Set defaults in database schema: no nulls in the database.
• Include a row for “Not Found” in every table: no outer joins.
• Check the data during the loads; it’s too late if an error is found in a report.
• Build a hierarchy of aggregates of facts: summary tables or OLAP cubes
• Build a consistent hierarchy of dimensions: rollups
• Build an enterprise architecture for dimensions.
Optimizing Your Data Warehouse Design for Superior Performance

Lester Knutsen, President and Principal Database Consultant—Advanced DataTools Corporation

Slide Updates at: www.advancedatatools.com

lester@advancedatatools.com
703-256-0267 x102

Session #2100A
Please Complete Your Session Evaluation