Warehouse Data

How to develop a data warehouse using the dimensional model

Objectives:

- Entity Relation Data Model
- Decision Support Limitations of E-R Model
- The dimensional Model
- Developing a warehouse using DM
- Limitations of DM
- Describe the structure, contents, and use of the various tables in the data warehouse.
- Examples of developing a Data warehouse using DM for:
  - A grocery store Chain

Overview

- Data and tables
  - Fact
  - Dimension
  - Derived
  - Reference
  - Summary
  - Metadata
- Terminology varies

THE ENTITY RELATION DATA MODEL

- The goal of entity relation modeling is transaction performance.
- Most important difference when developing OLTP (On Line Transaction Processing) systems and data warehouses is the organization of the data in the systems (i.e., data model).
- OLTP are developed using entity relation modeling technique.
- Entity relation modeling seeks to drive all the redundancy out of the data.
- If there is no redundancy in the data, then a transaction that changes any data (or adds or deletes data) only needs to touch the database in one place.
THE ENTITY RELATION DATA MODEL

- The typical requirements for the entity relation modeling supporting operational system are based on the need to effectively support a large number of small but spontaneous read and write requests.

- Database schema definition often focuses, maximizing concurrency and optimizing insert, update, and delete performance by defining relational tables that map very efficiently to operational requests while minimizing contention for access to individual records.

THE ENTITY RELATION DATA MODEL

- Entity relation modeling works by dividing the data into many discrete entities, each of which becomes a table in the OLTP database.

- Almost every table connects to every other table. This modeling approach is very symmetric.

- All the tables look the same. There is no way to tell which table is the most important or the largest.

- If two tables in the Model are needed in a given query, there are a huge number of possible connection paths between those two tables.

- Even in a simple model, there are may be a very large number of paths, and not all perform the same.

SO WHAT IS THE PROBLEM?

- A data warehouse RDBMS typically needs to process queries that are large, complex, ad hoc, and data-intensive.

- Queries that are simple in business terms, requiring one to two pages of single-spaced SQL with ten to fifteen separate constraints.

- For queries that span many records or many tables, entity relation diagrams are too complex for users to understand and too complex for software to navigate.

SO WHAT IS THE PROBLEM?

- Every DBA and application designer knows that a query involving several large tables of a million rows or more is very slow or will not return results at all.

- In a decision support context, Database professionals would consider it laughable to let end users construct their own queries in large databases.

- In these models, a join is processed in three steps:
  - Product
  - select
  - project
THE COMPLEXITY OF ANALYSIS

Consider a typical business analysis problem:

• Find the share of total sales represented by each product in different markets, categories, and periods, compared with the same period a year ago.

   To do so, you would calculate the percentage each number is of the total of its column, a simple and common concept.

   However, in a classic relational database these calculations and display would require definition of a separate view, requiring over 20 SQL commands.

SURELY THE DIFFERENT PATHWAYS WILL GIVE THE SAME ANSWER

• Many RDBMSs can join only two tables at a time.
• But different pathways will give the same answer?
• This is one of those relational-database myths.
• Unfortunately, it does matter which pathway is chosen.
• In a relational database we must build a "data bridge" between tables in order to link data elements in remote tables.
• These data bridges are almost always implemented as inner joins bridging from data element to data element.
• Thus, in general, each pathway gives a different answer.

PAIRWISE JOIN PROBLEM

• If a complex join involves more than two tables, the RDBMS needs to break the query into a series of pairwise joins
• the order in which the joins are done dramatically affects query performance.
• the number of combinations to be evaluated grows exponentially with the number of tables being joined, the problem of selecting the best order of pairwise joins rarely can be solved in a reasonable amount of time.

NUMBER OF POSSIBILITIES

• The number of ways to pairwise join a set of N tables is N! (N factorial).
   – a five-table query has 5! = (6 x 5 x 4 x 3 x 2 x 1) = 120 combinations.
   – a 10-table query, there would be 3,628,800 combinations.
• A typical RDBMS will decide the order in which to do the pairwise joins before the query begins to execute, thus delaying the execution of a query even further.
• Because the number of pairwise join combinations is often too large to fully evaluate, many RDBMS optimizers limit the selection on the basis of a particular criterion, often by picking combinations of tables that are directly related.
EXAMPLE OF QUERY

- Find all the product brands that were sold in the first quarter of 1995 and present the total dollar sales as well as the number of units.

<table>
<thead>
<tr>
<th>BRAND</th>
<th>$SALES</th>
<th>UNIT SALES</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBM</td>
<td>$7,800,000</td>
<td>263</td>
</tr>
<tr>
<td>ORACLE</td>
<td>$10,440,000</td>
<td>509</td>
</tr>
<tr>
<td>BRIO</td>
<td>$2,130,000</td>
<td>444</td>
</tr>
<tr>
<td>MICROSTRATEGY</td>
<td>$1,950,000</td>
<td>392</td>
</tr>
</tbody>
</table>

SQL command for the example query

```sql
select p.brand, sum(f.dollars), sum(f.units)
from salesfact f, product p, time t
where f.productkey = p.productkey
and f.timekey = t.timekey
and t.quarter = '1 Q 1995'
group by p.brand
order by p.brand
```

Decision Support Analysis

An Operational Query may ask:
- “How much revenue did the new product generate?”

A Decision support Query may ask:
- “How much revenue did the new product generate by month, in the northeastern division, broken down by user demographic, by sales office, relative to the previous version of the product, compared with the plan?”
- A six-dimensional question.

Is there a better way of doing this?

- Use a structure that matches the end-user’s decision support needs
- One way to answer this kind of multidimensional queries is to look at the data using a multidimensional data model.
- The multidimensional view of data that is expressed using relational database semantics is provided by the database design model called dimensional model.
- The design schema used for this model is referred to as a star schema
Dimensional Model

- The basic premise of star schemas is that information can be classified into two groups: facts and dimensions.
- Facts are the core data element being analyzed.
  - For example, units of individual items sold are facts.
- Dimensions are attributes about the facts.
  - For example, dimensions are the product types purchased and the date of purchase.
- A star schema database design has a fixed structure that has no alternative join paths. This greatly simplifies the optimization and evaluation of queries on these schemas.

“We Sell Products in various markets and we measure our performance over time.”

Star Schema

- Asking the business question against this schema is much more straightforward because we are looking up specific facts through a set of dimensions.
- In the typical star schema, the fact table is much larger than any of its dimension tables.
- Unlike the entity relation model, the dimensional model is very asymmetric.
- There is one large dominant table in the center of the schema. It is the only table in the schema with multiple joins connecting it to other tables.
- The other tables all have only a single join attaching them to the central table.
SNOWFLAKE SCHEMA

• The alternative to using the level indicator is the snowflake schema.
• The essential difference is that the dimensional tables are not ‘collapsed’ but are maintained in normalized form.
• The primary motivation for this approach is performance.
• It is used when there are a large number of categories and the dimensions are large.

SNOWFLAKE SCHEMA

• The snowflake schema creates tables for attributes within a dimension table.
• In this schema, aggregate fact tables are created separately from detail tables.
• In addition to the main fact tables the snowflake schema contains separate fact tables for each level of aggregation.
• The downside is that more joins will be needed to execute a query and so performance may be adversely impacted.

TYPES OF QUERIES SUPPORTED BY STAR SCHEMA

Browse queries:
• operate on only one of the dimension tables and do not involve joins. A typical browse query occurs when the user asks for a pull-down list of all the brand names in the product dimension table, perhaps subject to constraints on other elements in the dimension table. This query must respond instantly because the user’s full attention is on the screen.

TYPES OF QUERIES SUPPORTED BY STAR SCHEMA

Multi-table join queries:
• involve constraints placed on several of the dimension tables that are all joined to the fact table simultaneously.
• The goal is to fetch hundreds or possibly thousands of underlying records into a small answer set for the user, grouped together by one or more textual attributes selected from the dimension tables. This second kind of query is rarely instantaneous, because of the significant resources required to satisfy the query.
HOW DOES IT WORK?

The user begins by placing application constraints on the dimensions through the process of browsing the dimension tables one at a time.

The browse queries are always on single-dimension tables and are usually fast acting.

Browsing is for information purposes only to allow the user to assemble the correct constraints on each dimension.

The user may launch many browse queries during this phase.

HOW DOES IT WORK?

The user also drags row headers from the dimension tables and additive facts from the fact table to the answer set staging area (the report).

The user then launches the multitable join. The DBMS groups and summarizes hundreds, thousands, or even millions of low-level records from the fact table into the small answer set and returns the answer set to the user.

Drilling down in a data warehouse is nothing more than adding headers from the dimension tables.

Drilling up is subtracting row headers. An explicit hierarchy is not needed to support drilling down.

THE FACT TABLE

- The fact table is where the numerical measurements of the business are stored.
- Each of these measurements is taken at the intersection of all the dimensions.
  - We can imagine standing out in the marketplace watching products being sold. We write down the number of dollars, the number of units, and the extended cost each day, in each market, and for each product.
- The fact tables are presummarized and aggregated along business dimensions, these tables tend to be very large.

THE FACT TABLE

- A fact table Always has a composite key
- It is highly normalized
- if there is no activities on a given day, in a market, we leave the record out of the database. It is very important that we do not try to fill the fact table zeros representing "nothing happening." For this reason,
  - Fact tables can be very sparse. Most fact tables are extremely sparse.
THE FACT TABLE

- The best and most useful facts are numeric, continuously valued, and additive. This is the holy grail of dimensional database design.
- There are facts that are semiadditive, and facts that are nonadditive.
- Semiadditive facts can be added along only some of the dimensions, and nonadditive facts simply can’t be added at all.
- To produce user answers, large number of records may be compressed into a few dozen rows of the user’s answer set.

THE FACT TABLE

- The reason for preference for numeric, continuously valued, additive facts is that in virtually every query made against fact table we are going to ask for hundreds, thousands, or even millions of records to be used by the DBMS to construct the answer set.
- In most cases, the only useful way to compress these records into the answer set is to add them. Thus if the measurements are numbers and if they are additive, we can easily build the answer set.
- For nonadditive facts, we are forced to use counts if we wish to summarize the records, or we are reduced to printing out the fact records one at a time.

Fact Data

- Numerical measures of the business
- Accessed by dimensions
- Point-in-time data snapshots
- Element of time and dates
- Multipart primary key
- Indexed primary keys
- Non-indexed columns
- Many fact tables
- Avoid reorganizing facts

Fact Data

- Contains derived data

- Monthly.Unit_Sales
  - Average_Selling_Price
    - Unit_Price
    - Unit_Sales
    - January.Unit_Sales
    - January.Unit_Sales
    - January.Unit_Sales
    - January.Unit_Sales
    - January.Unit_Sales
  + January.Unit_Sales
Fact Data Tables

- Tables can be large.
- Data is introduced according to refresh cycles.
- Data is date stamped.
- Data allows navigation through history.

Database Keys

- Single Column Time Key
- Single Column Product Key
- Composite Key
- Single Column Store Key

Database Keys and Indexes

- Primary keys on fact and dimension table columns
- Foreign keys on fact table columns
- Indexed for speed
- Primary keys may be maintained in a
  - Composite index
  - Single column index
- Indexes may be ignored
- Index only queries possible
- Keys identified early during design
- Generalized keys may be employed

Granularity

- Affect on warehouse
  - Size of the warehouse database
  - Degree of analysis
  - Flexibility
- Level of detail of the data
  - Individual transactions
  - Daily snapshots
  - Monthly snapshots
  - Yearly snapshots
  - Any other time period
Granularity

• High Level (fine grain)
  Details of customer bank transactions per month

• Low Level (coarse grain)
  Summary of customer bank transactions per month

Drive level from business need

Granularity

• Analyze requirement to avoid lost information
• Review, revise, and realign
• Resolve issues
  – Time
  – Experience
  – Proof of concept
• Consider effect on DASD
• Design grain one level lower than specified

Fact Table Attributes

Additive
Add across all dimensions

Semi-additive
Add along some dimensions

Non-additive
Cannot be added along a dimension

Fact Table Attributes

Additive
Sales Fact
Time_key
Product_key
Store_key
Promotion_key
Quantity_sold
Revenue*
Cost*
Customer Count

Semi-Additive
Cost by Product + Store + Time
Cost by Product + Promotion + Time
Revenue by Product + Promotion + Time
Revenue by Product + Promotion + Store + Time . . .

Cost by Product + Store + Time

Store Dimension

Product Dimension

Promotion Dimension
THE DIMENSION TABLES

• The dimension tables are where the textual descriptions of the dimensions of the business are stored.
• Each of the textual descriptions helps to describe a member of the respective dimension.
  – For example, each record in the product dimension represents a specific product.
• In a well-designed database, each dimension table has many attributes (fields).
• The best attributes are textual, discrete, and used as the source of constraints and row headers in the user’s answer set.

A dimension table can NOT have a composite key.
• A key role for dimension table attributes is to serve as the source of constraints in a query or to serve as row headers in the user’s answer set.
• A non-normalized flat table
• If the dimension tables are normalized, we have Snow-Flake Schema.

Dimension Data

• Dimension data qualifies and drives user query constraints.
• Design is imperative.
• Dimension data is linked to fact data by keys.

Data must be of good quality.
• Data is often expanded for the warehouse.

<table>
<thead>
<tr>
<th>Operational Database</th>
<th>Warehouse Database</th>
</tr>
</thead>
<tbody>
<tr>
<td>E Dress</td>
<td>R Evening Dress Red</td>
</tr>
<tr>
<td>S Tie</td>
<td>MC Silk Tie Multi</td>
</tr>
<tr>
<td>S Jkt</td>
<td>BnX Sports Jacket Brown Check</td>
</tr>
</tbody>
</table>

• Dimension data is changed not refreshed
Dimension Data Tables

- Textual data
- Smaller volumes
- Discrete values
- Quality data is important

- Customer
- Sales
- Suppliers
- Time
- Items
- Products

Normalization

- Normalized data contains no
  - Redundancy.
  - Repeating data.
  - Key independent columns.
- Denormalized data often
  - Improves efficiency in OLAP systems.
  - Exists in data warehouse databases.
  - Comprises derived or summary data.
- Star and snowflake models are denormalized.

Facts or Dimensions

- Units Sold - Calls in a Month - Color
- Store on dimension table if the attribute is perceived as a constant or discrete value:
  - Description
  - Location
  - Color
  - Size
- Store on fact table if the attribute varies continuously:
  - Balance
  - Units Sold
  - Calls per Month

- Product Cost
The Time Dimension

- Operational systems present an up-to-date snapshot.
- Warehouses offer a time series.

<table>
<thead>
<tr>
<th>Time Dimension</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Dimension Table</td>
</tr>
<tr>
<td>- Workday</td>
</tr>
<tr>
<td>- Fiscal period</td>
</tr>
<tr>
<td>- Major event</td>
</tr>
<tr>
<td>- Month</td>
</tr>
<tr>
<td>- Holiday</td>
</tr>
<tr>
<td>- Flexible analysis</td>
</tr>
</tbody>
</table>

Reference Data and Tables

- Supports management of dimension data
- Reduces warehouse volume
- Provides lookup for encoded data

<table>
<thead>
<tr>
<th>Sales Fact Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item_id</td>
</tr>
<tr>
<td>Store_id</td>
</tr>
<tr>
<td>Sales_dollars</td>
</tr>
<tr>
<td>Sales_units</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Week_id</td>
</tr>
<tr>
<td>Period_id</td>
</tr>
<tr>
<td>Year_id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item_id</td>
</tr>
<tr>
<td>Dept_id</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item Lookup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Item_id</td>
</tr>
<tr>
<td>Item_desc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Totsales Summary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Month_id</td>
</tr>
<tr>
<td>Store_id</td>
</tr>
<tr>
<td>Item_id</td>
</tr>
<tr>
<td>Total_dollars</td>
</tr>
</tbody>
</table>

Summary Data

- Provide fast access to precomputed data
- Reduce use of
  - I/O
  - CPU
  - Memory
- Distill from
  - Source systems - lightly summarized
  - Precalculated summaries - highly summarized
- Determine requirements early
Summary Data

- Average
- Maximum
- Total
- Percentage

<table>
<thead>
<tr>
<th>Dimension Data</th>
<th>Fact Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Units Sold</td>
<td>Sales($)</td>
</tr>
<tr>
<td>Product A</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>Product B</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
<tr>
<td>Product C</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
</tr>
</tbody>
</table>

Summary Data Tables

- Important design consideration
- Based on facts calculated by dimension data
- Usually exist in summary fact tables
- May be many hundreds
- Some tools are not summary table aware

Summary Data Tables

- Cumulative summary
  | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 | Day 7 | Day 8 |
- Rolling summary
  | Day 1 | Day 2 | Day 3 | Day 4 | Day 5 | Day 6 | Day 7 | Week 1
  | Day 8 | Day 9 | Day 10| Day 11| Day 12| Day 13| Day 14| Week 2 |

Separate Summary Fact Tables
Summary Records and Level Fields

- Fact and summary data reside in a single table.
- Not all tools can use this approach.
- New summary tables are easier to manage.

Summarization Levels

- Determine levels
- Define a strategy

Summary Matrix

- One summary for individual requirements
- Existing summaries rolled up

Country | X | X | X
State   | X | X | X
District| X | X | X
Office  | X | X | X
Salesperson | X | X | X

Country by year could be a roll up of State, District, Office, or Salesperson

Metadata

- Vital to the warehouse
- Data about data
- Used by everyone

- ETT metadata - physical design, sources, mapping rules
- User metadata - navigation aid, business information, rules
- Operational metadata - scheduling, analysis
Convert ER to DM

If you already have an ER based model for your data, you can convert it into a DM model using the following three steps:

1. Separate the ER diagram into its discrete business processes and model each one separately as a DM.

2. Select those many-to-many relationships in the ER model containing numeric and additive non-key facts and to designate them as fact tables.

3. Denormalize all of the remaining tables into flat tables with single-part keys that connect directly to fact tables. These tables will become your dimension tables.

STEPS IN THE DESIGN OF A DATA WAREHOUSE USING DIMENSIONAL MODELING

1. Choose a business process to model.
   - A business process is a major operational process in your organization that is supported by some kind of legacy system (or systems) from which data can be collected for the purposes of the data warehouse. Examples of business processes are orders, invoices, shipments, inventory, account administration, sales, and the general ledger.

2. Choose the grain of the business process.
   - The grain is the fundamental, atomic level of data to be represented in the fact table for this process.
   - Typical grains are individual transactions, individual daily snapshots, or individual monthly snapshots. It is impossible to proceed to step 3 without defining the grain.

3. Choose the dimensions that will apply to each fact table record.
   - Typical dimensions are time, product, customer, promotion, warehouse, transaction type, and status.
   - With the choice of each dimension, describe all discrete, text like dimensional attributes (fields) that fill out each dimension table.

4. Choose the measured facts that will populate each fact table record.
   - Typical measured facts are numeric additive quantities like Quantity Sold and Dollars Sold.

NINE DECISIONS IN THE DESIGN OF DATA WAREHOUSE

(Using a dimensional modeling approach)

Choose the subject matter
Decide what a fact table represents
Identifying and conforming the dimensions
Choosing the facts
Storing precalculations in the fact table
Rounding out the dimension tables
Choosing the duration of the database
The need to track slowly changing dimensions
Deciding the query priorities and the query modes
POTENTIAL PERFORMANCE PROBLEMS WITH STAR SCHEMES

• A dimensional database design has a fixed structure that has no alternative join paths.
• This greatly simplifies the optimization and evaluation of queries on these schemas.
• But, it may limit the kind of ad hoc queries possible.
• Also, a star schema requires multiple metadata definitions (one for each key component) to define a single relationship (table); this adds to the design complexity, and sluggishness in performance.

PAIRWISE JOIN PROBLEM REVISITED

• The only table directly related to most other tables is the fact table.
• This means that the fact table is a natural candidate for the first pairwise join.
• Unfortunately, the fact table is typically the very largest table in the query, so this strategy invariably leads to selecting a pairwise join order that generates a very large intermediate result set.
• Usually, this generates large intermediate result sets severely that can affect query performance.

LEVEL INDICATOR

• Another potential problem with the star schema design is that in order to navigate the dimensions successfully, the dimensional table design often includes a level of hierarchy indicator for every record.
• Every query that is retrieving detail records from a table that stores details and aggregates must use this indicator as an additional constraint to obtain a correct result.
• The level is a useful tool for the environments that are tightly controlled by the DBA and DA staff, and very few ad hoc queries are allowed.
• If the user is not aware of the level indicator, or its values are incorrect, the otherwise valid query may result in a totally invalid answer.

POTENTIAL PERFORMANCE PROBLEMS WITH STAR SCHEMES

• Since the fact table must carry all key components as part of its primary key, addition or deletion of levels in the hierarchy in the dimension tables may require physical modification of the affected table, which is a time-consuming process that limits flexibility.
• Carrying all the segments of the compound dimensional key in the fact table increases the size of the index, thus impacting both performance and scalability.
SOLUTIONS TO PERFORMANCE PROBLEMS
Optimizers on order of tables to join
Parallel processing
Innovative indexing
STARjoin and STARindex approach by Red Brick
• STARindex is an index on the multikey foreign key columns of the Fact table.
• STARindex is used in STARjoin to perform a high speed, single pass multitable joins.

THE GROCERY STORE CHAIN DATA WAREHOUSE EXAMPLE
Each of the stores has typical departments including grocery, frozen foods, dairy, meat, produce, bakery, floral, hard goods, liquor, and drugs.
Each store has roughly 60,000 individual products on its shelves.
The individual products are called stock keeping units, or SKUs.

THE GROCERY STORE CHAIN DATA WAREHOUSE EXAMPLE
• About 40,000 of the SKUs come from outside manufacturers and have bar codes imprinted on the product package.
• These bar codes are called Universal Product Codes, or UPCs. UPCs are at the same grain as individual SKUs.
• Each different package variation of a product has a separate UPC and hence is a separate SKU.
• The remaining 20,000 SKUs come from departments like the meat, produce, bakery, or floral departments and don’t have nationally recognized UPC codes.

MAPPING THE DESIGN STEP TO THIS CASE
1. Identifying the process to model:
• we will build a daily item movement warehouse (what products are selling in which stores, at which price, on which days and using which promotion
2. Choose the grain:
• SKU
• by store
• by promotion
• by day

( how many dimension tables ?)
3. Choose the dimensions:

- Time Dimension:
  - The time dimension is the one dimension virtually guaranteed to be present in every data warehouse, because virtually every data warehouse is a time series.

- Store Dimension:
  - The store dimension describes every store in our grocery chain.

- Product Dimension:
  - The product dimension describes every SKU in the grocery store.

3. Choose the dimensions:

- Promotion Dimension:
  - Potentially the most interesting dimension in our schema is the promotion dimension.
  - The promotion dimension describes each promotion condition under which a product is sold in the grocery chain.
  - Promotion conditions include temporary price reductions, end aisle displays, newspaper ads, and coupons.

4. Choose the measured facts:

- Dollar sales
- Units sales
- Dollar cost
- Customer counts
ESTIMATING OF THE SIZE OF THE WAREHOUSE

- Time dimension: 2 years X 365 days = 730 days
- Store dimension: 300 stores, reporting sales each day
- Product dimension: 30,000 products in each store, of which 3,000 sell each day in a given store
- Promotion dimension: a sold item appears in only one promotion condition in a store on a day
- Number of base fact records = 730 X 300 X 3000 X 1 = 657 million records
- Number of key fields = 4; Number of fact fields = 4; Total fields = 8
- Base fact table size = 657 million X 8 fields X 4 bytes = 21 GB

Summary

- Data warehouse tables
  - Fact
  - Dimension
  - Reference
  - Summary
- Metadata

4-77