

SYLLABUS

- Logical Design for data warehouse
 - Data Warehousing Schemas
 - Star Join (Star Schema) and Snowflake structure
- Physical Design for data warehouse
 - Physical Design Structures
 - Table spaces
 - Tables and Partitioned Tables
 - Views
 - Integrity Constraints
 - Dimensions
- Design dimension table fact table for data warehouse
- Design and implement effective physical data structure for data warehouse

Table of Contents

1. Logical Design for Data Warehouse	3
2. Granularity Level	8
3. Auditing and Lineage	9
4. Physical Design Structures.....	9
5. Designing Fact Tables	12
6. Types of fact tables.....	12
7. Designing fact table steps.....	12
8. Design and implement effective physical data structure for data warehouse	15
QUESTION BANK.....	17

1. Logical Design for Data Warehouse

A DW is a centralized data silo for an enterprise that contains merged, cleansed, and historical data. DW schemas are simplified and thus more suitable for generating reports than normalized relational schemas. For a DW, you typically use a special type of logical design called a Star schema, or a variant of the Star schema called a Snowflake schema. Tables in a Star or Snowflake schema are divided into dimension tables (commonly known as dimensions) and fact tables. DW logical design seems to be simple at first glance. It is definitely much simpler than a normalized relational design. However, despite the simplicity, you can still encounter some advanced problems. In this chapter, you will learn how to design a DW and how to solve some of the common advanced design problems. You will explore **Star and Snowflake schemas, dimensions, and fact tables**. You will also learn how to track the source and time for data coming into a DW through auditing or, in DW terminology, lineage information.

1.1 Star Schemas

Before you design a data warehouse, you need to understand some common design patterns used for a DW, namely the Star and Snowflake schemas. These schemas evolved in the 1980s. In particular, the Star schema is currently so widely used that it has become a kind of informal standard for all types of business intelligence (BI) applications.

Star Schema Often, a picture is worth more than a thousand words. Figure 1 shows a Star schema, a diagram created in SSMS from a subset of the tables in the AdventureWorksDW2012 sample database.

In Figure 1, you can easily spot how the Star schema got its name—it resembles a star. There is a single central table, called a fact table, surrounded by multiple tables called dimensions. One Star schema covers one business area. In this case, the schema covers Internet sales. An enterprise data warehouse covers multiple business areas and consists of multiple Star (and/or Snowflake) schemas.

The fact table is connected to all the dimensions with foreign keys. Usually, all foreign keys taken together uniquely identify each row in the fact table, and thus collectively form a unique key, so you can use all the foreign keys as a composite primary key of the fact table. You can also add a simpler key. The fact table is on the “many” side of its relationships with the dimensions.

If you were to form a proposition from a row in a fact table, you might express it with a sentence such as, “Customer A purchased product B on date C in quantity D for amount E.”. This proposition is a fact; this is how the fact table got its name.

The Star schema evolved from a conceptual model of a cube. You can imagine all sales as a big box. When you search for a problem in sales data, you use a divide-and-conquer technique: slicing the cube over different categories of customers, products, or time. In other words, you slice the cube over its dimensions. Therefore, customers, products, and time represent the three dimensions in the conceptual model of the sales cube. Dimension tables (dimensions) got their name from this conceptual model. In a logical model of a Star schema, you can represent more than three dimensions. Therefore, a Star schema represents a multidimensional hypercube.

As you already know, a data warehouse consists of multiple Star schemas. From a business perspective, these Star schemas are connected. For example, you have the same customers in sales as in accounting. You deal with many of the same products in sales, inventory, and production. Of course, your business is performed at the same time over all the different business areas. To represent the business correctly, you must be able to

connect the multiple Star schemas in your data warehouse. The connection is simple – you use the same dimensions for each Star schema. In fact, the dimensions should be shared among multiple Star schemas.

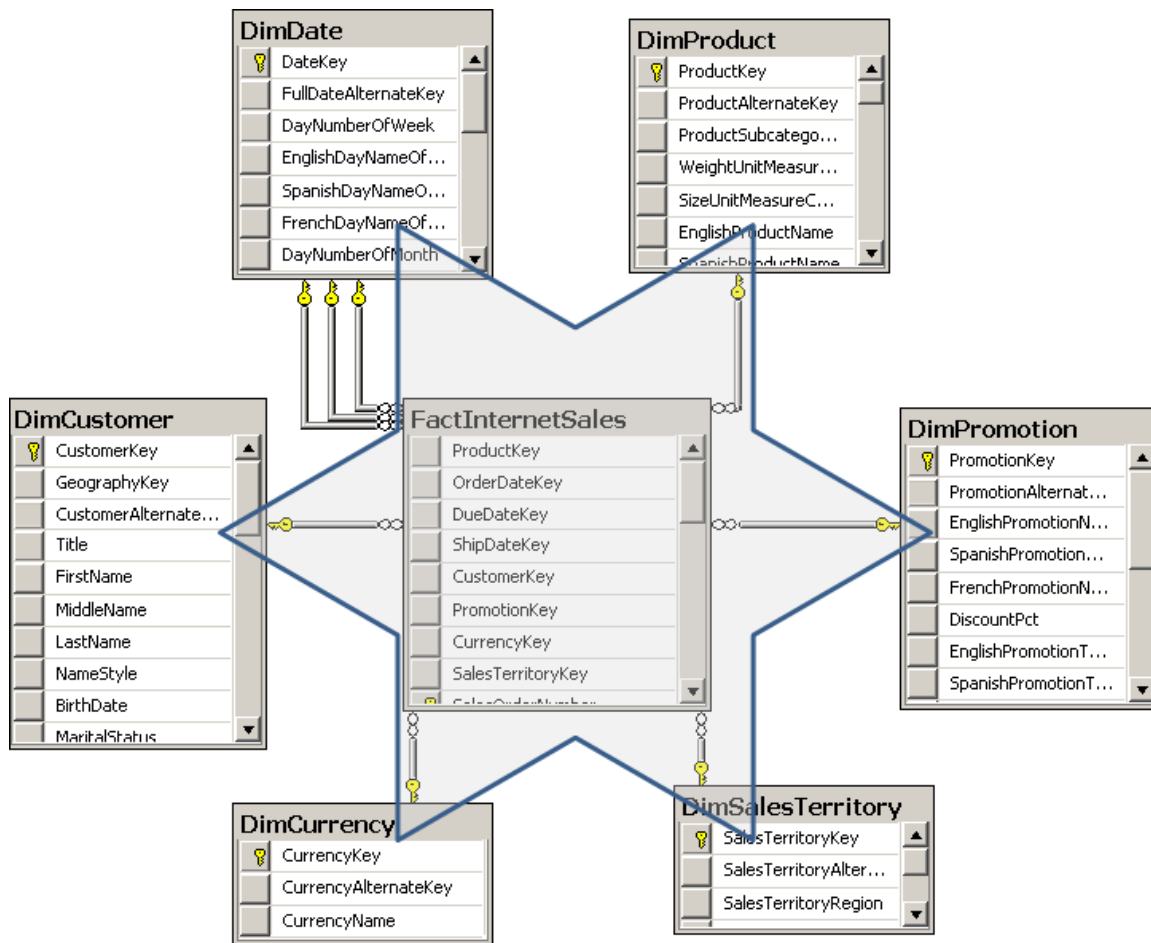


Figure 1 - A Star schema example.

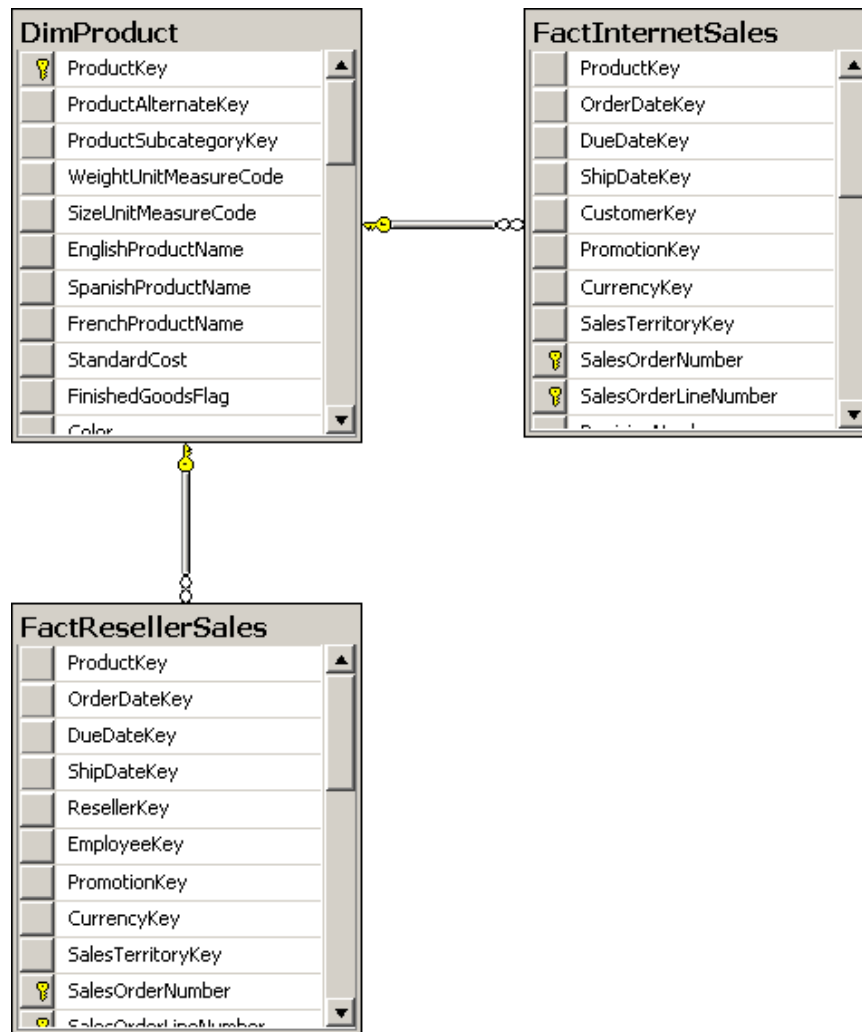
As you already know, a data warehouse consists of multiple Star schemas. From a business perspective, these Star schemas are connected. For example, you have the same customers in sales as in accounting. You deal with many of the same products in sales, inventory, and production. Of course, your business is performed at the same time over all the different business areas. To represent the business correctly, you must be able to connect the multiple Star schemas in your data warehouse. The connection is simple – you use the same dimensions for each Star schema. In fact, the dimensions should be shared among multiple Star schemas.

Dimensions have foreign key relationships with multiple fact tables. Dimensions with connections to multiple fact tables are called shared or conformed dimensions. Figure 2 shows a conformed dimension from the AdventureWorksDW2012 sample database with two different fact tables sharing the same dimension.

In the past, there was a big debate over whether to use shared or private dimensions. Private dimensions are dimensions that pertain to only a single Star schema. However, it is quite simple to design shared dimensions; you do not gain much from the design-time perspective by using private dimensions. In fact, with private dimensions, you lose the connections between the different fact tables, so you cannot compare the data in different fact tables over the same dimensions. For example, you could not compare sales and accounting data for the same customer if the sales and accounting fact tables didn't share the same customer dimension.

Therefore, unless you are creating a small proof-of-concept (POC) project that covers only a single business area where you do not care about connections with different business areas, you should always opt for shared dimensions.

Figure 2 - Dim Product is a shared dimension



A data warehouse is often the source for specialized analytical database management systems, such as SQL Server Analysis Services (SSAS). SSAS is a system that performs specialized analyses by drilling down and is used for analyses that are based on the conceptual model of a cube. Systems such as SSAS focus on a single task and fast analyses, and they are considerably more optimized for this task than general systems such as SQL Server. SSAS enables analysis in real time, a process called online analytical processing (OLAP). However, to get such performance, you have to pay a price. SSAS is out of the scope of this book, but you have to know the limitations of SSAS to prepare a data warehouse in a way that is useful for SSAS. One thing to remember is that in an SSAS database, you can use shared dimensions only. This is just one more reason why you should prefer shared to private dimensions.

1.2 Snowflake Schema

Figure 3 shows a more detailed view of the Dim Date dimension from the AdventureWorksDW2012 sample database.

The highlighted attributes show that the dimension is denormalized. It is not in third normal form. In third normal form, all non-key columns should not transitively depend on the key. A different way to say this is that there should be no functional dependency between non-key columns. You should be able to retrieve the value of a non-key column only if you know the key. However, in the Dim Date dimension, if you know the month, you obviously know the calendar quarter, and if you know the calendar quarter, you know the calendar semester.

In a Star schema, dimensions are denormalized. In contrast, in an LOB normalized schema, you would split the table into multiple tables if you found a dependency between non-key columns. Figure 4 shows such a normalized example for the Dim Product, Dim Product Subcategory and Dim Product Category tables from the AdventureWorksDW2012 database.




















DimDate	
	DateKey
	FullDateAlternateKey
	DayNumberOfWeek
	EnglishDayNameOfWeek
	SpanishDayNameOfWeek
	FrenchDayNameOfWeek
	DayNumberOfMonth
	DayNumberOfYear
	WeekNumberOfYear
	EnglishMonthName
	SpanishMonthName
	FrenchMonthName
	MonthNumberOfYear
	CalendarQuarter
	CalendarYear
	CalendarSemester
	FiscalQuarter
	FiscalYear
	FiscalSemester

Figure 3 - The Dim Date denormalized dimension

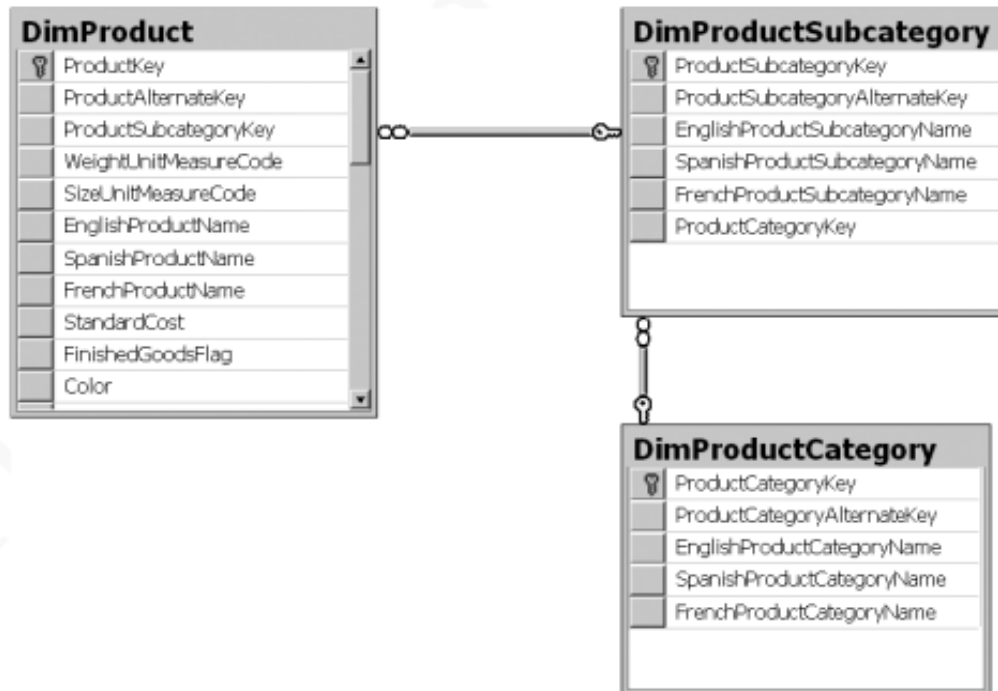


Figure 4 - The Dim Product normalized dimension.

The Dim Product dimension is not denormalized. The Dim Product table does not contain the subcategory name, only the Product Subcategory Key value for the foreign key to the Dim Product Subcategory lookup table. Similarly, the Dim Product Subcategory table does not contain a category name; it just holds the foreign key Product Category Key from the Dim Product Category table. This design is typical of an LOB database schema.

You can imagine multiple dimensions designed in a similar normalized way, with a central fact table connected by foreign keys to dimension tables, which are connected with foreign keys to lookup tables, which are connected with foreign keys to second-level lookup tables.

In this configuration, a star starts to resemble a snowflake. Therefore, a Star schema with normalized dimensions is called a Snowflake schema.

In most long-term projects, you should design Star schemas. Because the Star schema is simpler than a Snowflake schema, it is also easier to maintain. Queries on a Star schema are simpler and faster than queries on a Snowflake schema, because they involve fewer joins. The Snowflake schema is more appropriate for short POC projects, because it is closer to an LOB normalized relational schema and thus requires less work to build. In some cases, you can also employ a hybrid approach, using a Snowflake schema only for the first level of a dimension lookup table. In this type of approach, there are no additional levels of lookup tables; the first-level lookup table is denormalized. Figure 5 shows such a partially denormalized schema.

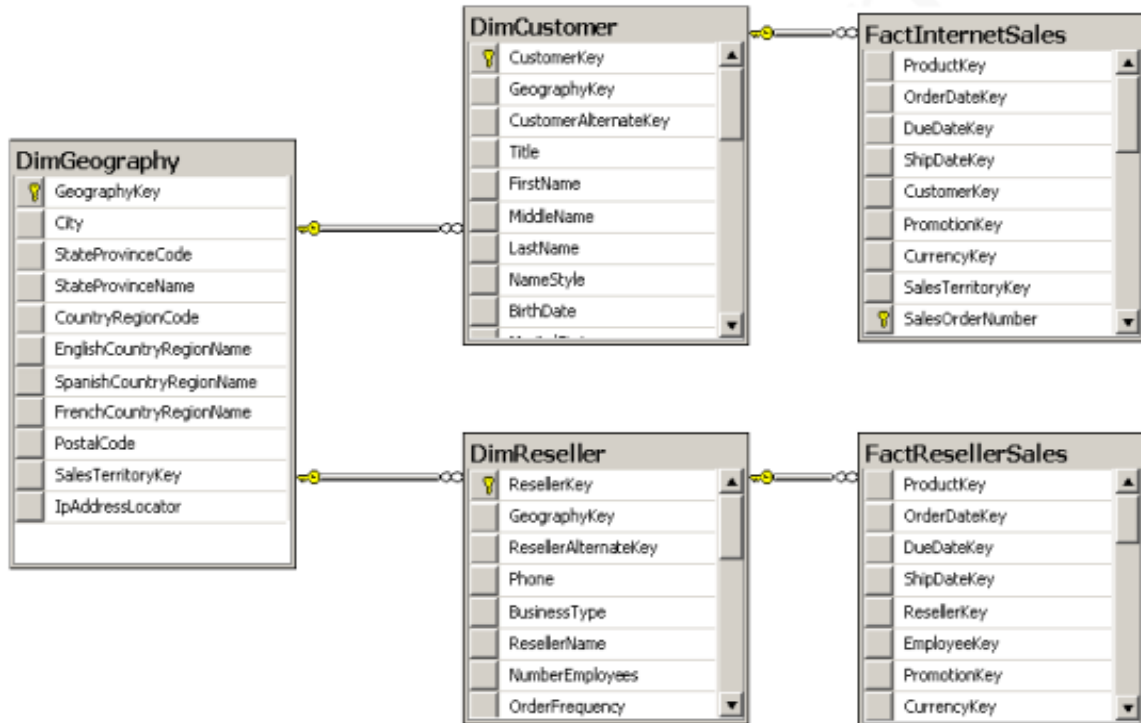


Figure 5 - Partially denormalized dimensions

In Figure 5, the Dim Customer and Dim Reseller dimensions are partially normalized. The dimensions now contain only the Geography Key foreign key. However, the Dim Geography table is denormalized. There is no additional lookup table even though a city is in a region and a region is in a country. A hybrid design such as this means that geography data is written only once and needs to be maintained in only a single place. Such a design is appropriate when multiple dimensions share the same attributes. In other cases, you should use the simpler Star schema. To repeat: you should use a Snowflake schema only for quick POC projects.

2. Granularity Level

The number of dimensions connected with a fact table defines the level of granularity of analysis you can get. For example, if no products dimension is connected to a sales fact table, you cannot get a report at the product level—you could get a report for sales for all products only. This kind of granularity is also called the dimensionality of a Star schema.

But there is another kind of granularity, which lets you know what level of information a dimension foreign key represents in a fact table. Different fact tables can have different granularity in a connection to the same dimension. This is very typical in budgeting and planning scenarios. For example, you do not plan that customer A will come on date B to store C and buy product D for amount E. Instead, you plan on a higher level—you might plan to sell amount E of products C in quarter B in all stores in that region to all customers in that region. Figure 6 shows an example of a fact table that uses a higher level of granularity than the fact tables introduced so far.

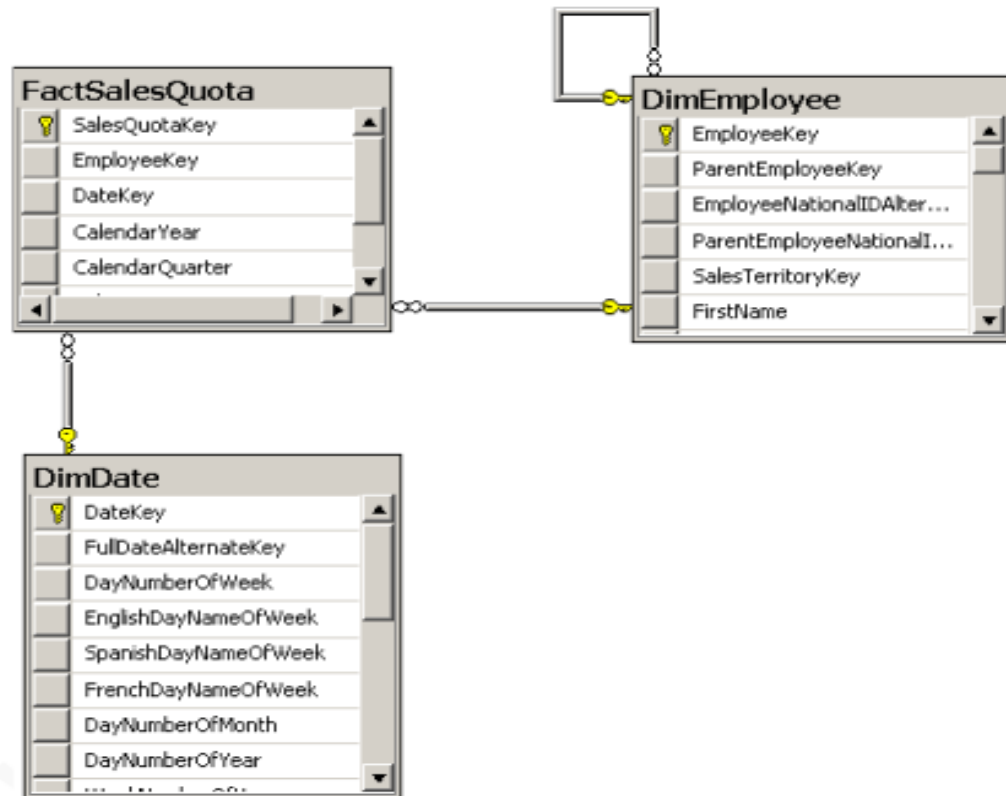


Figure 6 - A fact table with a higher level of granularity.

3. Auditing and Lineage

In addition to tables for reports, a data warehouse may also include auditing tables. For every update, you should audit who made the update, when it was made, and how many rows were transferred to each dimension and fact table in your DW. If you also audit how much time was needed for each load, you can calculate the performance and take action if it deteriorates.

You store this information in an auditing table or tables. However, you should realize that auditing does not help you unless you analyze the information regularly.

Auditing tables hold batch-level information about regular DW loads, but you might also want or need to have more detailed information. For example, you might want to know where each row in a dimension and/or fact table came from and when it was added. In such cases, you must add appropriate columns to the dimension and fact tables. Such detailed auditing information is also called lineage in DW terminology. To collect either auditing or lineage information, you need to modify the extract-transform load (ETL) process you use for DW loads appropriately.

4. Physical Design Structures

Once you have converted your logical design to a physical one, you must create some or all of the following

structures:

- Table spaces
- Tables and Partitioned Tables
- Views
- Integrity Constraints
- Dimensions

4.1 Table spaces

A table space consists of one or more data files, which are physical structures within the operating system you are using. A data file is associated with only one table space. From a design perspective, table spaces are containers for physical design structures.

Table spaces need to be separated by differences. For example, tables should be separated from their indexes and small tables should be separated from large tables. Table spaces should also represent logical business units if possible. Because a table space is the coarsest granularity for backup and recovery or the transportable table spaces mechanism, the logical business design affects availability and maintenance operations.

You can now use ultra large data files, a significant improvement in very large databases. Tables and Partitioned

Tables

Tables are the basic unit of data storage. They are the container for the expected amount of raw data in your data warehouse.

Using partitioned tables instead of no partitioned ones addresses the key problem of supporting very large data volumes by allowing you to divide them into smaller and more manageable pieces. The main design criterion for partitioning is manageability, though you also see performance benefits in most cases because of partition pruning or intelligent parallel processing. For example, you might choose a partitioning strategy based on a sales transaction date and a monthly granularity. If you have four years' worth of data, you can delete a month's data as it becomes older than four years with a single, fast DDL statement and load new data while only affecting 1/48th of the complete table. Business questions regarding the last quarter only affect three months, which is equivalent to three partitions, or 3/48ths of the total volume.

Partitioning large tables improves performance because each partitioned piece is more manageable. Typically, you partition based on transaction dates in a data warehouse. For example, each month, one month's worth of data can be assigned its own partition.

4.2 Table Compression

You can save disk space, increase memory efficiency, and improve query performance by compressing heap-organized tables. This often leads to better scalability and query performance. You can enable compression at the table space, table, or partition level. A typical type of heap-organized table you should consider for table compression is partitioned tables. Although compressed tables or partitions are updatable, there is some overhead in updating these tables, and high update activity may work against compression by causing some space to be wasted.

OLTP table compression is best suited for tables with significant update activity. Hybrid Columnar

Compression, a feature of certain Oracle storage systems, utilizes a combination of both row and columnar methods for storing data. When data is loaded, groups of rows are stored in columnar format, with the values for a given column stored and compressed together. Storing column data together, with the same data type and similar characteristics, drastically increases the storage savings achieved from compression. Hybrid Columnar Compression provides multiple levels of compression and is best suited for tables or partitions with minimal update activity.

4.3 Views

A view is a tailored presentation of the data contained in one or more tables or other views. A view takes the output of a query and treats it as a table. Views do not require any space in the database.

Integrity Constraints

Integrity constraints are used to enforce business rules associated with your database and to prevent having invalid information in the tables. Integrity constraints in data warehousing differ from constraints in OLTP environments. In OLTP environments, they primarily prevent the insertion of invalid data into a record, which is not a big problem in data warehousing environments because accuracy has already been guaranteed. In data warehousing environments, constraints are only used for query rewrite. NOT NULL constraints are particularly common in data warehouses. Under some specific circumstances, constraints need space in the database. These constraints are in the form of the underlying unique index.

4.4 Indexes and Partitioned Indexes

Indexes are optional structures associated with tables or clusters. In addition to the classical B-tree indexes, bitmap indexes are very common in data warehousing environments. Bitmap indexes are optimized index structures for set-oriented operations. Additionally, they are necessary for some optimized data access methods such as star transformations.

Indexes are just like tables in that you can partition them, although the partitioning strategy is not dependent upon the table structure. Partitioning indexes makes it easier to manage the data warehouse during refresh and improves query performance.

Materialized Views

Materialized views are query results that have been stored in advance so long-running calculations are not necessary when you actually execute your SQL statements. From a physical design point of view, materialized views resemble tables or partitioned tables and behave like indexes in that they are used transparently and improve performance.

4.5 Dimensions

A dimension is a schema object that defines hierarchical relationships between columns or column sets. A hierarchical relationship is a functional dependency from one level of a hierarchy to the next one. A dimension is a container of logical relationships and does not require any space in the database. A typical dimension is city, state (or province), region, and country.

5. Designing Fact Tables

Fact tables, like dimensions, have specific types of columns that limit the actions that can be taken with them. Queries from a DW aggregate data; depending on the particular type of column, there are some limitations on which aggregate functions you can use. Many-to-many relationships in a DW can be implemented differently than in a normalized relational schema.

6. Types of fact tables

All fact tables are categorized by the three most basic measurement events:

- **Transactional** – Transactional fact table is the most basic one that each grain associated with it indicated as “one row per line in a transaction”, e.g., every line item appears on an invoice. Transaction fact table stores data of the most detailed level, therefore, it has a high number of dimensions associated with it.
- **Periodic snapshots** – Periodic snapshots fact table stores the data that is a snapshot in a period of time. The source data of the periodic snapshots fact table is data from a transaction fact table where you choose a period to get the output.
- **Accumulating snapshots** – The accumulating snapshots fact table describes the activity of a business process that has a clear beginning and end. This type of fact table, therefore, has multiple date columns to represent milestones in the process. A good example of accumulating snapshots fact table is the processing of a material. As steps towards handling the material are finished, the corresponding record in the accumulating snapshots fact table gets updated.

7. Designing fact table steps

Here is an overview of four steps to designing a fact table described by Kimball:

1. **Choosing business process to a model** – The first step is to decide what business process to model by gathering and understanding business needs and available data
2. **Declare the grain** – by declaring a grain means describing exactly what a fact table record represents
3. **Choose the dimensions** – once the grain of the fact table is stated clearly, it is time to determine dimensions for the fact table.
4. **Identify facts** – identify carefully which facts will appear in the fact table.

7.1 Fact Table Column Types:

Fact tables are collections of measurements associated with a specific business process. You store measurements in columns. Logically, this type of column is called a measure. Measures are the essence of a fact table. They are usually numeric and can be aggregated. They store values that are of interest to the business, such as sales amount, order quantity, and discount amount.

A fact table includes foreign keys from all dimensions. These foreign keys are the second type of column in a fact table. A fact table is on the “many” side of the relationships with dimensions. All foreign keys together usually uniquely identify each row and can be used as a composite primary key.

You often include an additional surrogate key. This key is shorter and consists of one or two columns only. The surrogate key is usually the business key from the table that was used as the primary source for the fact table. For example, suppose you start building a sales fact table from an order details table in a source system,

and then add foreign keys that pertain to the order as a whole from the Order Header table in the source system. Tables 1, 2, and 3 illustrate an example of such a design process.

Table 1 shows a simplified example of an Orders Header source table. The Order ID column is the primary key for this table. The Customer ID column is a foreign key from the Customers table. The Order Date column is not a foreign key in the source table; however, it becomes a foreign key in the DW fact table, for the relationship with the explicit date dimension. Note, however, that foreign keys in a fact table can—and usually are—replaced with DW surrogate keys of DW dimensions.

Orderid	Customerid	Orderdate
12541	17	2012/02/21

TABLE 1 - The Source Orders Header Table

Table 2 shows the source Order Details table. The primary key of this table is a composite one and consists of the Order ID and Line Item Id columns. In addition, the source Order Details table has the Product ID foreign key column. The Quantity column is the measure.

Orderid	Linitemid	Productid	Quantity
12541	2	5	47

TABLE 2 - The Source Order Details Table

Table 3 shows the Sales Fact table created from the Orders Header and Order Details source tables. The Order Details table was the primary source for this fact table. The Order ID, Line Item Id, and Quantity columns are simply transferred from the source Order Details table.

The Product ID column from the source Order Details table is replaced with a surrogate DW Product Key column. The Customer ID and Order Date columns take the source Orders Header table; these columns pertain to orders, not order details. However, in the fact table, they are replaced with the surrogate DW keys Customer Key and Order Date Key.

Orderid	Linitemid	CustomerKey	OrderDateKey	ProductKey	Quantity
12541	2	289	444	25	47

TABLE 3 - The Sales Fact Table

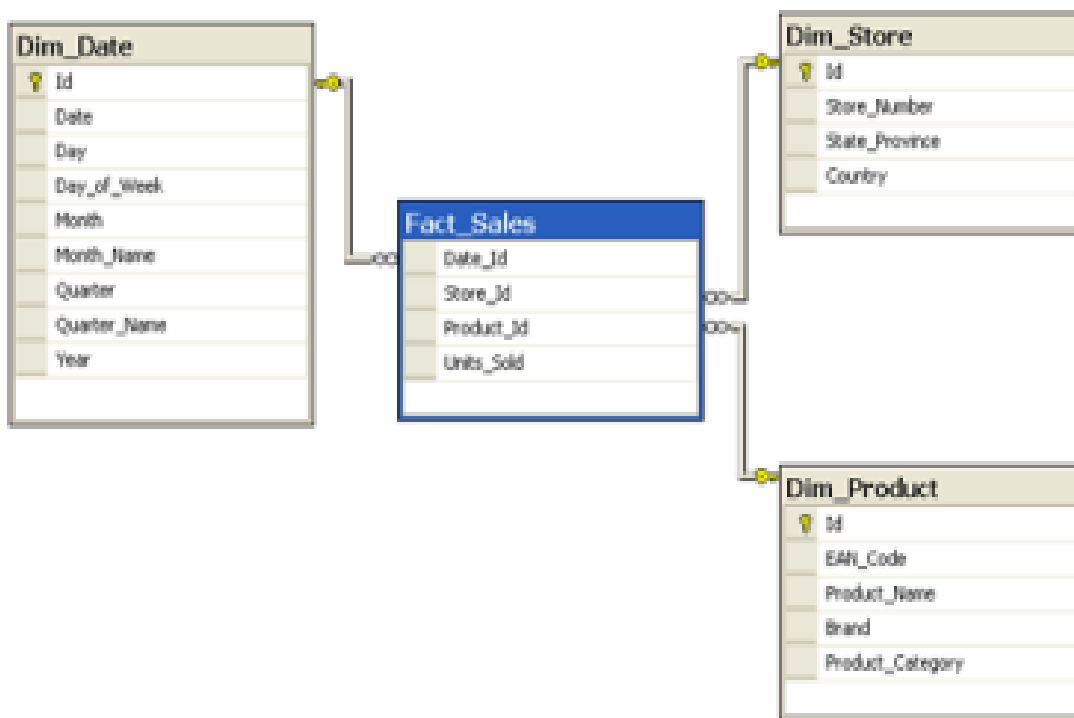
You do not need the Order ID and Line Item Id columns in this sales fact table. For analyses, you could create a composite primary key from the Customer Key, Order Date Key, and Product-Key columns. However, you should keep the Order ID and Line Item Id columns to make quick controls and comparisons with source data possible. In addition, if you were to use them as the primary key, then the primary key would be shorter than one composed from all foreign keys. The last column type used in a fact table is the lineage type, if you implement the lineage. Just as with dimensions, you never expose the lineage information to end users. To recapitulate, fact tables have the following column types:

- Foreign keys
- Measures
- Lineage columns (optional)
- Business key columns from the primary source table (optional)

An example of a fact table

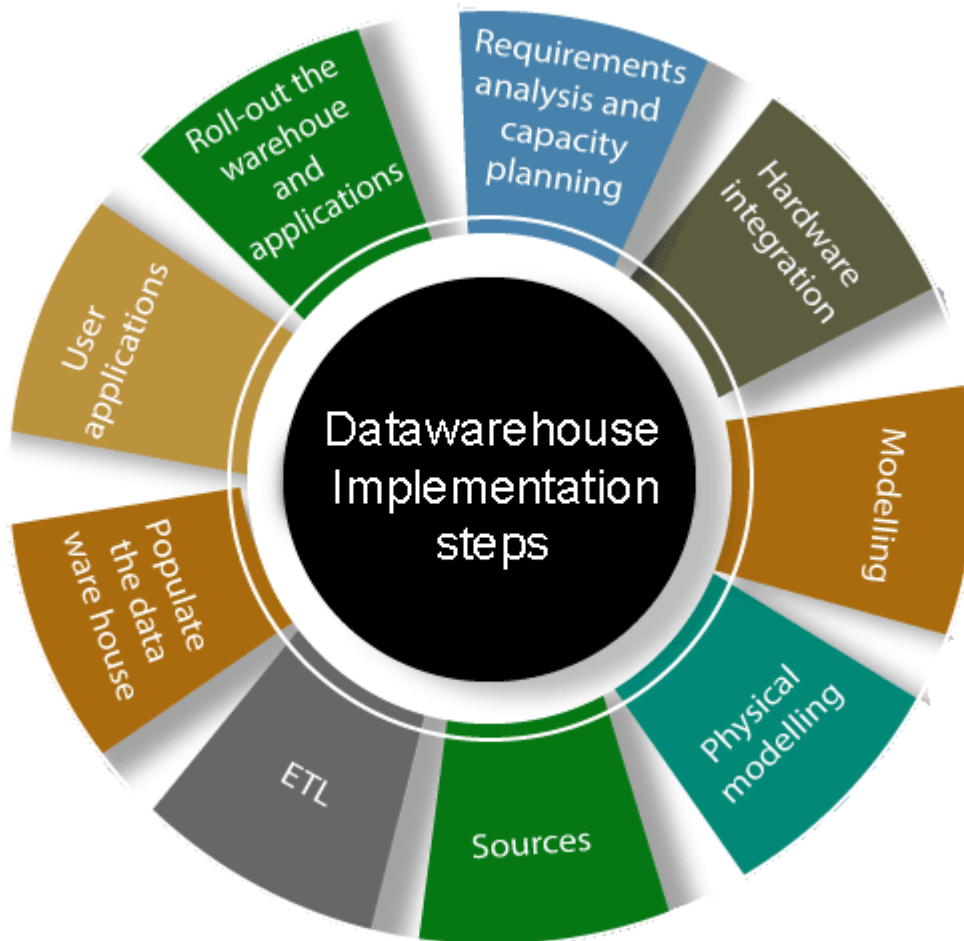
In the schema below, we have a fact table FACT_SALES that has a grain that gives us the number of units sold by date, by store, and product.

All other tables such as DIM_DATE, DIM_STORE and DIM_PRODUCT are dimensions tables. This schema is known as the star schema.



8. Design and implement effective physical data structure for data warehouse

There are various implementation in data warehouses which are as follows



8.1. Requirements analysis and capacity planning: The first process in data warehousing involves defining enterprise needs, defining architectures, carrying out capacity planning, and selecting the hardware and software tools. This step will contain be consulting senior management as well as the different stakeholder.

8.2. Hardware integration: Once the hardware and software has been selected, they require to be put by integrating the servers, the storage methods, and the user software tools.

8.3. Modeling: Modelling is a significant stage that involves designing the warehouse schema and views. This may contain using a modeling tool if the data warehouses are sophisticated.

8.4. Physical modeling: For the data warehouses to perform efficiently, physical modeling is needed. This contains designing the physical data warehouse organization, data placement, data partitioning, deciding on access techniques, and indexing.

8.5. Sources: The information for the data warehouse is likely to come from several data sources. This step contains identifying and connecting the sources using the gateway, ODBC drives, or another wrapper.

8.6. ETL: The data from the source system will require to go through an ETL phase. The process of designing and implementing the ETL phase may contain defining a suitable ETL tool vendors and purchasing and implementing the tools. This may contains customize the tool to suit the need of the enterprises.

8.7. Populate the data warehouses: Once the ETL tools have been agreed upon, testing the tools will be needed, perhaps using a staging area. Once everything is working adequately, the ETL tools may be used in populating the warehouses given the schema and view definition.

8.8. User applications: For the data warehouses to be helpful, there must be end-user applications. This step contains designing and implementing applications required by the end-users.

8.9. Roll-out the warehouses and applications: Once the data warehouse has been populated and the end-client applications tested, the warehouse system and the operations may be rolled out for the user's community to use.

QUESTION BANK

Q-1: 1 Mark Question

1. Give the full form of UID. (July-2021)
2. Give the full form of OLTP. (July-2021,2019)
3. Give the full form of SSAS. (July-2021)
4. What is Table Space? (July-2021)
5. List types of Data warehouse Designs. (July-2021)
6. What is fact table? (July-2021)
7. What is Role playing dimension? (July-2021)
8. Logical Design Entity known as in physical design. (July-2021)
9. List out types of entity relationship. (BKNMU- June-2021,2019)
10. Fact table Contain only numeric facts: True/False (BKNMU - april-2020)
11. Where Fact table is located in the database? (BKNMU - april-2020,2019)
12. Where Dimension table is located in the database? (BKNMU- april-2020)
13. What is Star Schema? (BKNMU - april-2020,2019)
14. What is snowflake schema?
15. How many fact tables are there in a star schema? (BKNMU - april-2022)
16. DMQL stands for _____. (BKNMU - april-2022)
17. DM stands for _____. (BKNMU - april-2022)
18. What is data warehouse schema? (2023)
19. SCD stands for_____ (2023)
20. What is fact table? (2023)

Q-2: 3 Mark Question

1. Explain types of dimension. (July-2021)
2. Give the difference between Fact table and Dimension table. (July-2021)
3. Explain Out trigger dimension. (July-2021)
4. What is Relationship? (July-2021)
5. Define Surrogate Key. Where is it used? (April-2020)
6. Explain Out trigger dimension. (April-2020)
7. Give steps for logical design of data warehouse. (BKNMU - June-2021)
8. Give steps for physical design of data warehouse. (BKNMU - June-2021,2019)(2023)
9. What is data compression? (BKNMU - June-2021,2019)
10. What is Snowflake Schema? (BKNMU - april-2020)(2023)
11. What is Data Sources? (BKNMU - april-2020)
12. Write differences between fact table and dimension table.
13. Discuss logical design for data warehouse. (BKNMU - april-2022)
14. Explain snowflake schema. (BKNMU - april-2022)
15. Write a note on table space. (BKNMU - april-2022)
16. Discuss view in detail. (2023)
17. Define partitioned table. How it is useful? (2023)

Q-3: 5 Mark Question

1. Explain Star Schema with Example. (July-2021) (BKNMU - april-2022)(2023)
2. Give the difference between logical design and Physical design. (July-2021)
3. Explain snowflake schema with example. (BKNMU - June-2021)
4. Explain Fact table with its type. (BKNMU - april-2020,2019)
5. Explain Logical Design for Data Warehouse. (BKNMU - april-2020)(2023)
6. Explain Dimension table with its type. (BKNMU - april-2020,2019)
7. Explain physical Design for Data Warehouse. (BKNMU - april-2022)