Unit 3 : Data Warehouse Logical and Physical Design

Data Warehouse Logical Design

- **Logical design** is the phase of a database design concerned with identifying the relationships among the data elements.
- The logical design should result in
  - A set of entities and attributes corresponding to fact tables and dimension tables.
  - A model of operational data from your source into subject-oriented information in your target data warehouse schema.

- The steps for logical data model are indicated below:
  - Identify all entities.
  - Identify primary keys for all entities.
  - Find the relationships between different entities.
  - Find all attributes for each entity.
  - Resolve all entity relationships that is many-to-many relationships.
  - Normalization if required.

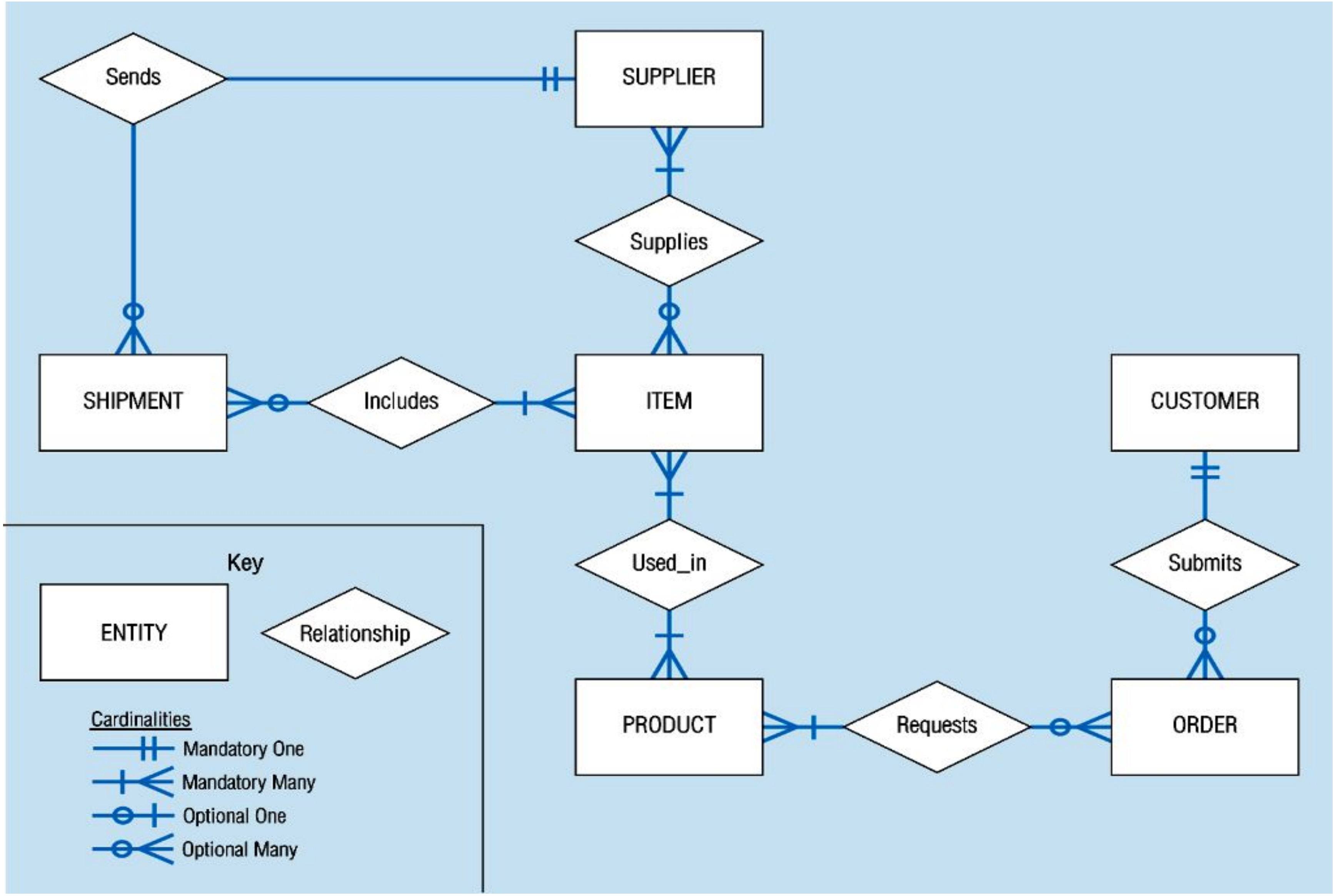


Figure: Sample E-R Diagram

Multi-dimensional data model

- The multidimensional data model is an integral part of On-Line Analytical Processing, or OLAP.
- Because OLAP is on-line, it must provide answers quickly.
- Because OLAP is also analytic, the queries are complex. The multidimensional data model is designed to solve complex queries in real time.
- The multidimensional data model is composed of logical cubes, measures, dimensions, hierarchies, levels, and attributes.
- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- The lattice of cuboids forms a data cube.

Data Cube

- A **data cube** provides a multidimensional view of data and allows the pre-computation and fast accessing of summarized data.
- A data cube allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.
- A **data cube** is a three- (or higher) dimensional array of value.

Example of 2D view of sales

A 2-D view of sales data for *AllElectronics* according to the dimensions *time* and *item*, where the sales are from branches located in the city of Vancouver. The measure displayed is *dollars_sold* (in thousands).

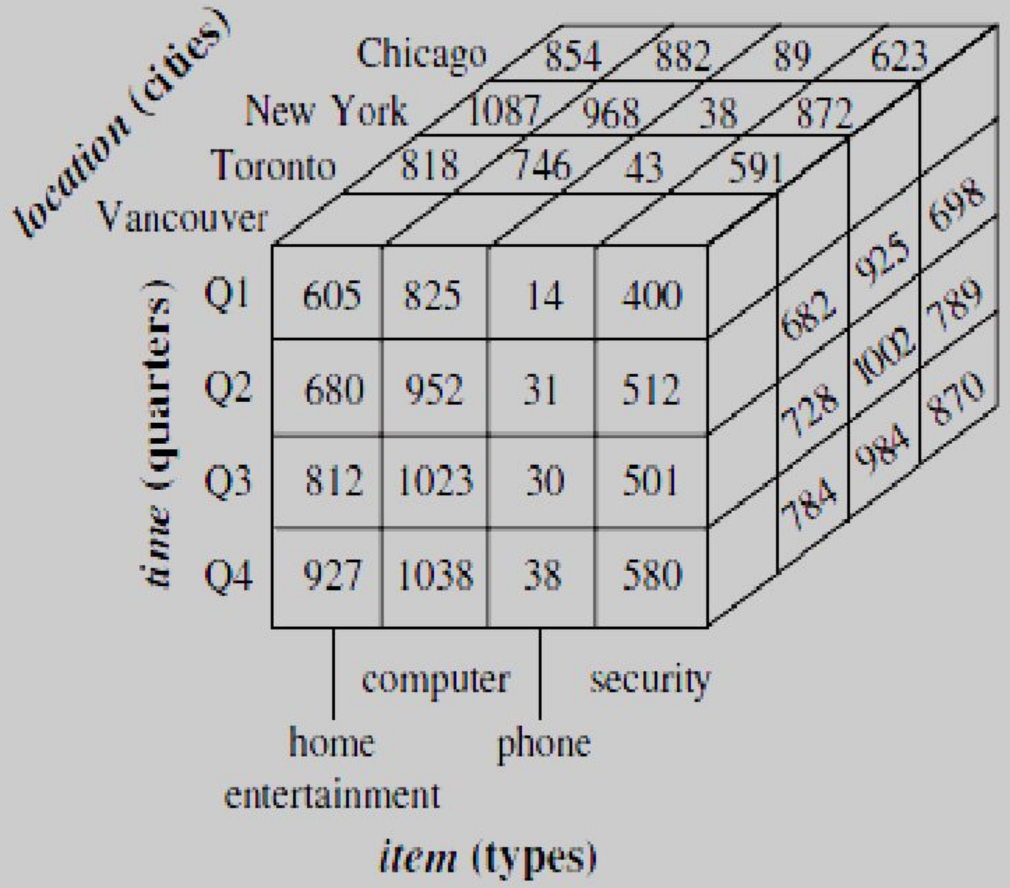
	location = "Vancouver"			
	item *(type)*			
time *(quarter)*	*home entertainment*	*computer*	*phone*	*security*
Q1	605	825	14	400
Q2	680	952	31	512
Q3	812	1023	30	501
Q4	927	1038	38	580

Example of 3D view of sales

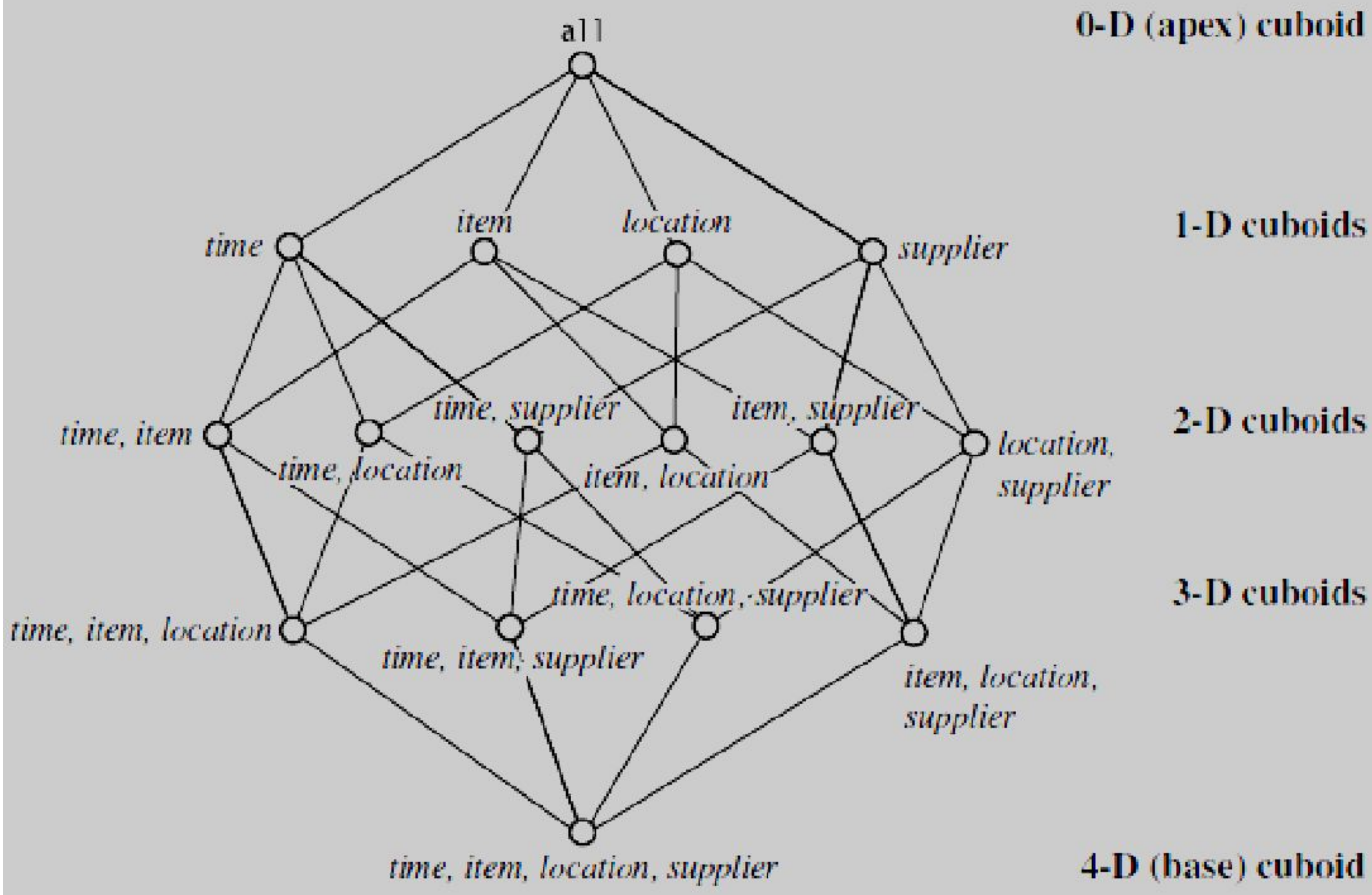
Table 3.3 A 3-D view of sales data for *AllElectronics*, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).

	location = "Chicago"				location = "New York"				location = "Toronto"				location = "Vancouver"			
	item				item				item				item			
time	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.	home ent.	comp.	phone	sec.
Q1	854	882	89	623	1087	968	38	872	818	746	43	591	605	825	14	400
Q2	943	890	64	698	1130	1024	41	925	894	769	52	682	680	952	31	512
Q3	1032	924	59	789	1034	1048	45	1002	940	795	58	728	812	1023	30	501
Q4	1129	992	63	870	1142	1091	54	984	978	864	59	784	927	1038	38	580

Example of 3D data cube



A 3-D data cube representation of the data in Table 3.3, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars_sold* (in thousands).



Lattice of cuboids, making up a 4-D data cube for the dimensions *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

Dimensions and Measures

- Every dimensional model (DM) is composed of one table with a composite primary key, called the fact table, and a set of smaller tables called dimension tables.
- Each dimension table has a simple (non-composite) primary key that corresponds exactly to one of the components of the composite key in the fact table.

- In general terms, dimensions are the perspectives or entities with respect to which an organization wants to keep records.
- For example, *AllElectronics* may create a *sales* data warehouse in order to keep records of the store's sales with respect to the dimensions *time, item, branch,* and *location*.

- Each dimension may have a table associated with it, called a dimension table, which further describes the dimension.
- For example, a dimension table for item may contain the attributes item name, brand, and type. Dimension tables can be specified by users or experts, or automatically generated and adjusted based on data distributions.

Fact Table

- A multidimensional data model is typically organized around a central theme, like *sales*, for instance.
- This theme is represented by a fact table. Facts are numerical measures.
- Think of them as the quantities by which we want to analyze relationships between dimensions.

- The fact table contains the names of the *facts*, or measures, as well as keys to each of the related dimension tables.
- A fact table is composed of two or more primary keys and usually also contains numeric data.
- Because it always contains at least two primary keys it is always a M-M relationship.
- Fact tables contain business event details for summarization.

- The logical model for a fact table contains a foreign key column for the primary keys of each dimension.
- The combination of these foreign keys defines the primary key for the fact table.
- Fact tables are often very large, containing hundreds of millions of rows and consuming hundreds of gigabytes or multiple terabytes of storage.

Dimension table

- A dimension table stores data about the ways in which the data in the fact table can be analyzed.
- Contrary to fact tables, dimension tables contain descriptive attributes (or fields) that are typically textual fields.
- A dimension table may be used in multiple places if the data warehouse contains multiple fact tables or contributes data to data marts.

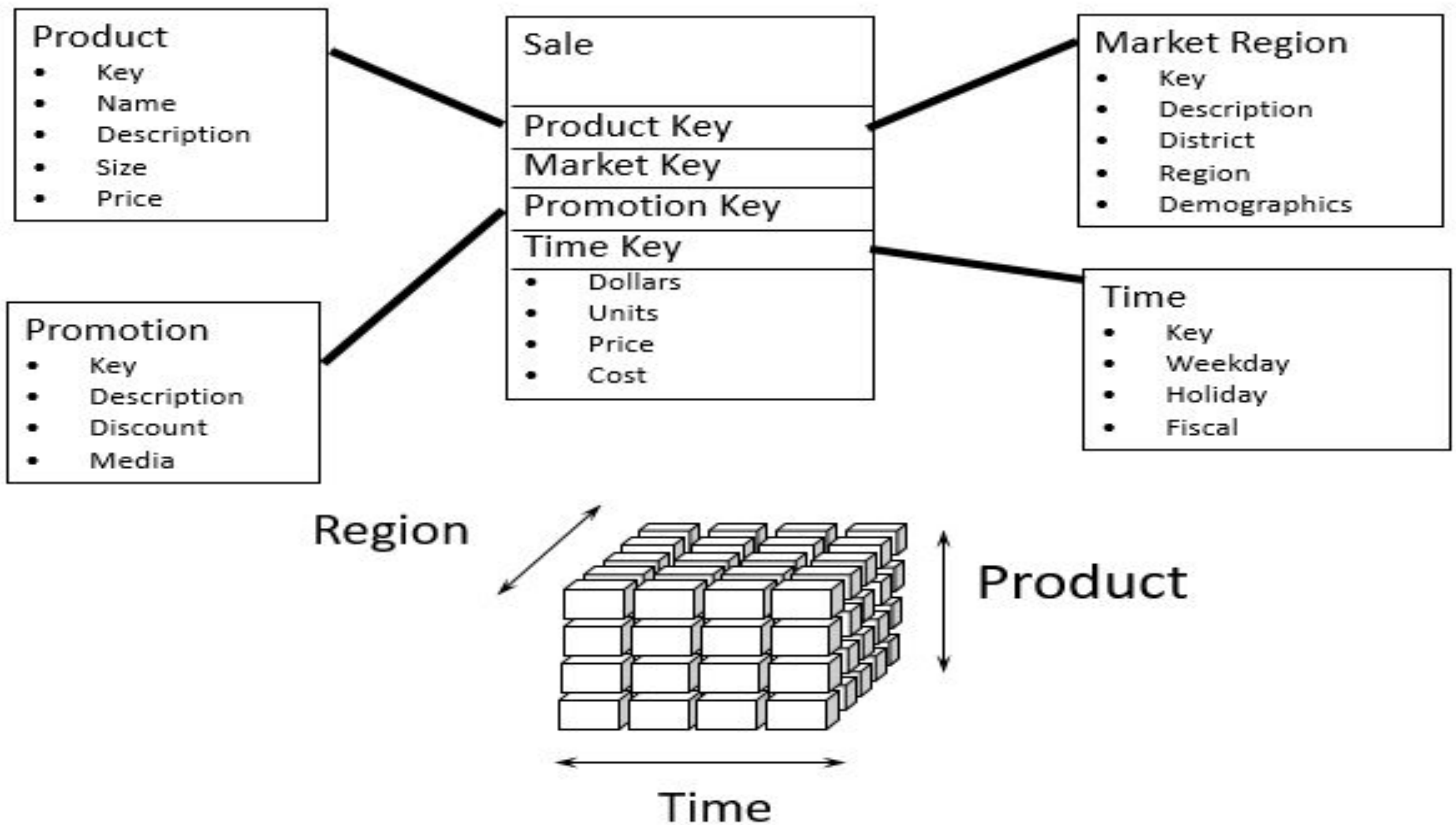


Figure: Dimensional Model

For example, a time dimension often contains the hierarchy elements: (all time), Year, Quarter, Month, Day, or (all time), Year Quarter, Week, Day.

A Data Warehouse Schema

- The schema is a logical description of the entire database.
- The schema includes the name and description of records of all record types including all associated data-items and aggregates.
- The most popular data model for a data warehouse is **a multidimensional model.**
- Such a model can exist in the following forms
  - a star schema
  - a snowflake schema
  - a fact constellation schema.

Star Schema

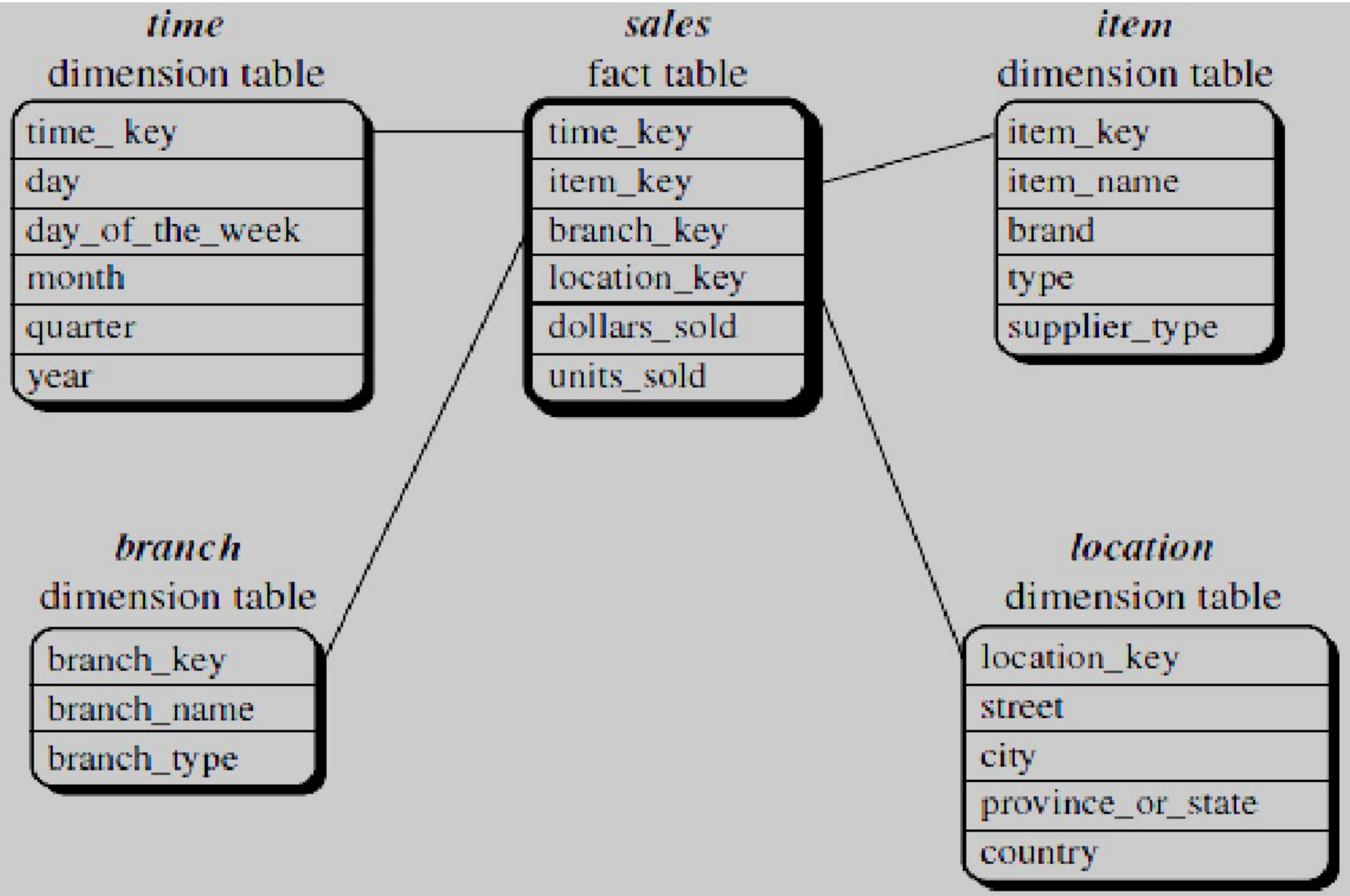
- The most popular schema design for data warehouses is the Star Schema.
- Each dimension is stored in a dimension table and each entry is given its own unique identifier.
- The dimension tables are related to one or more fact tables.
- The fact table contains a composite key made up of the identifiers (primary keys) from dimension tables.

Star Schema

- The fact table also contains facts about the given combination of dimensions. For example, a combination of store_key, date_key and product_key giving the amount of a certain product sold on a given day at a given store.
- Fact table has foreign keys to all dimension tables in a star schema. In the example, there are three foreign keys (date key, product key, and store key).

Star Schema

- Fact tables are normalized, whereas dimension tables are not.
- Fact tables are very large as compared to dimension tables.
- Others name: star-join schema, data cube, data list, grid file and multi-dimension schema.
- The facts in a star schema are of the following three types
  - Fully-additive
  - Semi-additive
  - Non-additive



Star schema of a data warehouse for sales.

Advantages of Star Schema

- A star schema is very simple from the users‘ point of view.
- Queries are never complex because the only joins and conditions involve a fact table and a single level of dimension tables, without the indirect dependencies to other tables.
- Provide a direct mapping between the business entities being analyzed by end users and the schema design.
- Provide highly optimized performance for typical star queries.

Advantages of Star Schema

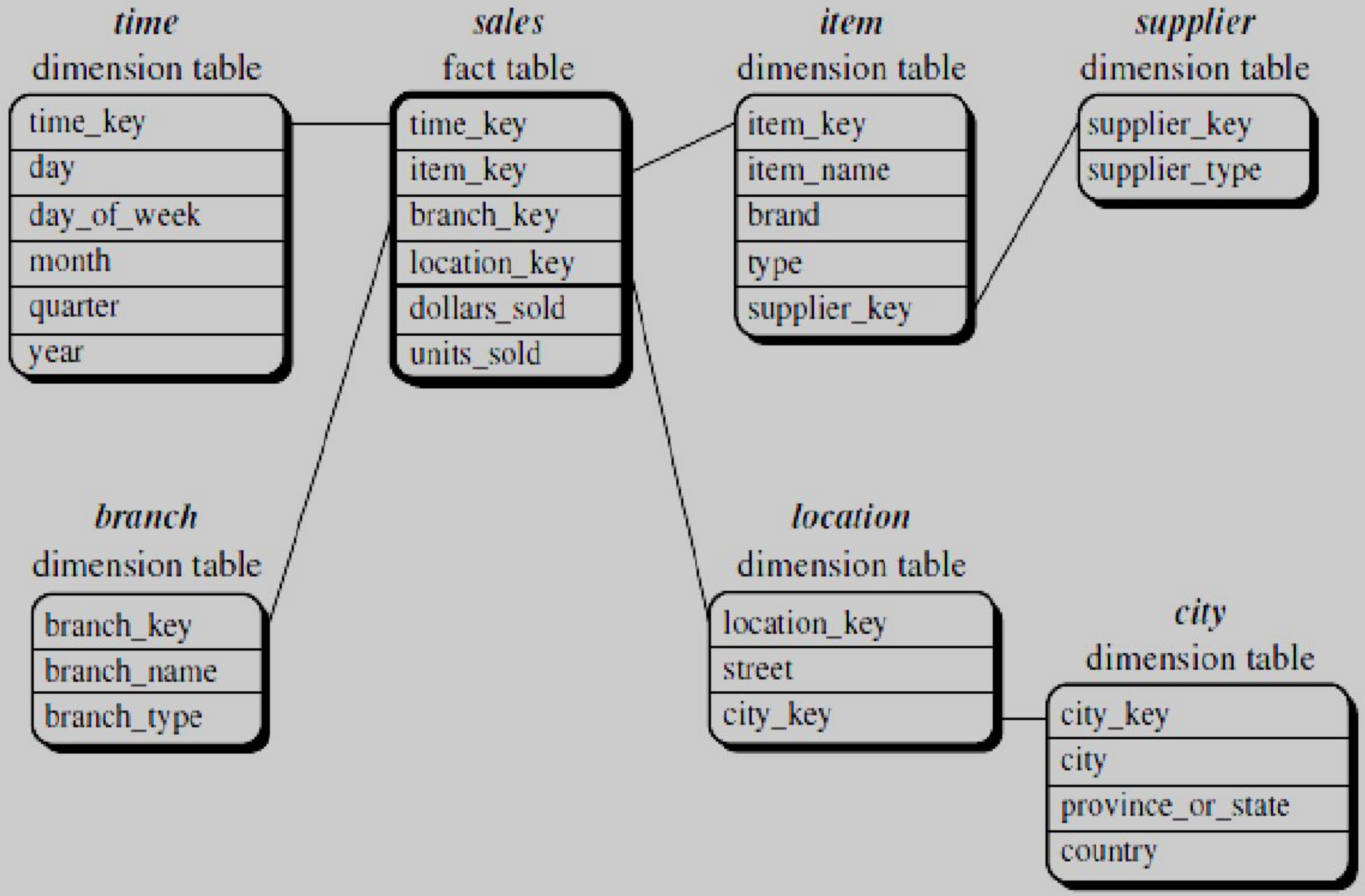
- Are widely supported by a large number of business intelligence tools.
- Star schemas are used for both simple data marts and very large data warehouses.
- Star Schema is very easy to understand, even for non-technical business manager.
- Star Schema provides better performance and smaller query times
- Star Schema is easily extensible and will handle future changes easily

Snowflake Schema

- A schema is called a *snowflake schema* if one or more dimension tables do not join directly to the fact table but must join through other dimension tables.
- It is a variant of star schema model.
- It has a single, large and central fact table and one or more tables for each dimension.

Snowflake Schema

- The major difference between the snowflake and star schema models is that the dimension tables of the snowflake model may be kept in normalized form to reduce redundancies.
- Normalization of dimension tables
- Each hierarchical level has its own table
- easy to maintain and less memory space is required
- a lot of joins can be required if they involve attributes in secondary dimension tables



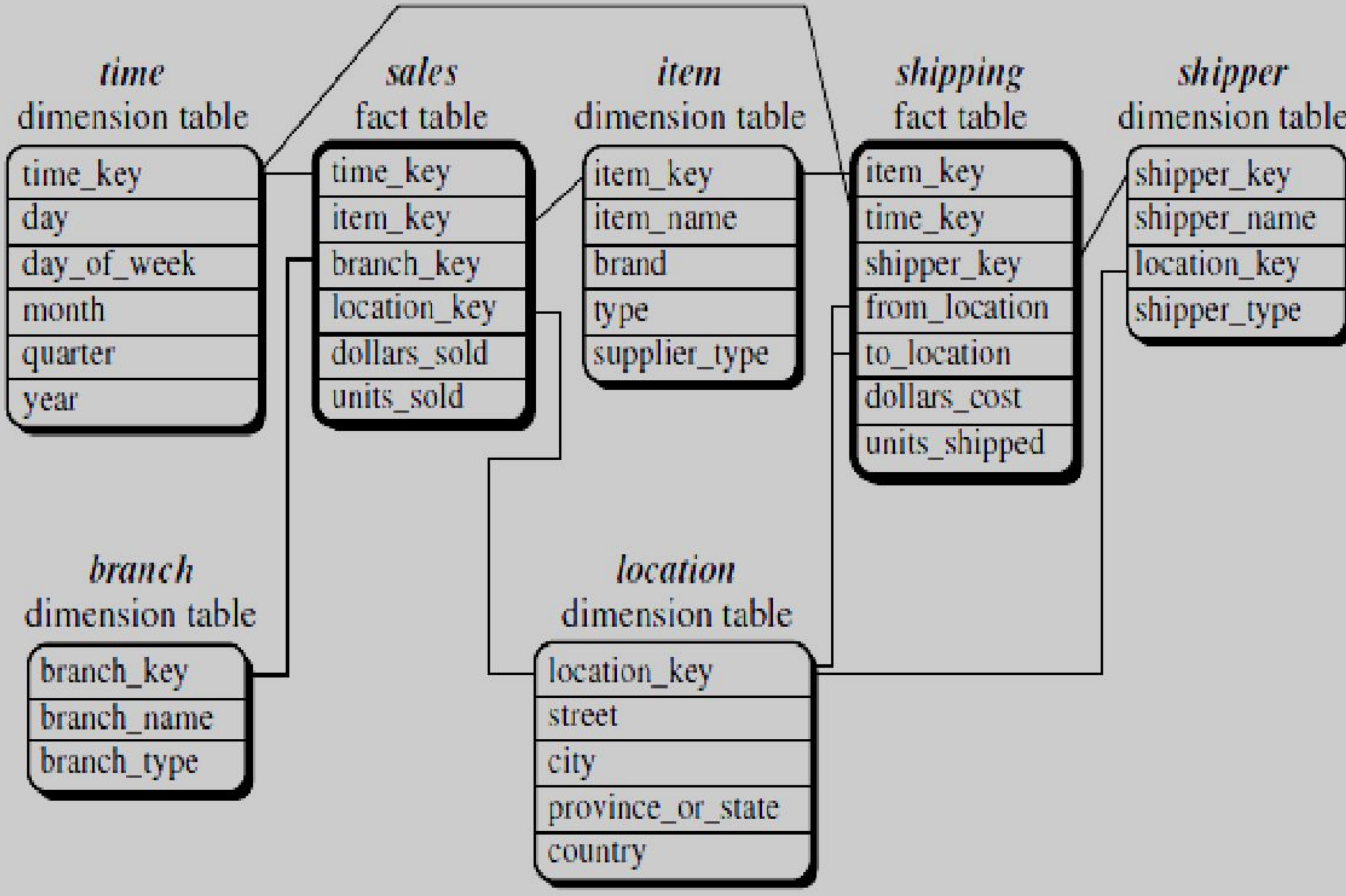
Snowflake schema of a data warehouse for sales.

Difference between Star Schema and Snow-flake Schema

- Star Schema is a multi-dimension model where each of its disjoint dimension is represented in single table.
- Snow-flake is normalized multi-dimension schema when each of disjoint dimension is represent in multiple tables.
- Star schema can become a snow-flake
- Both star and snowflake schemas are dimensional models; the difference is in their physical implementations.
- Snowflake schemas support ease of dimension maintenance because they are more normalized.
- Star schemas are easier for direct user access and often support simpler and more efficient queries.

Fact Constellation Schema

- Sophisticated applications may require multiple fact tables to *share* dimension tables.
- This kind of schema can be viewed as a collection of stars, and hence, galaxy schema or a fact constellation.
- In a fact constellation, there are multiple fact tables.
- In the Fact Constellations, aggregate tables are created separately from the detail, therefore, it is impossible to pick up other queries from different tables.
- Fact Constellation is a good alternative to the Star, but when dimensions have very high cardinality, the sub-selects in the dimension tables can be a source of delay.



Fact constellation schema of a data warehouse for sales and shipping.

Starflake Schema

- A starflake schema is a combination of a star schema and a snowflake schema.
- Starflake schemas are snowflake schemas where only some of the dimension tables have been de-normalized.

Materialized View

- 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.
- Materialized views can be best explained by Multidimensional lattice.
- It is useful to materialize a view when:
  - It directly solves a frequent query
  - It reduces the costs of some queries

Physical Design

Physical design is the phase of a database design following the logical design that identifies the actual database tables and index structures used to implement the logical design.

In the physical design, you look at the most effective way of storing and retrieving the objects as well as handling them from a transportation and backup/recovery perspective.

Physical design decisions are mainly driven by query performance and database maintenance aspects.

During the logical design phase, you defined a model for your data warehouse consisting of entities, attributes, and relationships.

The entities are linked together using relationships.

Attributes are used to describe the entities.

The unique identifier (UID) distinguishes between one instance of an entity and another.

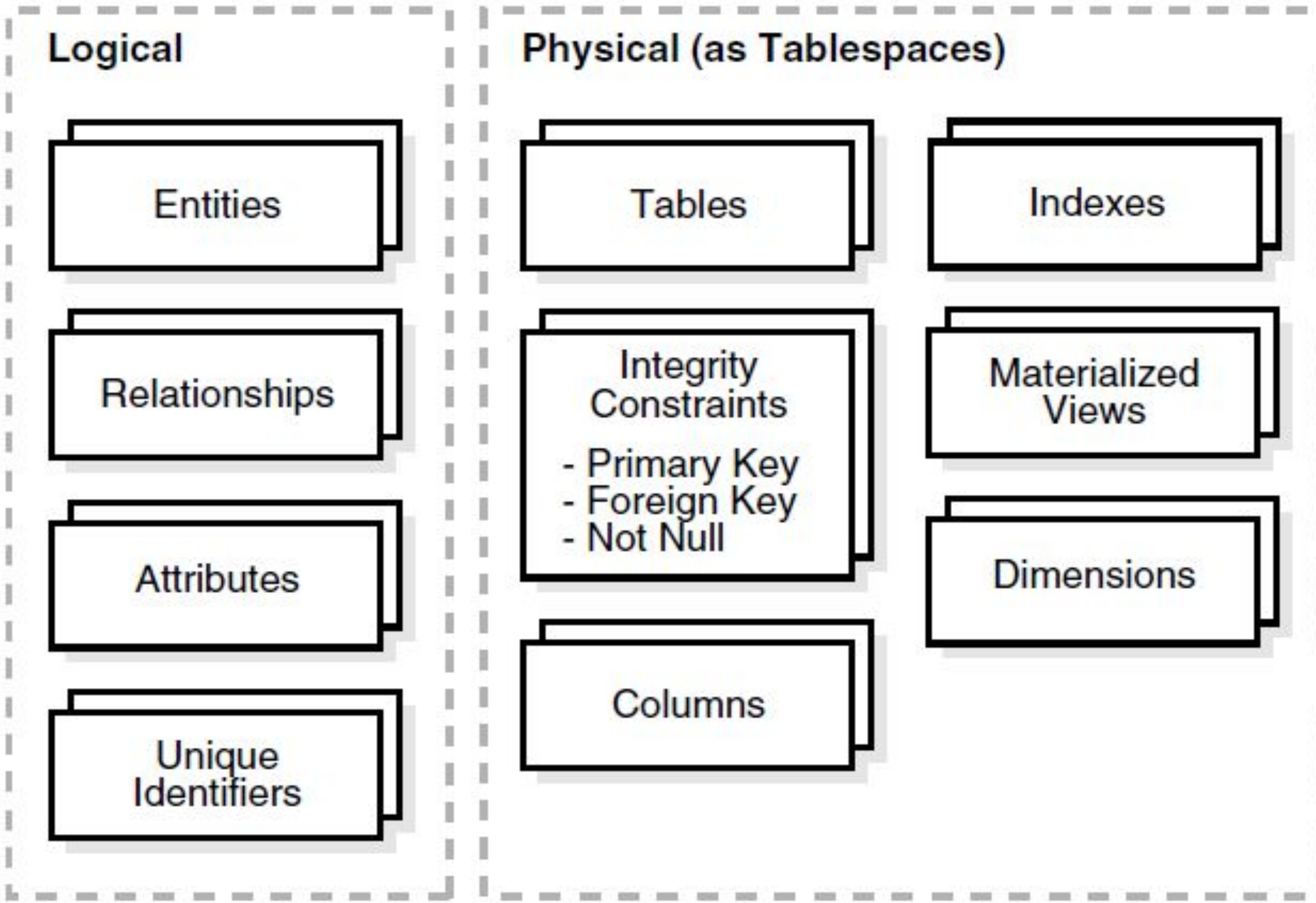


Figure: Logical Design Compared with Physical Design

During the physical design process, you translate the expected schemas into actual database structures.

At this time, you have to map:

- Entities to tables
- Relationships to foreign key constraints
- Attributes to columns
- Primary unique identifiers to primary key constraints
- Unique identifiers to unique key constraints

Physical Data Model

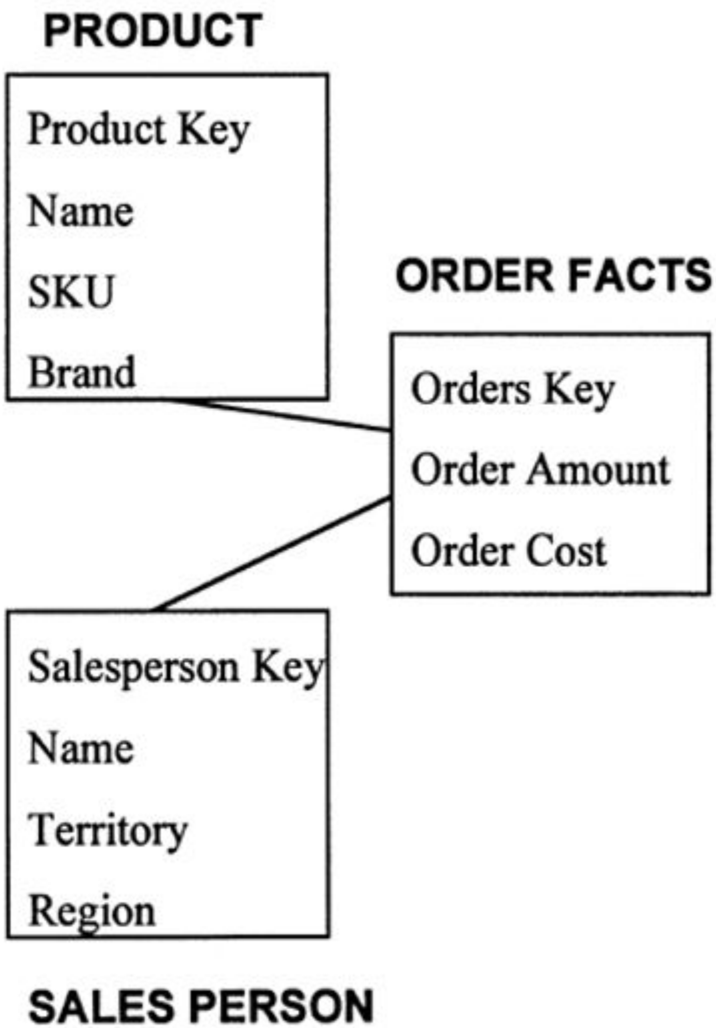
Features of physical data model include:

- ❖ Specification of all tables and columns.
- ❖ Specification of Foreign keys.
- ❖ De-normalization may be performed if necessary.
- ❖ At this level, specification of logical data model is realized in the database.

The steps for physical data model design involves:

- Conversion of entities into tables,
- Conversion of relationships into foreign keys, Conversion of attributes into columns, and
- Changes to the physical data model based on the physical constraints.

LOGICAL MODEL



PHYSICAL MODEL

Name	Type	Comments
PRODUCT		Product dimension table including all of
		company's products
product_key	integer	Primary key
product_name	char(25)	Name of product as used by marketing
product_sku	char(20)	Stock Keeping Unit in source systems
product_brand	char(25)	Name of brand as used by marketing
SALESPERSON		Sales Person dimension table includes all
		sales persons from all regions
salpers_key	integer	Primary key
salpers_name	char(30)	Name of sales person as used officially
territory	char(20)	Territory covered by the sales person
region	char(20)	Region containing the territory
ORDER FACT		Fact table containing metrics about all
		the orders received by the company
product_ref	integer	Partial primary key, also foreign key
		referencing product dimension table
salpers_ref	integer	Partial primary key, also foreign key
		referencing salesperson dimension table
order_amount	num(8,2)	Sales amount of order in dollars
order_cost	num(8,2)	Cost amount for the order in dollars

Figure: Logical model and physical model

Physical Design Structures

Once you have converted your logical design to a physical one, you will need to create some or all of the following structures:

- Tablespaces
- Tables and Partitioned Tables
- Views
- Integrity Constraints
- Dimensions

Additionally, the following structures may be created for performance improvement:

- Indexes and Partitioned Indexes
- Materialized Views

Tablespaces

- A tablespace consists of one or more datafiles, which are physical structures within the operating system you are using.
- A datafile is associated with only one tablespace.
- From a design perspective, tablespaces are containers for physical design structures.

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 non-partitioned ones addresses the key problem of supporting very large data volumes by allowing you to divide them into smaller and more manageable pieces.
- Partitioning large tables improves performance because each partitioned piece is more manageable.

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.
- In data warehousing environments, constraints are only used for query rewrite.
- NOT NULL constraints are particularly common in data warehouses.

Indexes and Partitioned Indexes

- Indexes are optional structures associated with tables.
- Indexes are just like tables in that you can partition them (but 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.

Hardware and I/O Consideration

- I/O performance should always be a key consideration for data warehouse designers and administrators.
- The typical workload in a data warehouse is especially I/O intensive, with operations such as large data loads and index builds, creation of materialized views, and queries over large volumes of data.
- The underlying I/O system for a data warehouse should be designed to meet these heavy requirements.

- In fact, one of the leading causes of performance issues in a data warehouse is poor I/O configuration.
- Database administrators who have previously managed other systems will likely need to pay more careful attention to the I/O configuration for a data warehouse than they may have previously done for other environments.
- The I/O configuration used by a data warehouse will depend on the characteristics of the specific storage and server capabilities

There are following five high-level guidelines for data-warehouse I/O configurations:

- Configure I/O for Bandwidth not Capacity
- Stripe Far and Wide
- Use Redundancy
- Test the I/O System Before Building the Database
- Plan for Growth

Parallelism

- Parallelism is the idea of breaking down a task so that, instead of one process doing all of the work in a query, many processes do part of the work at the same time.
- Parallel execution is sometimes called parallelism.
- Parallel execution dramatically reduces response time for data-intensive operations on large databases typically associated with decision support systems (DSS) and data warehouses.
- An example of this is when four processes handle four different quarters in a year instead of one process handling all four quarters by itself.

Parallelism improves processing for:

- Queries requiring large table scans, joins, or partitioned index scans
- Creation of large indexes
- Creation of large tables (including materialized views)
- Bulk inserts, updates, merges, and deletes

Parallelism benefits systems with all of the following characteristics:

- Symmetric multiprocessors (SMPs), clusters, or massively parallel systems
- Sufficient I/O bandwidth
- Underutilized or intermittently used CPUs (for example, systems where CPU usage is typically less than 30%)
- Sufficient memory to support additional memory-intensive processes, such as sorts, hashing, and I/O buffers

Indexes

- Indexes are optional structures associated with tables and clusters.
- Indexes are structures actually stored in the database, which users create, alter, and drop using SQL statements.
- You can create indexes on one or more columns of a table to speed SQL statement execution on that table.
- In a query-centric system like the data warehouse environment, the need to process queries faster dominates.
- Among the various methods to improve performance, indexing ranks very high.
- Indexes are typically used to speed up the retrieval of records in response to search conditions.

- **Indexes** can be unique or non-unique.
- Unique indexes guarantee that no two rows of a table have duplicate values in the key column (or columns).
- Non-unique indexes do not impose this restriction on the column values.
- Index structures applied in warehouses are:
  - Inverted lists
  - Bitmap indexes
  - Join indexes
  - Text indexes
  - B-Tree Index

Thank you !!!