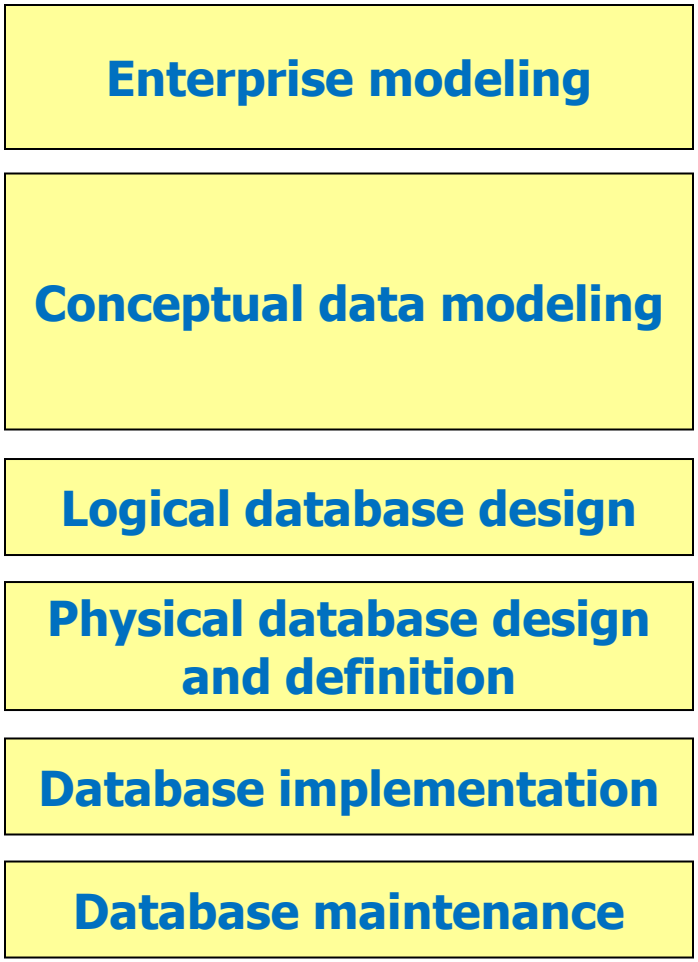
Unit 2: Data Warehouse Logical Design

Lecturer : Bijay Mishra

Database Development Process



- A conceptual data model include identification of important entities and the relationships among them. At this level, the objective is to identify the relationships among the different entities.

What is a Logical Design??

Logical Design

Logical design is the phase of a database design concerned with identifying the relationships among the data elements.

A logical design is conceptual and abstract. You do not deal with the physical implementation details yet. You deal only with defining the types of information that you need.

Logical design deals with concepts related to a certain kind of DBMS (e.g. relational, object oriented,) but are understandable by end users

The logical design should result in

(1) A set of entities and attributes corresponding to fact tables and dimension tables.

(2) A model of operational data from your source into subject-oriented information in your target data warehouse schema.

You can create the logical design using a pen and paper, or you can use a design tool such as Oracle Warehouse Builder (specifically designed to support modeling the ETL process) or Oracle Designer (a general purpose modeling tool).

The steps of the logical data model include identification of all entities and relationships among them. All attributes for each entity are identified and then the primary key and foreign key is identified. Normally normalization occurs at this level.

In data warehousing, it is common to combine the conceptual data model and the logical data model to a single step. The steps for logical data model are indicated below:

1. Identify all entities.
2. Identify primary keys for all entities.
3. Find the relationships between different entities.
4. Find all attributes for each entity.
5. Resolve all entity relationships that is many-to-many relationships.
6. Normalization if required.

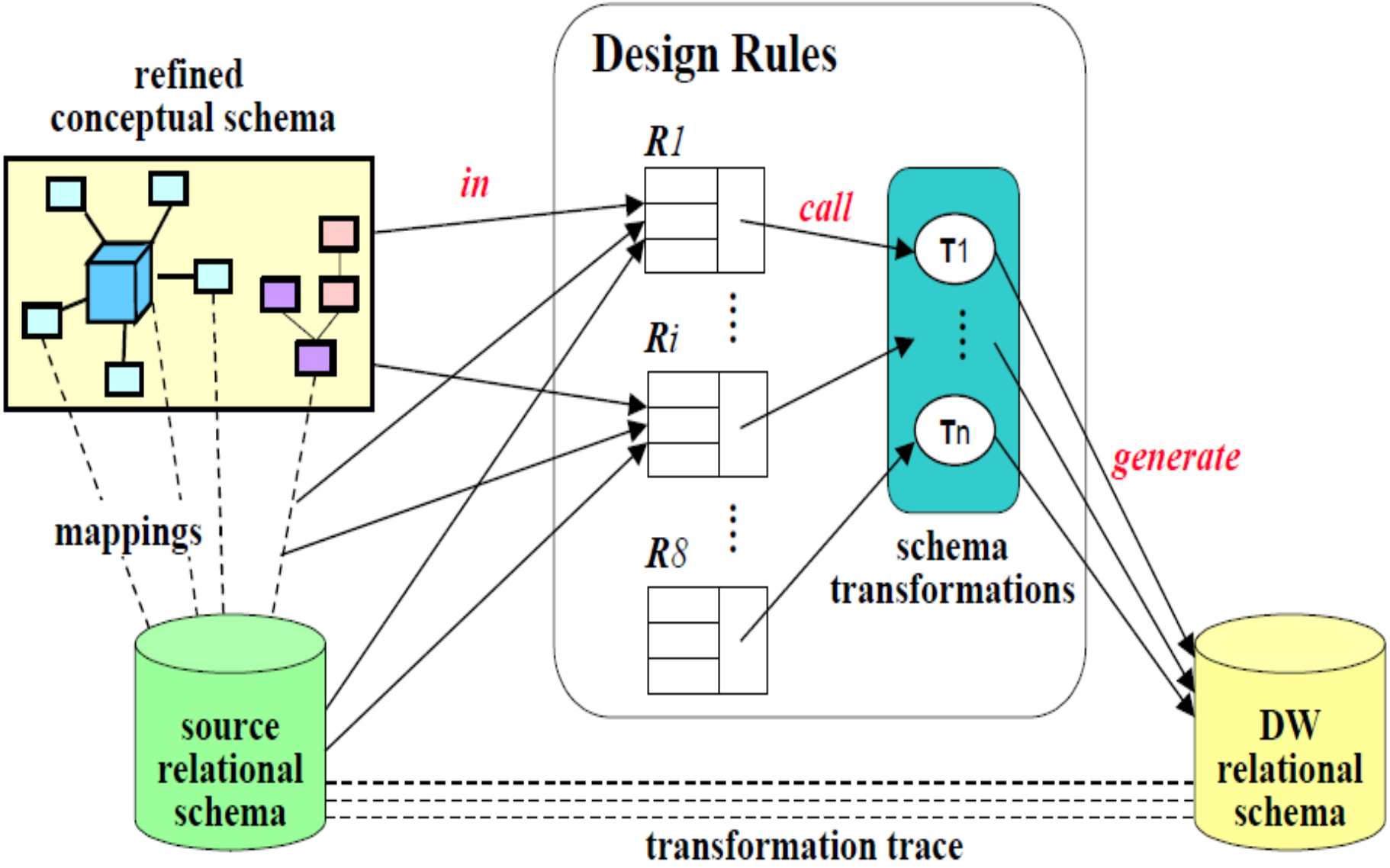
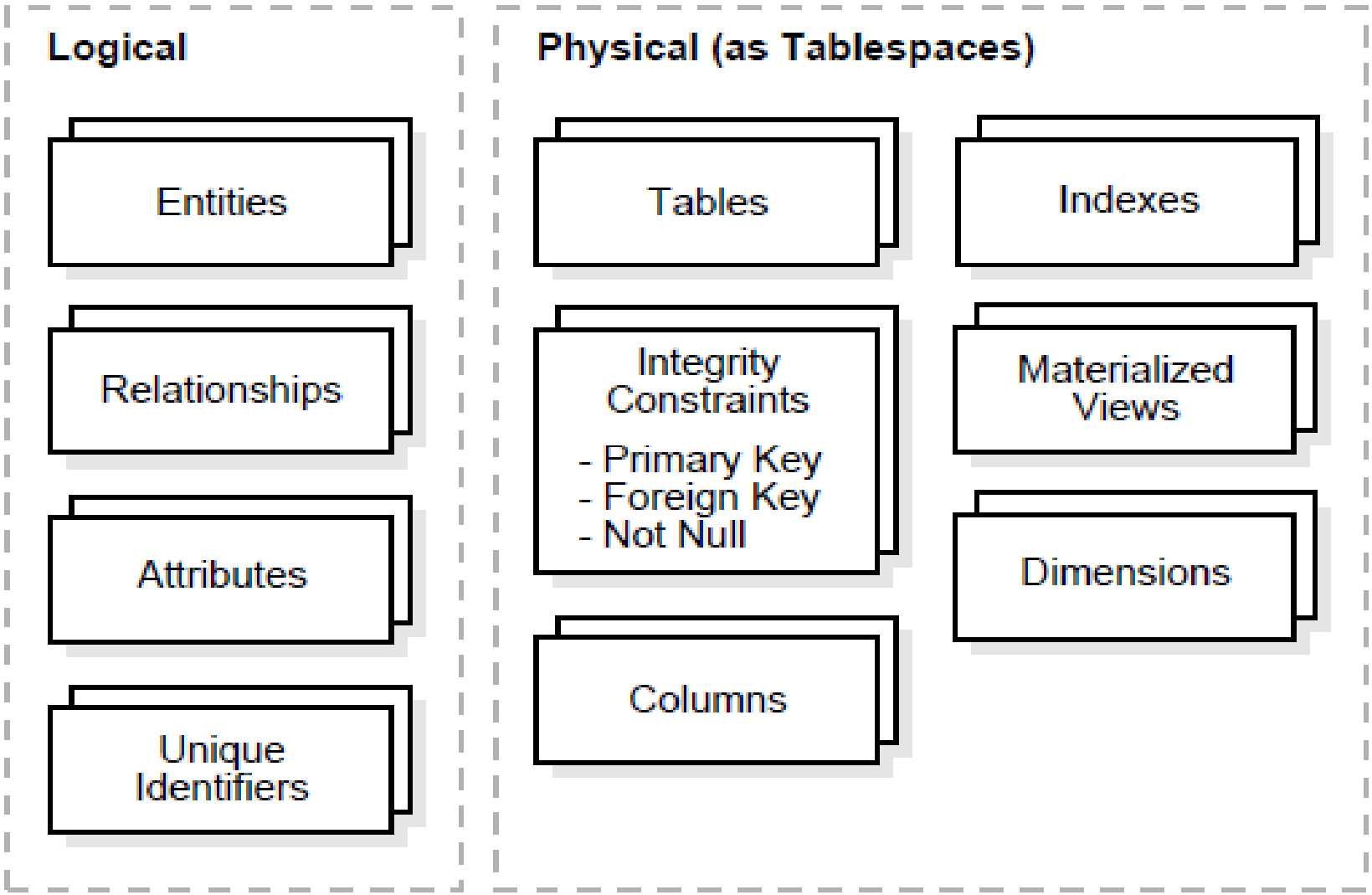


Figure: Data warehouse logical design environment.

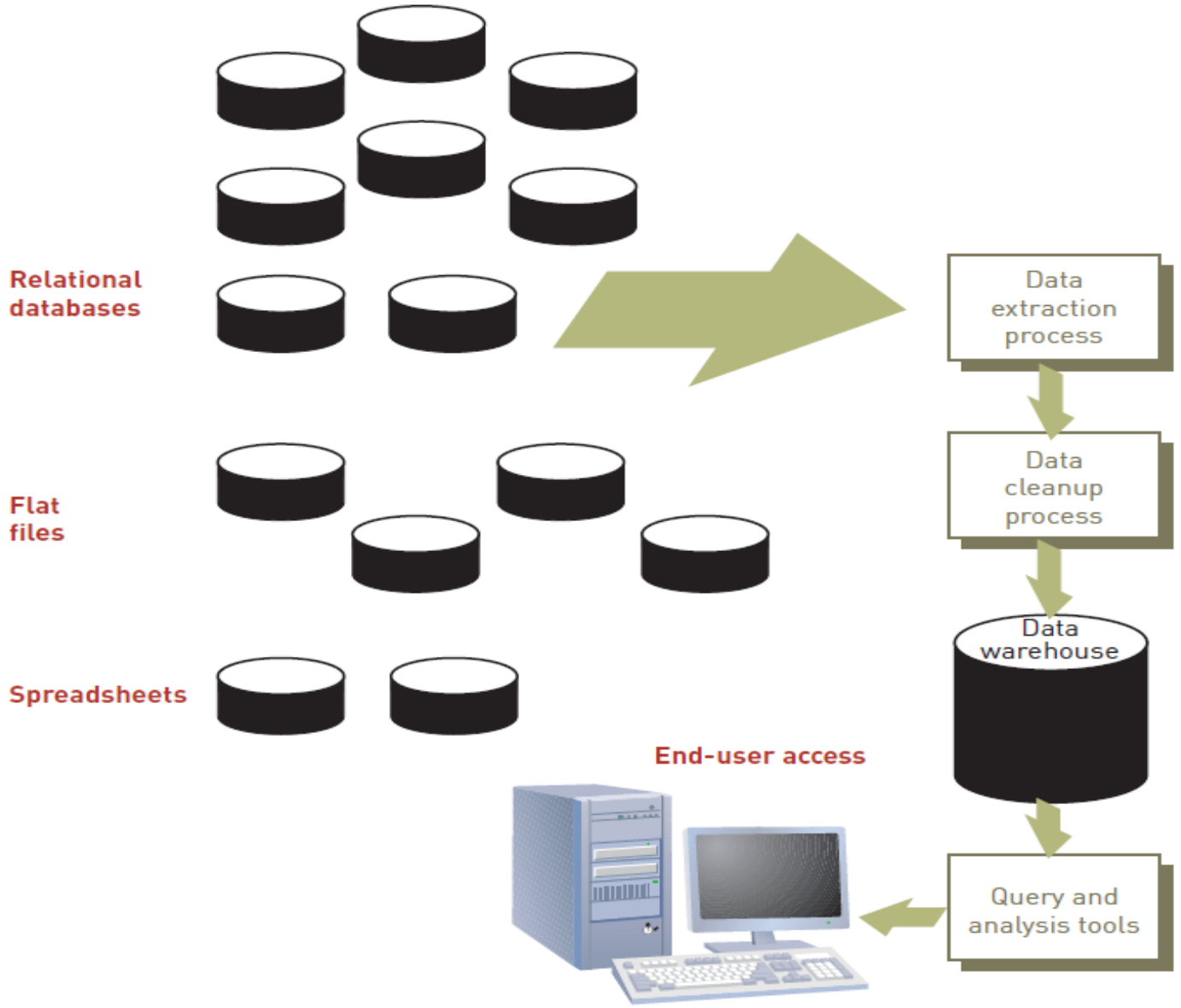
The environment provides the infrastructure to carry out the specified process. It consists of:

- A *refined conceptual schema*, which is built from a conceptual multidimensional schema enriched with design guidelines.
- The *source schema* and the *DW schema*.
- Schema *mappings*, which are used to represent correspondences between the conceptual schema and the source schema.
- A set of *design rules*, which apply the *schema transformations* to the source schema in order to build the DW schema.
- A set of pre-defined *schema transformations* that build new relations from existing ones, applying DW design techniques.
- A *transformation trace*, which keeps the transformations that where applied, providing the mappings between source and DW schemas.

Logical Design compared with Physical Design



From Tables and Spreadsheets to Data Cubes



The process of logical design involves arranging data into a series of logical relationships called entities and attributes.

An **entity** represents a chunk of information. In relational databases, an entity often maps to a table.

An **attribute** is a component of an entity that helps define the uniqueness of the entity. In relational databases, an attribute maps to a column.

Relational database model's structural and data independence enables us to view data *logically* rather than *physically*.

- The logical view allows a simpler file concept of data storage.
- The use of logically independent tables is easier to understand.
- Logical simplicity yields simpler and more effective database design methodologies.

An entity is a person, place, event, or thing for which we intend to collect data.

- **University** -- Students, Faculty Members, Courses
- **Airlines** -- Pilots, Aircraft, Routes, Suppliers

Each entity has certain characteristics known as attributes.

- **Student** -- Student Number, Name, GPA, Date of Enrollment, Data of Birth, Home Address, Phone Number, Major
- **Aircraft** -- Aircraft Number, Date of Last Maintenance, Total Hours Flown, Hours Flown since Last Maintenance

A grouping of related entities becomes an entity set.

- The STUDENT entity set contains all student entities.
- The FACULTY entity set contains all faculty entities.
- The AIRCRAFT entity set contains all aircraft entities

A table contains a group of related entities -- i.e. an entity set.

The terms entity set and table are often used interchangeably.

A table is also called a relation.

While entity-relationship diagramming has traditionally been associated with highly normalized models such as OLTP applications, the technique is still useful for data warehouse design in the form of dimensional modeling.

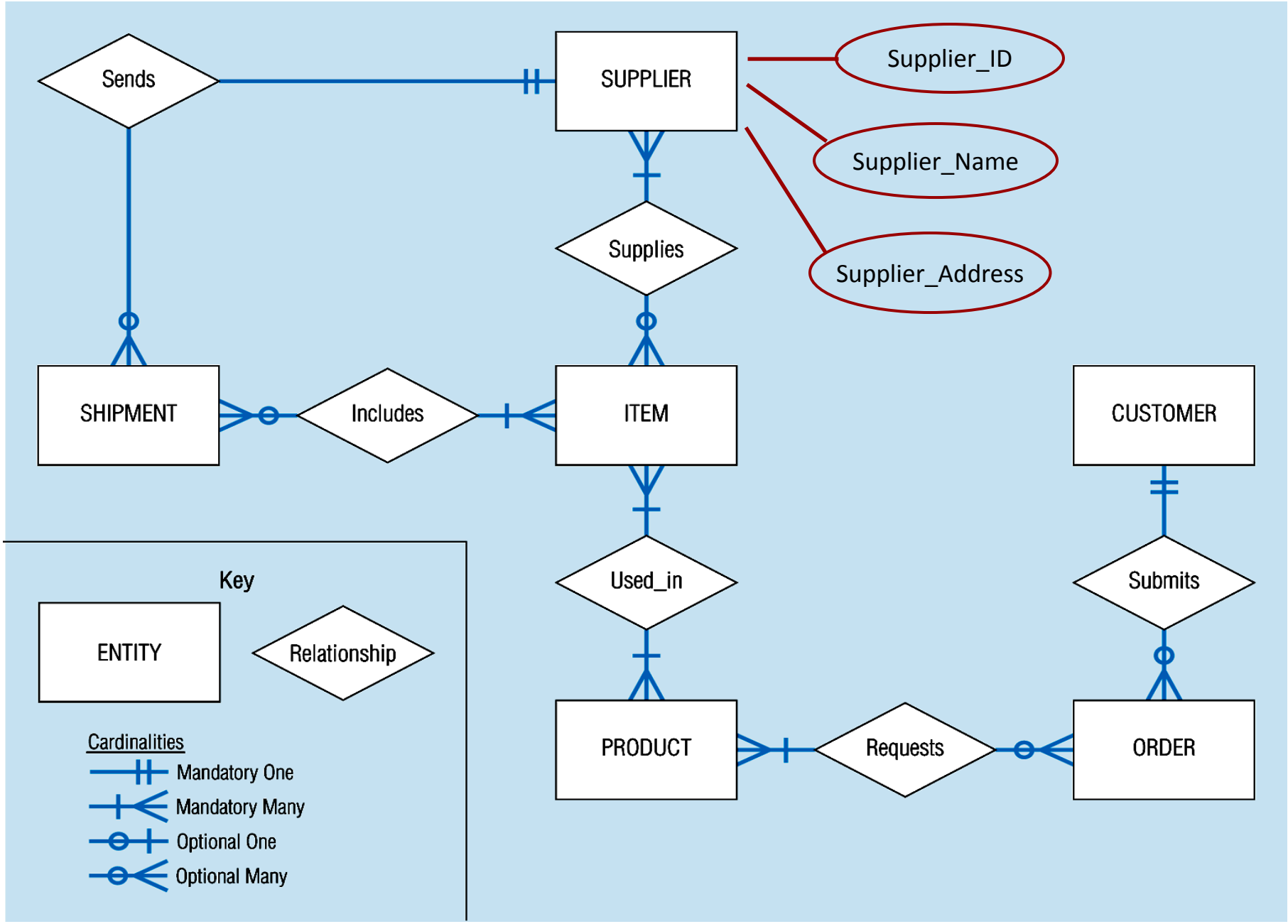


Figure: Sample E-R Diagram

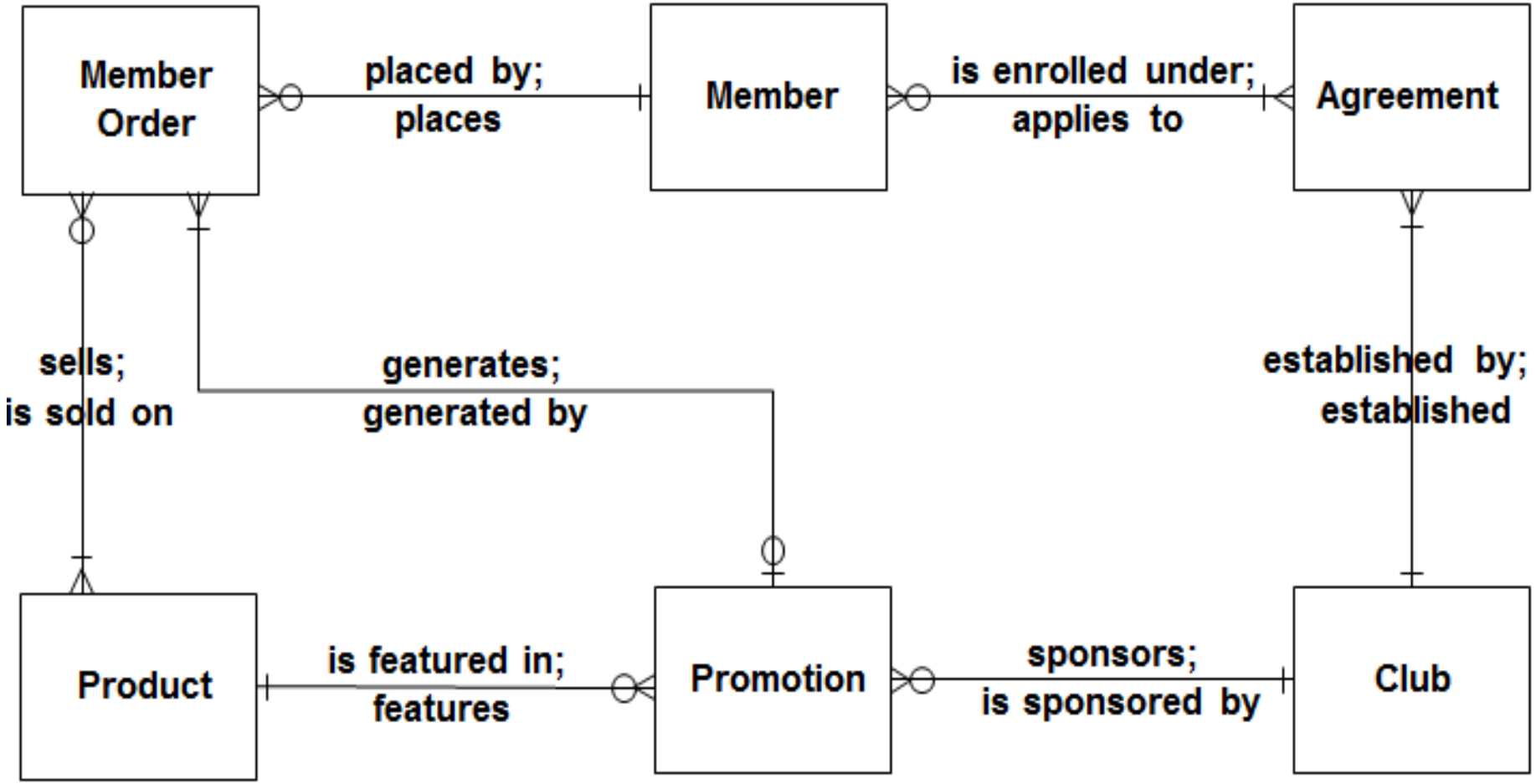
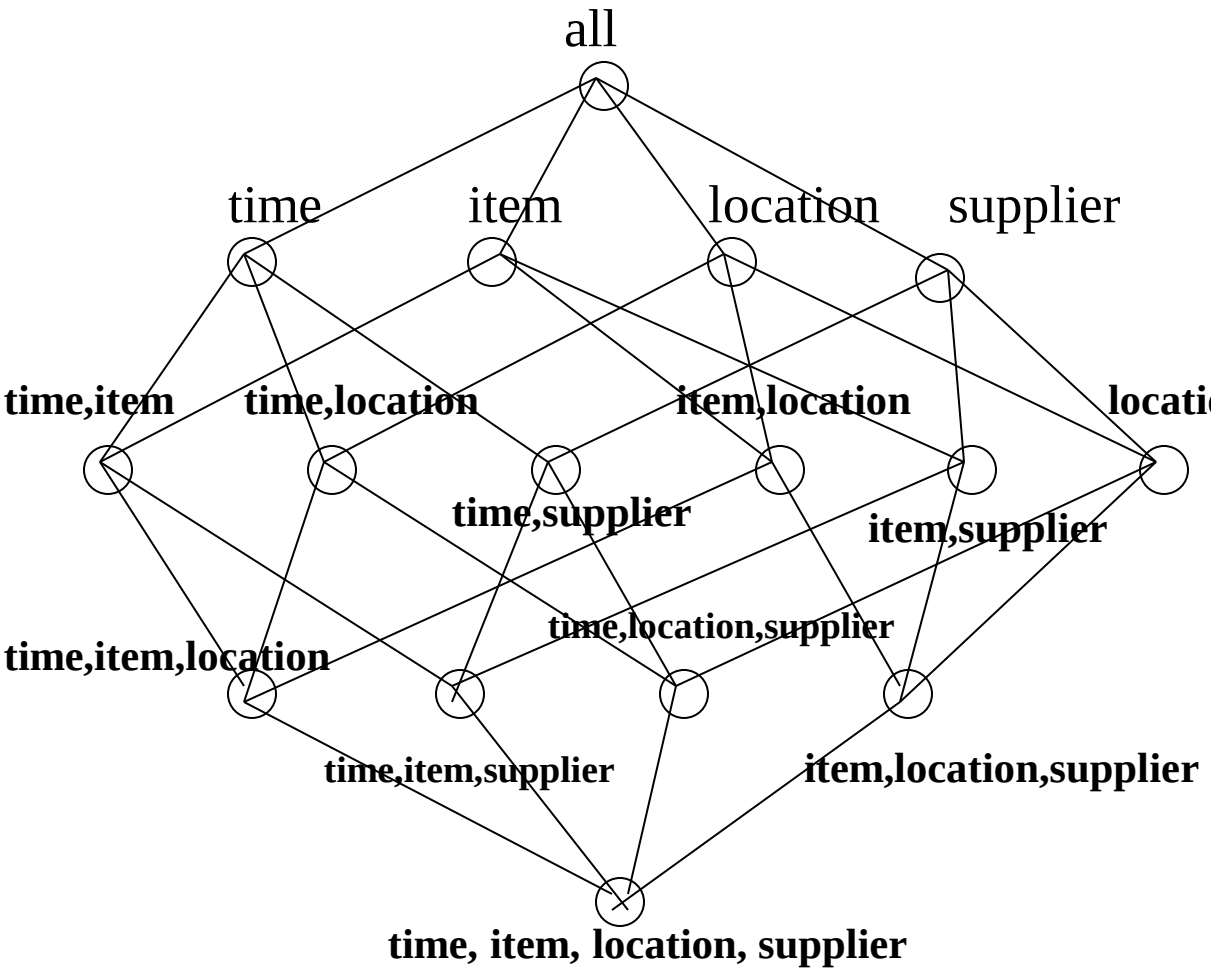


Figure: Sample E-R Diagram

- A data warehouse is based on a multidimensional data model which views data in the form of a data cube
- A data cube, such as sales, allows data to be modeled and viewed in multiple dimensions
  - Dimension tables, such as item (item_name, brand, type), or time(day, week, month, quarter, year)
  - Fact table contains measures (such as dollars_sold) and keys to each of the related dimension tables
- The lattice of cuboids forms a data cube.

Cube: A Lattice of Cuboids



0-D(apex) cuboid

1-D cuboids

2-D cuboids

3-D cuboids

4-D(base) cuboid

Dimensions and Measures

The database component of a data warehouse is described using a technique called dimensionality modeling.

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.

Fact 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.
- Because dimension tables contain records that describe facts, the fact table can be reduced to columns for dimension foreign keys and numeric fact values. Text and de-normalized data are typically not stored in the fact table.

- 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 Tables

- Dimension tables encapsulate the attributes associated with facts and separate these attributes into logically distinct groupings, such as time, geography, products, customers, and so forth.
- A dimension table may be used in multiple places if the data warehouse contains multiple fact tables or contributes data to data marts.
- The data in a dimension is usually hierarchical in nature.
- Hierarchies are determined by the business need to group and summarize data into usable information.
- For example, a time dimension often contains the hierarchy elements: (all time), Year, Quarter, Month, Day, or (all time), Year Quarter, Week, Day.

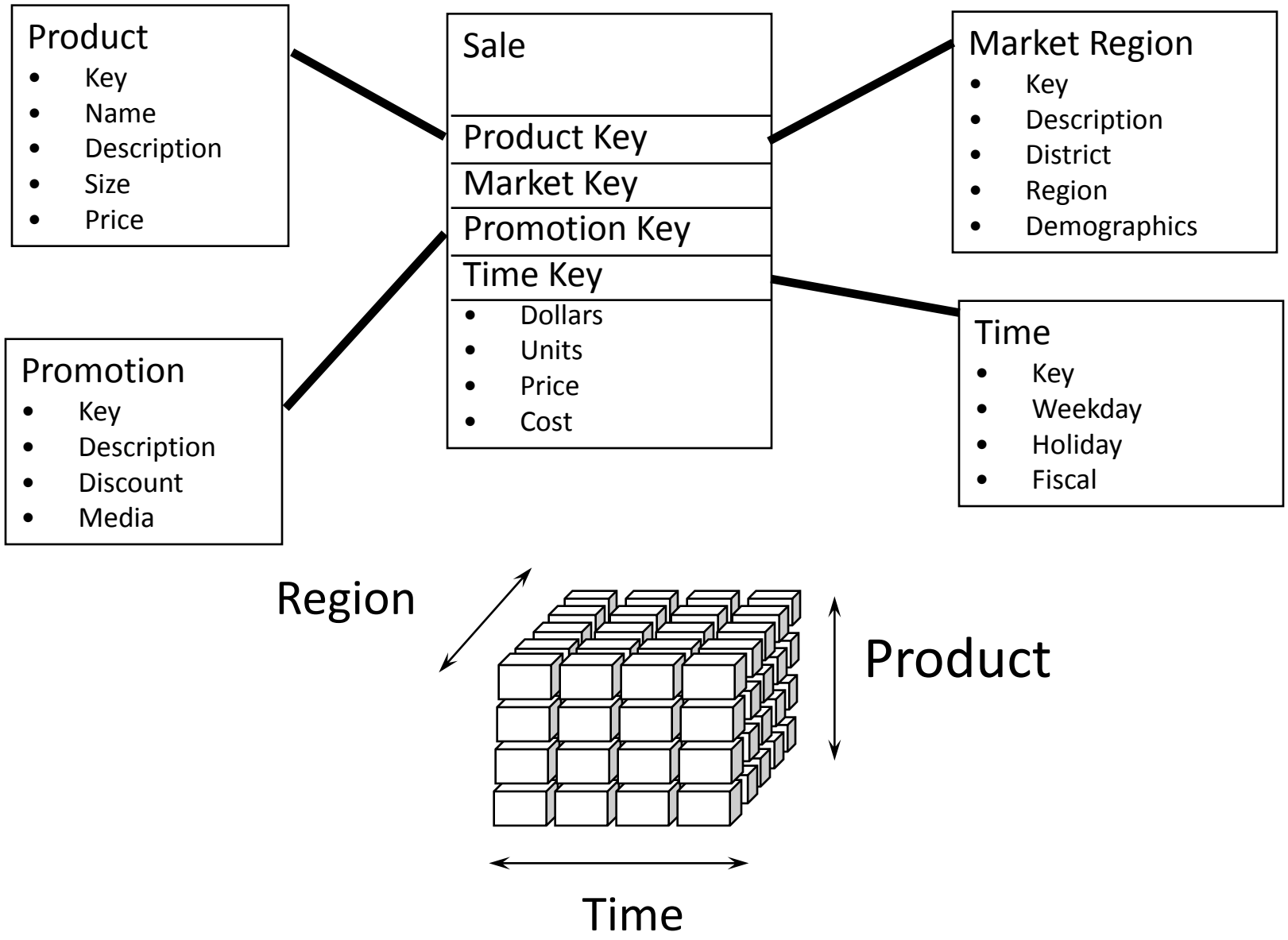


Figure: Dimensional Model

A Data Mining Query Language, DMQL: Language Primitives

Cube Definition (Fact Table)

define cube <cube_name> [<dimension_list>]: <measure_list>

Dimension Definition (Dimension Table)

define dimension <dimension_name> as
(<attribute_or_subdimension_list>)

Special Case (Shared Dimension Tables)

- First time as “cube definition”
- define dimension <dimension_name> as <dimension_name_first_time> in cube <cube_name_first_time>

Data Warehouse Schema

- A data warehouse, however, requires a concise, subject-oriented schema that facilitates on-line data processing (OLAP).
- 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.
- The major focus will be on the star schema which is commonly used in the design of many data warehouse.

Star Schema

- The star schema is a data modeling technique used to map multidimensional decision support into a relational database.
- Star schemas yield an easily implemented model for multidimensional data analysis while still preserving the relational structure of the operational database.
- Others name: star-join schema, data cube, data list, grid file and multi-dimension schema

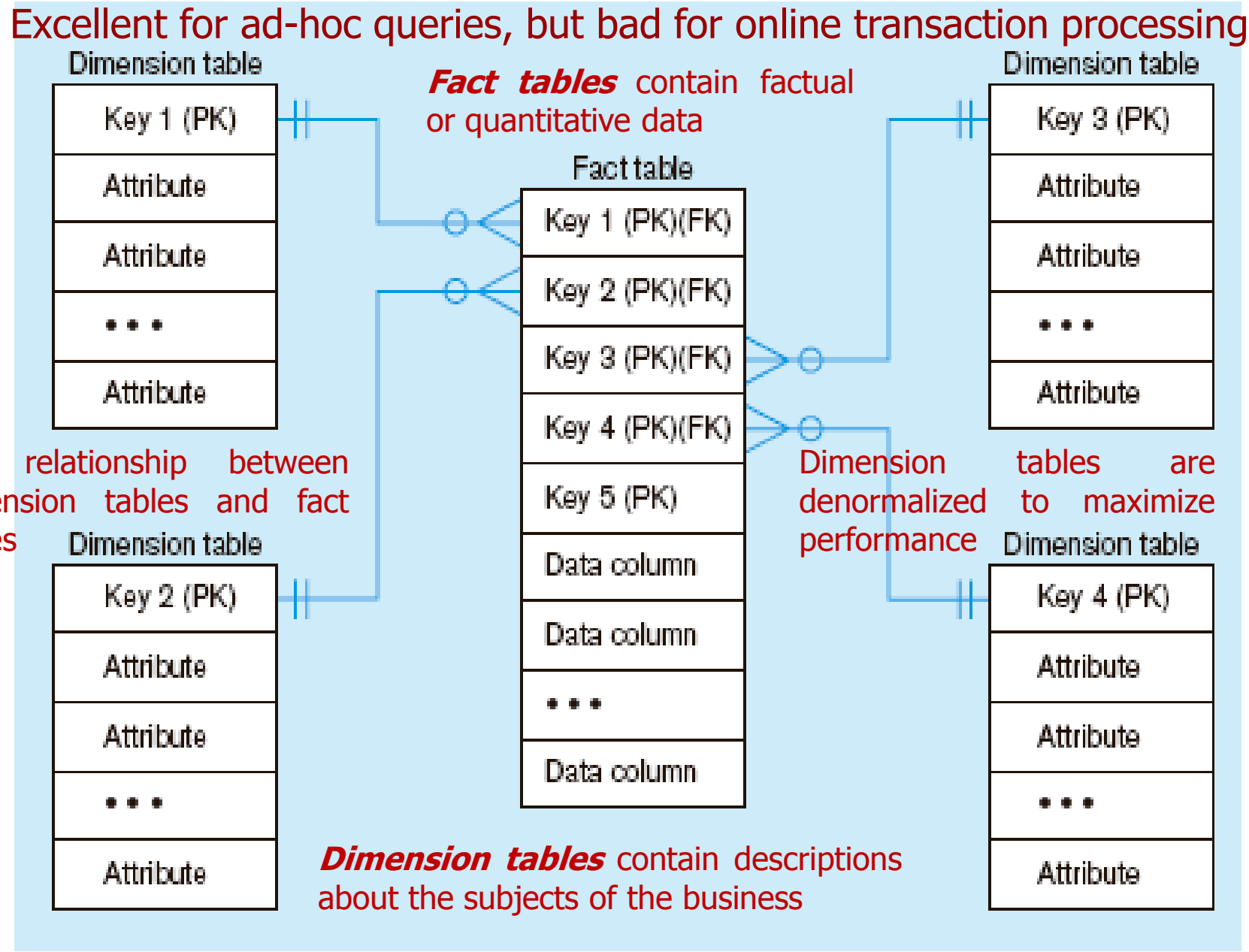


Figure: Components of a **star schema**

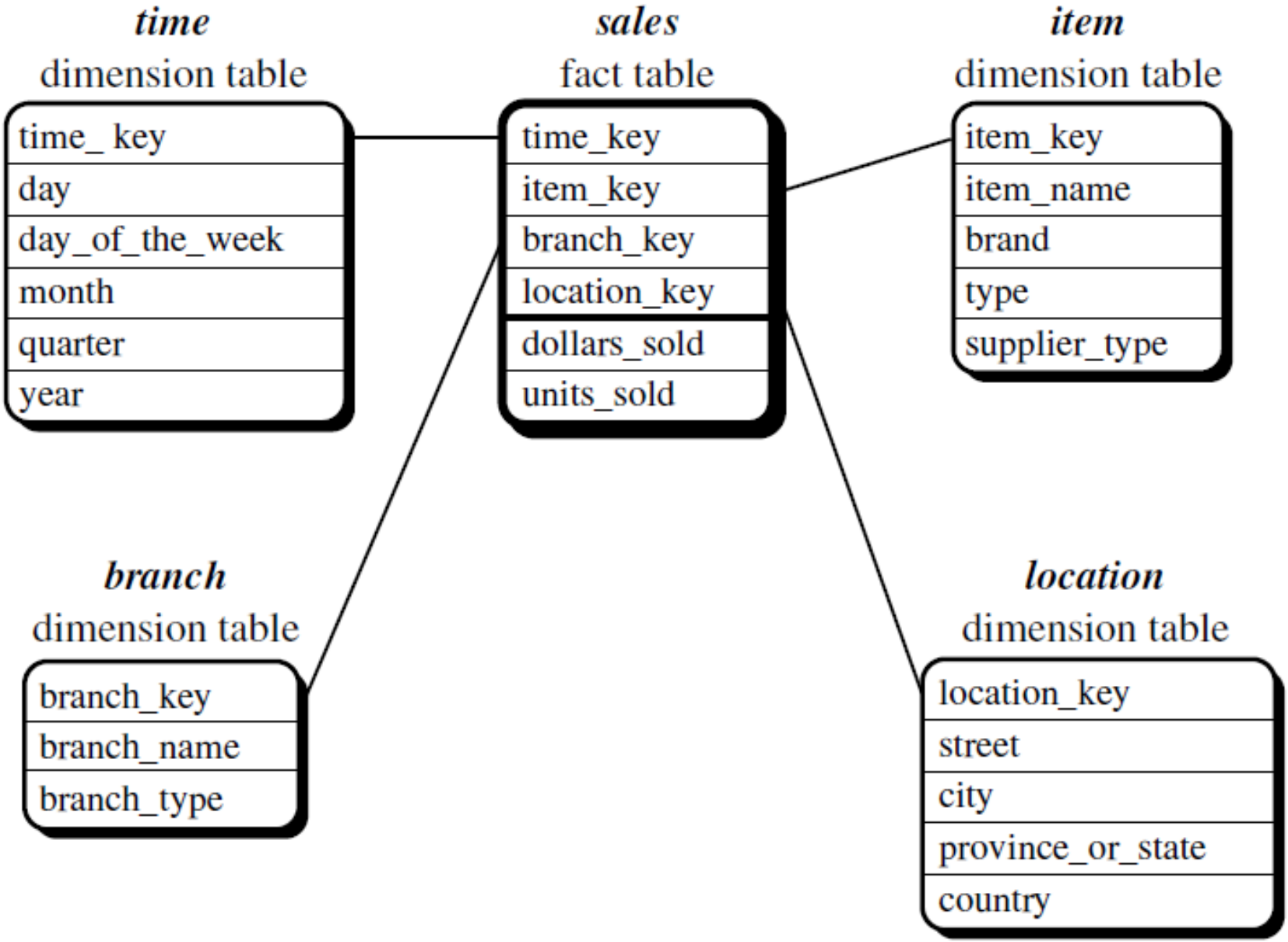


Figure: Star schema of a data warehouse for sales

- The schema contains a central fact table for *sales* that contains keys to each of the four dimensions, along with two measures: *dollars_sold, avg_sales,* and *units_sold*.
- To minimize the size of the fact table, dimension identifiers (such as *time key* and *item key*) are system-generated identifiers.
- Notice that in the star schema, each dimension is represented by only one table, and each table contains a set of attributes.
- For example, the *location* dimension table contains the attribute set {*location key, street, city, province or state, country*}

Defining a Star Schema in DMQL

define cube sales_star [time, item, branch, location]:

dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier_type)

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city, province_or_state, country)

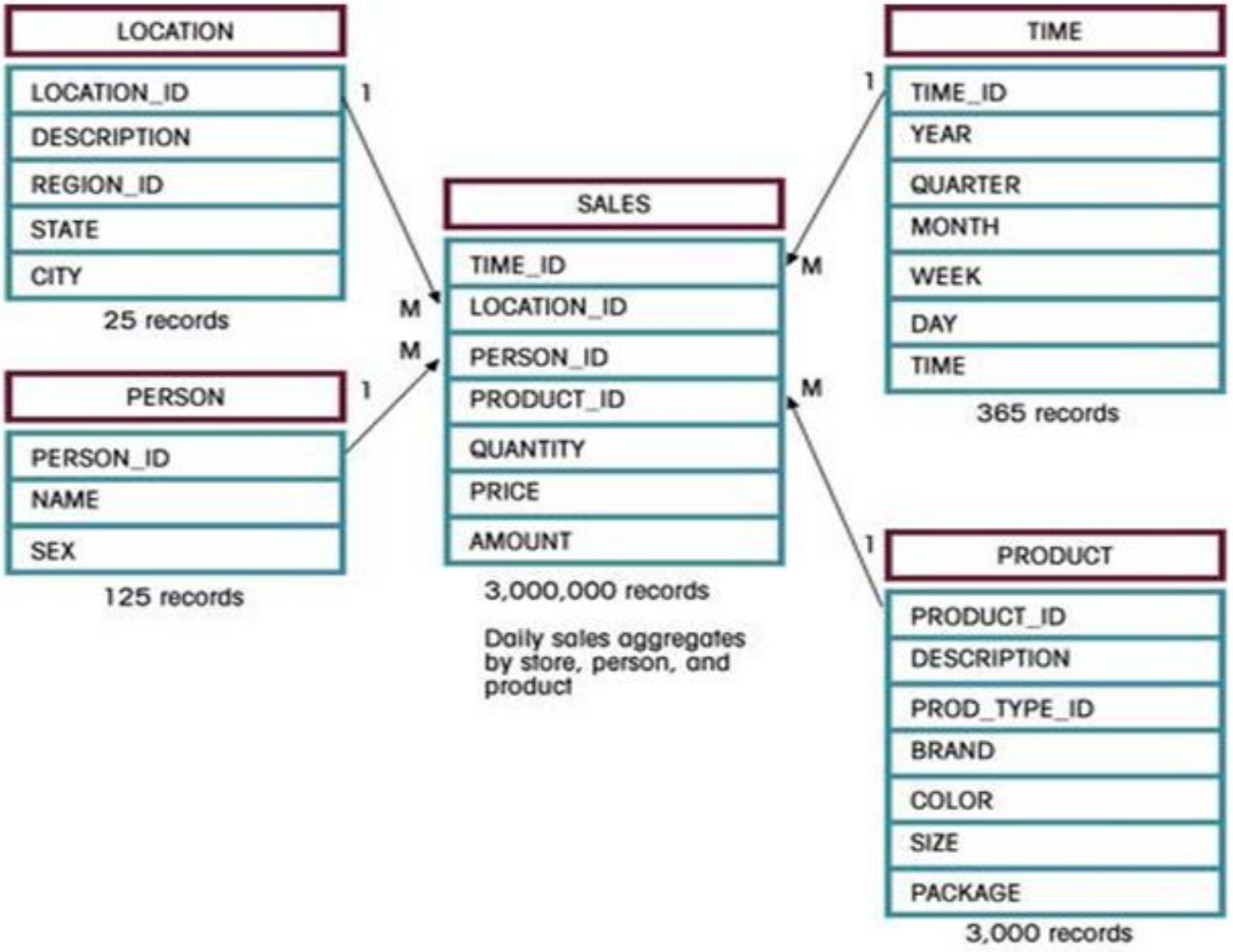


Figure: Star Schema for Sales

Advantages of Star Schema

- 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

Issues Regarding Star Schema

- Dimension table keys must be ***surrogate*** (non-intelligent and non-business related), because:
  - Keys may change over time
  - Length/format consistency
- Granularity of Fact Table–what level of detail do you want?
  - Transactional grain–finest level
  - Aggregated grain–more summarized
  - Finer grains → better ***market basket analysis*** capability
  - Finer grain → more dimension tables, more rows in fact table
- Duration of the database–how much history should be kept?
  - Natural duration–13 months or 5 quarters
  - Financial institutions may need longer duration
  - Older data is more difficult to source and cleanse

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.

Characteristics:

- Normalization of dimension tables
- Each hierarchical level has its own table
- less memory space is required
- a lot of joins can be required if they involve attributes in secondary dimension tables

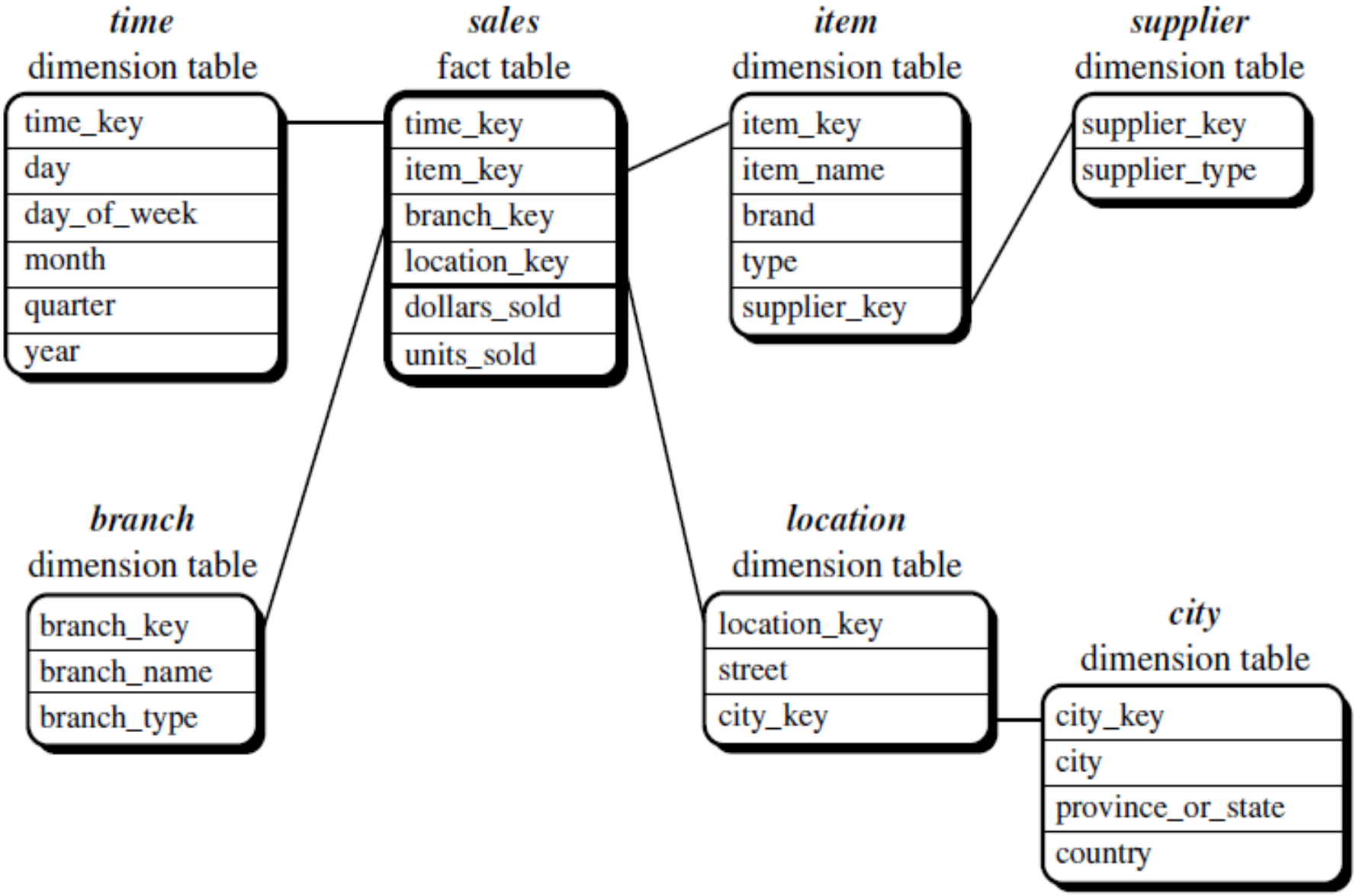


Figure: Snowflake schema of a data warehouse for sales

Defining a Snowflake Schema in DMQL

define cube sales_snowflake [time, item, branch, location]:

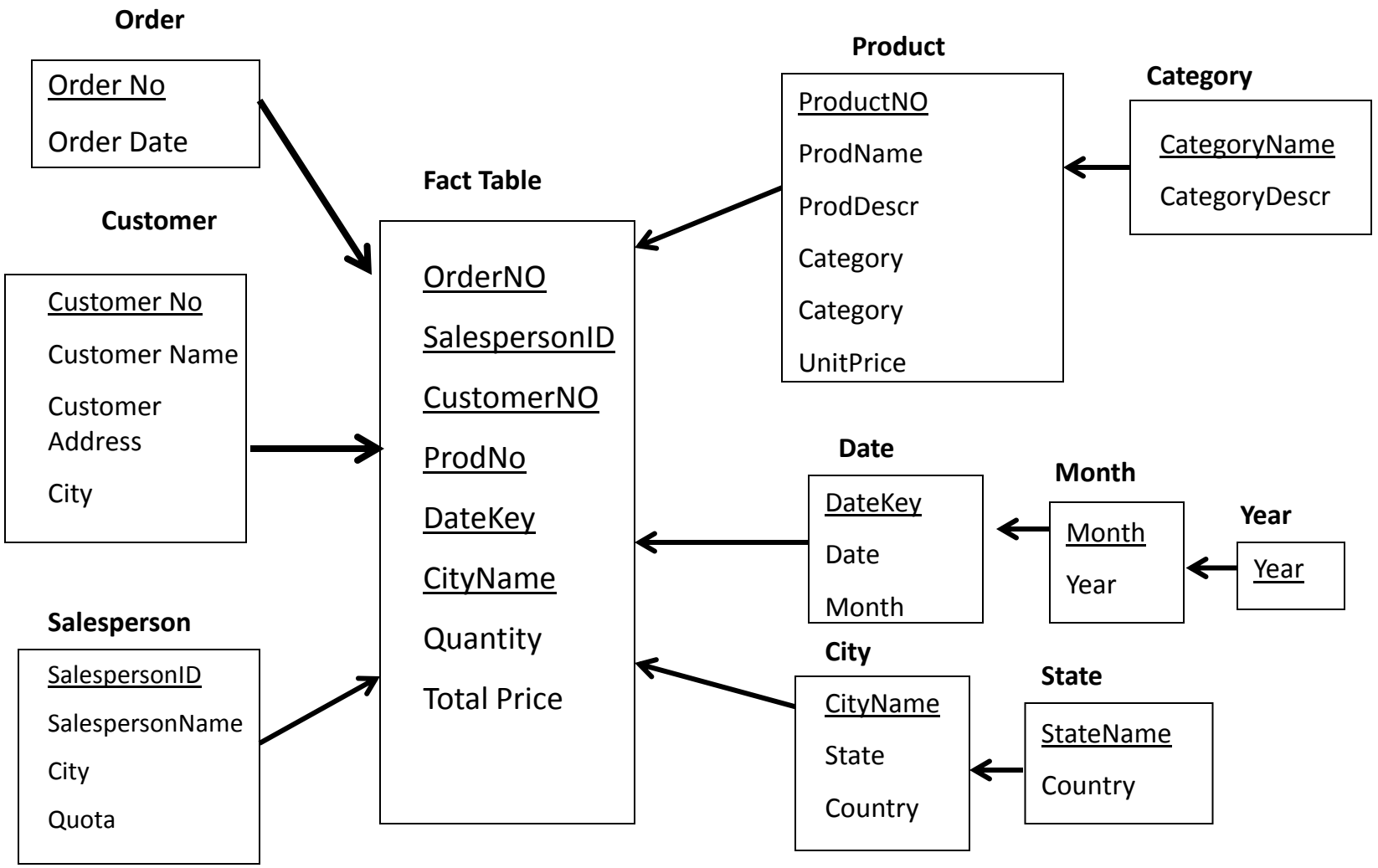
dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars), units_sold = count(*)

define dimension time as (time_key, day, day_of_week, month, quarter, year)

define dimension item as (item_key, item_name, brand, type, supplier(supplier_key, supplier_type))

define dimension branch as (branch_key, branch_name, branch_type)

define dimension location as (location_key, street, city(city_key, province_or_state, country))


Order
Order No
Order Date
Customer
Customer No
Customer Name
Customer Address
City
Salesperson
SalespersonID
SalespersonName
City
Quota
Fact Table
OrderNO
SalespersonID
CustomerNO
ProdNo
DateKey
CityName
Quantity
Total Price
Product
ProductNO
ProdName
ProdDescr
Category
Category
UnitPrice
Category
CategoryName
CategoryDescr
Date
DateKey
Date
Month
Month
Month
Year
Year
Year
City
CityName
State
Country
State
StateName
Country

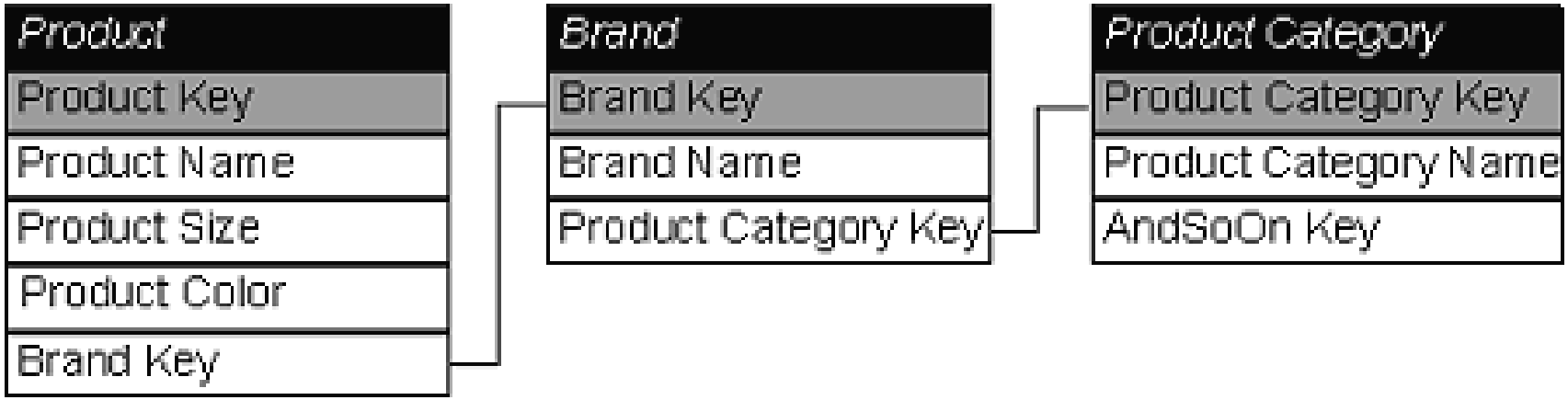


Figure: Snowflake Schema

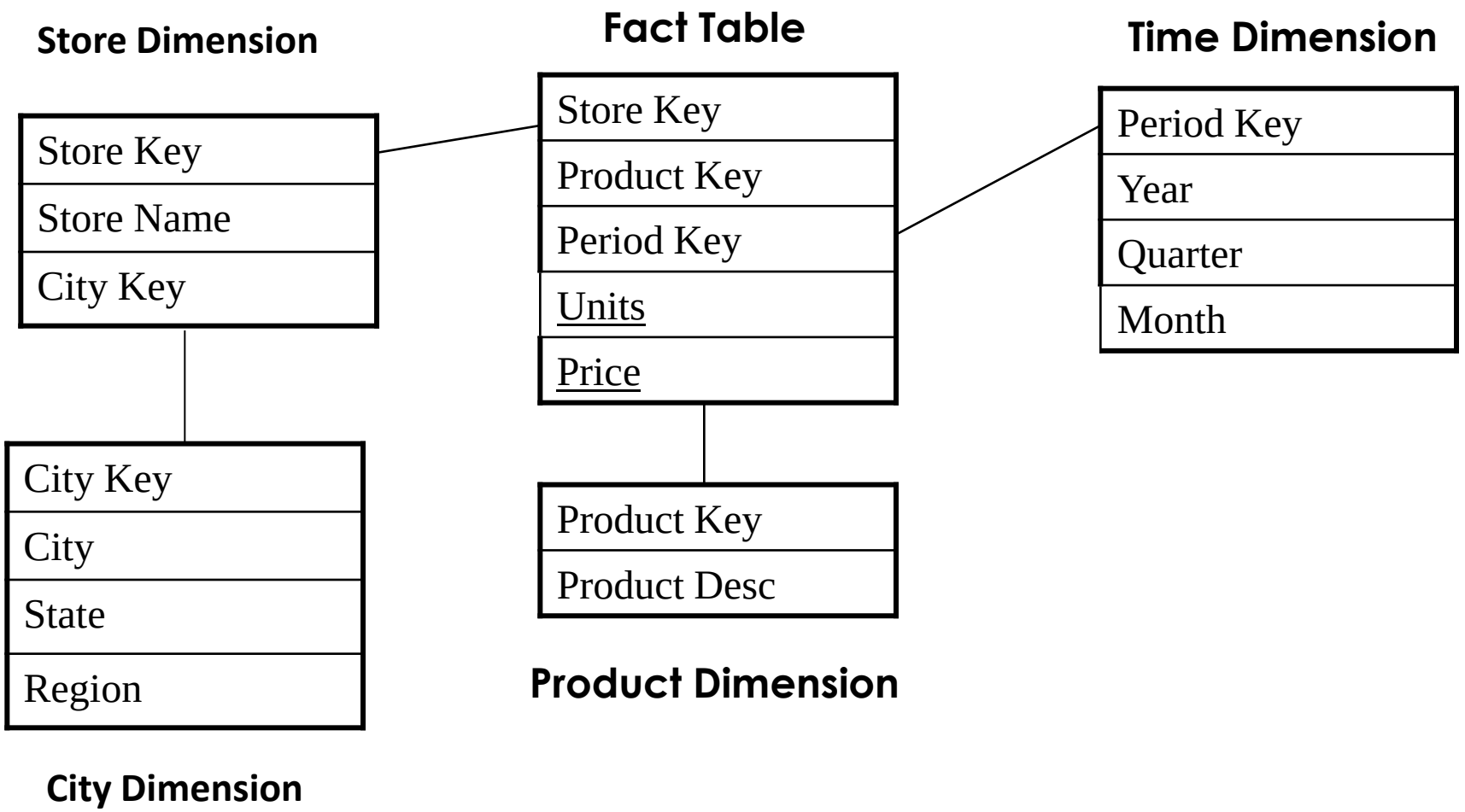


Figure: Snowflake Schema

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.
- It may be better to create a star version of the snowflaked dimension for presentation to the users

Fact-Constellation Schema

- Multiple fact tables share dimension tables.
- This schema is viewed as collection of stars hence called as galaxy schema or fact constellation.
- Sophisticated application requires such schema.
- In the Fact Constellations, aggregate tables are created separately from the detail, therefore, it is impossible to pick up, for example, Store detail when querying the District Fact Table.
- 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.

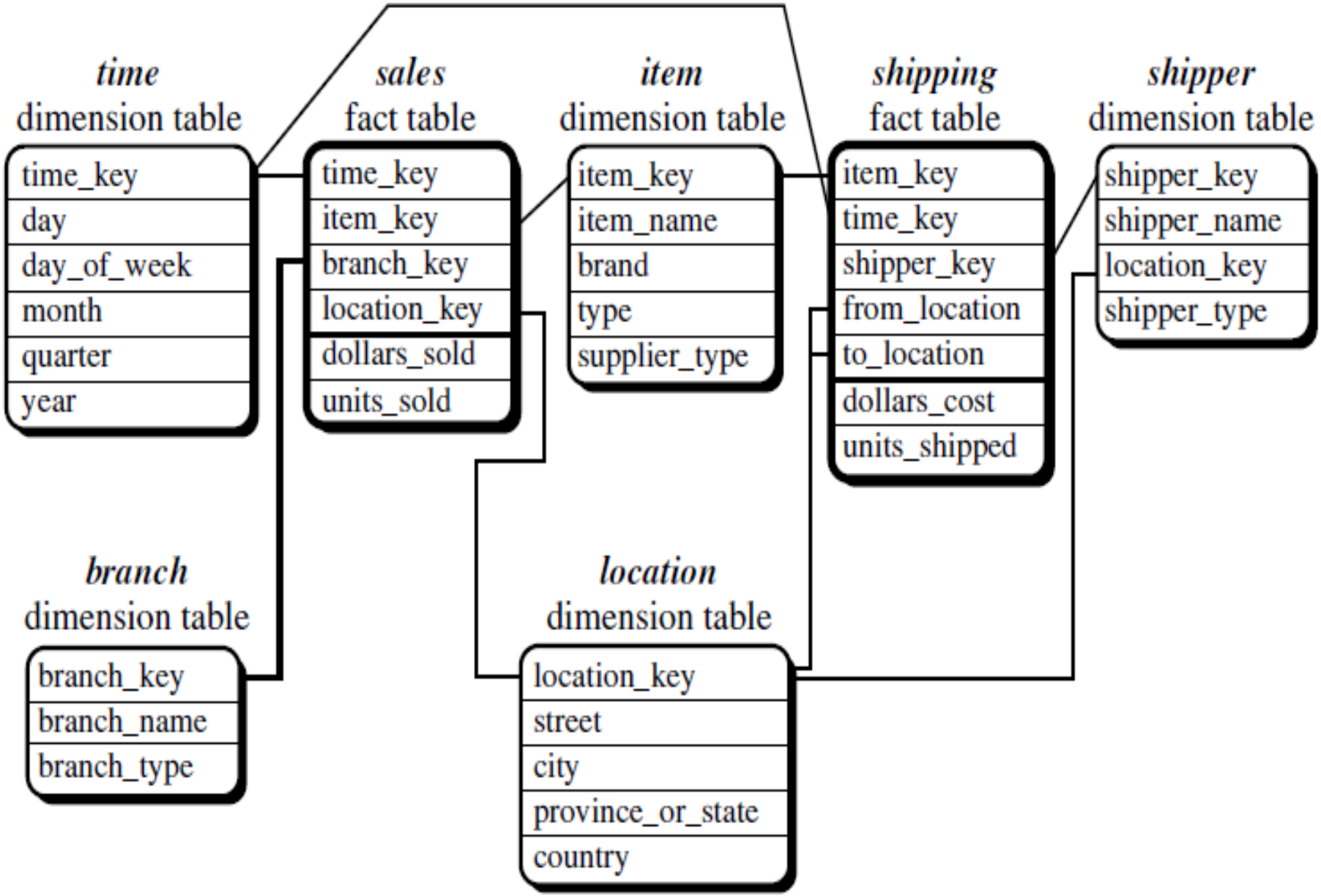
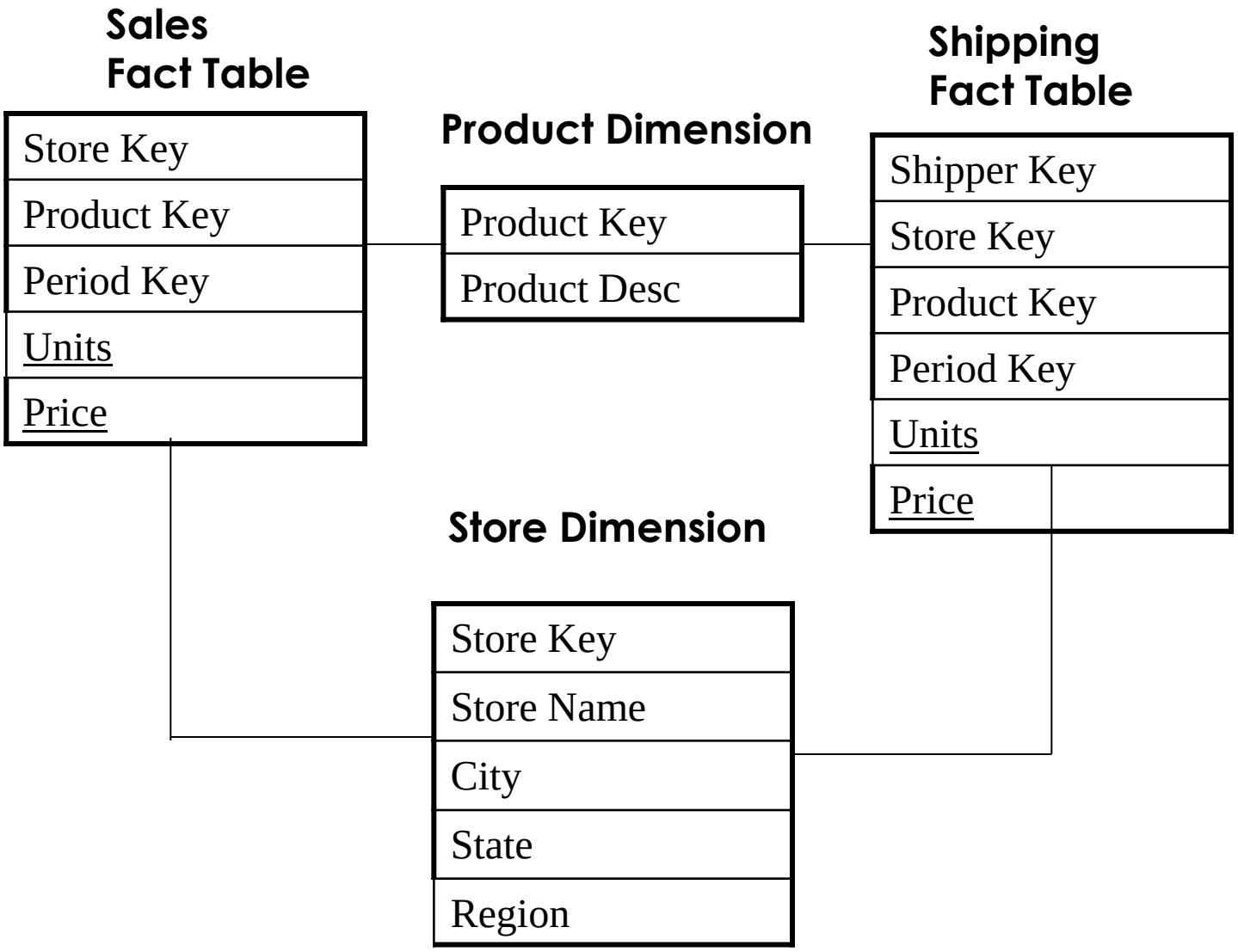


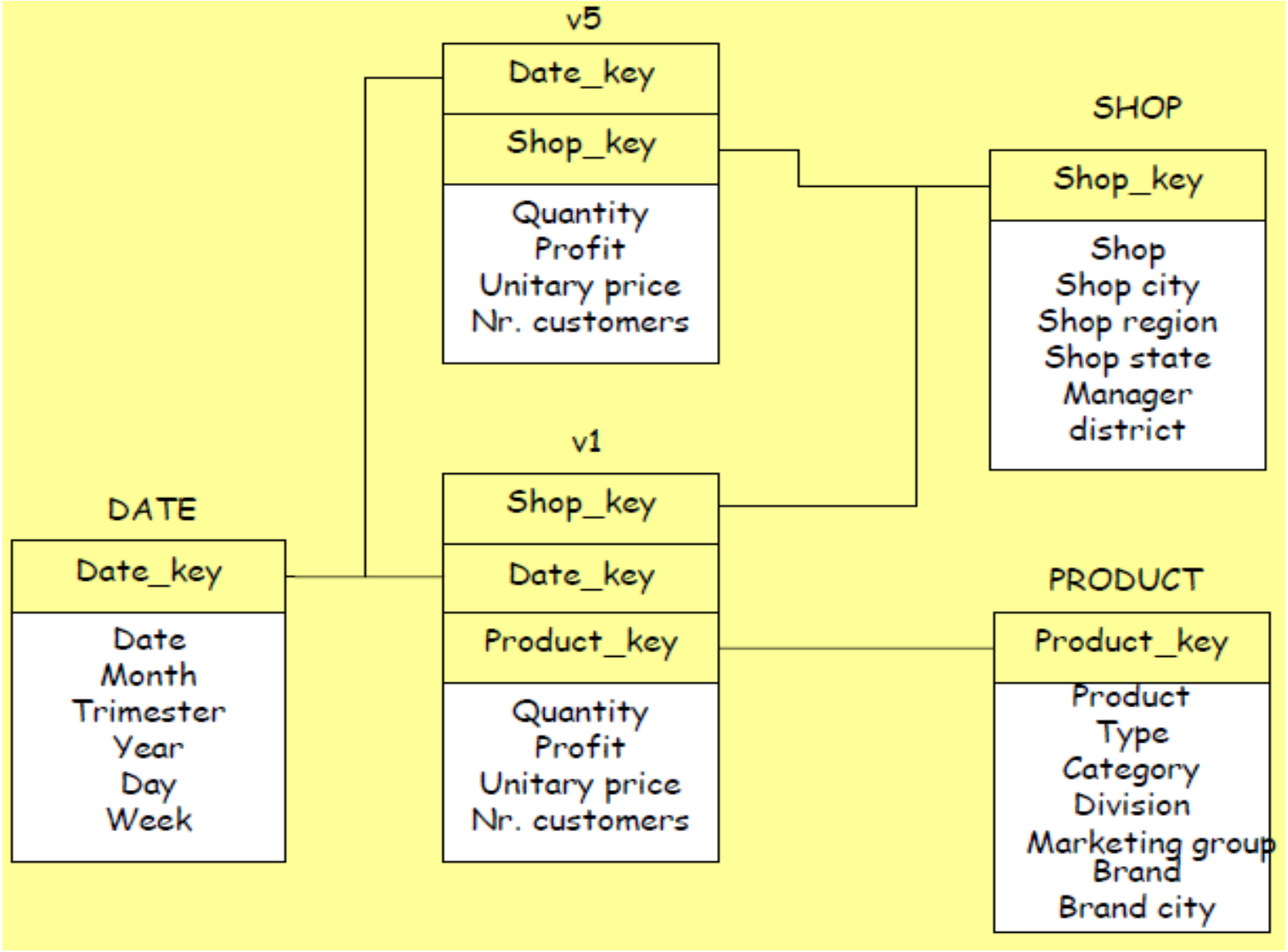
Figure: Fact constellation schema of a data warehouse for sales and shipping

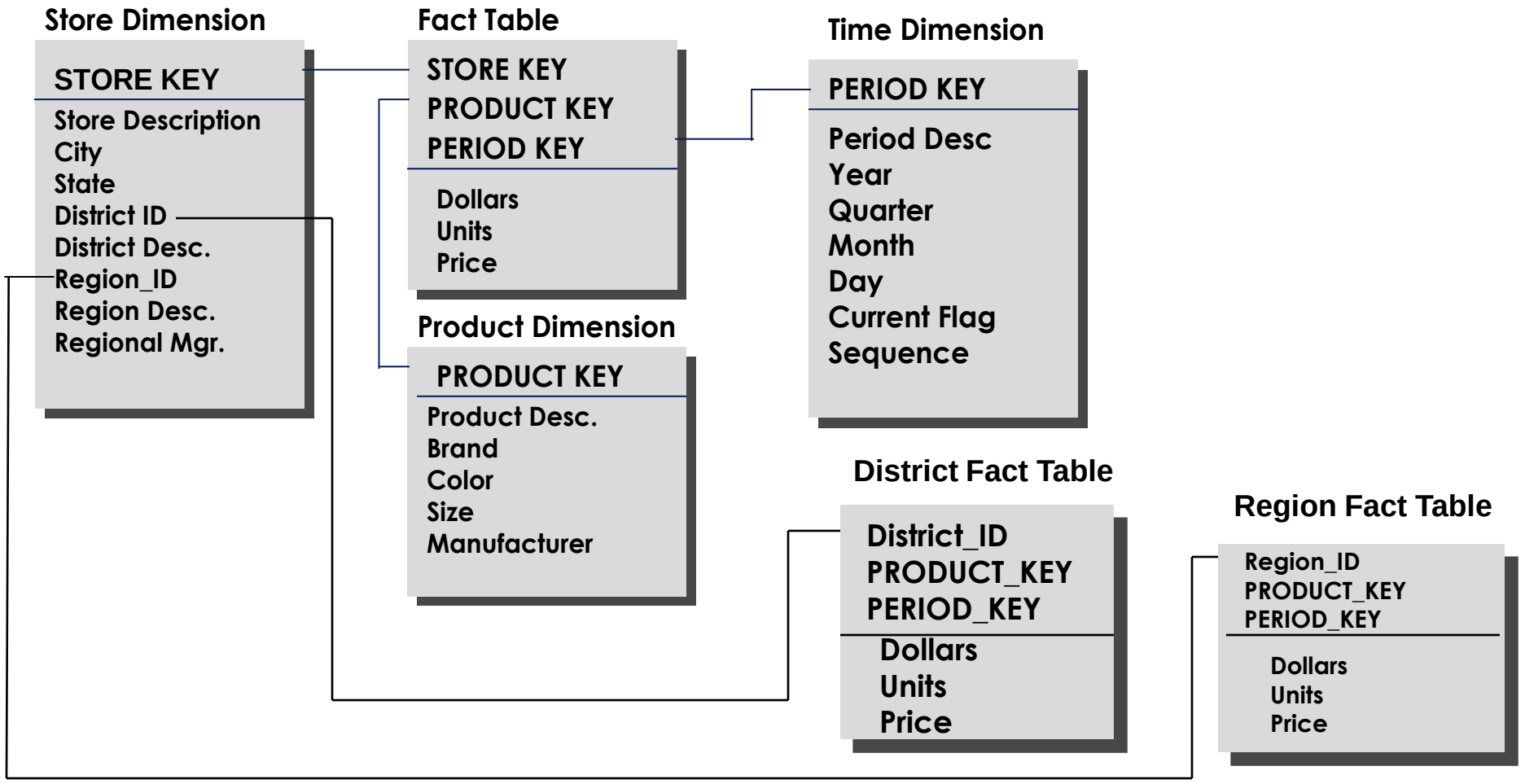
Defining a Fact Constellation in DMQL

```
define cube sales [time, item, branch, location]:
        dollars_sold = sum(sales_in_dollars), avg_sales = avg(sales_in_dollars),
          units_sold = count(*)
define dimension time as (time_key, day, day_of_week, month, quarter, year)
define dimension item as (item_key, item_name, brand, type, supplier_type)
define dimension branch as (branch_key, branch_name, branch_type)
define dimension location as (location_key, street, city, province_or_state,
    country)
define cube shipping [time, item, shipper, from_location, to_location]:
        dollar_cost = sum(cost_in_dollars), unit_shipped = count(*)
define dimension time as time in cube sales
define dimension item as item in cube sales
define dimension shipper as (shipper_key, shipper_name, location as location in
    cube sales, shipper_type)
define dimension from_location as location in cube sales
define dimension to_location as location in cube sales
```

More Examples on Fact-Constellation Schema




v5
Date_key
Shop_key
Quantity
Profit
Unitary price
Nr. customers
SHOP
Shop_key
Shop
Shop city
Shop region
Shop state
Manager
district
v1
Shop_key
Date_key
Product_key
Quantity
Profit
Unitary price
Nr. customers
DATE
Date_key
Date
Month
Trimester
Year
Day
Week
PRODUCT
Product_key
Product
Type
Category
Division
Marketing group
Brand
Brand city


Store Dimension
STORE KEY
Store Description
City
State
District ID
District Desc.
Region_ID
Region Desc.
Regional Mgr.
Fact Table
STORE KEY
PRODUCT KEY
PERIOD KEY
Dollars
Units
Price
Time Dimension
PERIOD KEY
Period Desc
Year
Quarter
Month
Day
Current Flag
Sequence
Product Dimension
PRODUCT KEY
Product Desc.
Brand
Color
Size
Manufacturer
District Fact Table
District_ID
PRODUCT_KEY
PERIOD_KEY
Dollars
Units
Price
Region Fact Table
Region_ID
PRODUCT_KEY
PERIOD_KEY
Dollars
Units
Price

Multidimensional Data Model

Data warehouses and OLAP tools are based on a multidimensional data model. This model views data in the form of a *data cube*.

"What is a data cube?" A data cube allows data to be modeled and viewed in multiple dimensions. It is defined by dimensions and facts.

In general terms, dimensions are the perspectives or entities with respect to which an organization wants to keep records.

Each dimension may have a table associated with it, called a dimension table, which further describes the dimension.

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.

	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

Table: A 2-D view of sales data 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 = "Chicago"				location = "New York"				location = "Toronto"				location = "Vancouver"			
	item				item				item				item			
	home				home				home				home			
time	ent.	comp.	phone	sec.	ent.	comp.	phone	sec.	ent.	comp.	phone	sec.	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

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

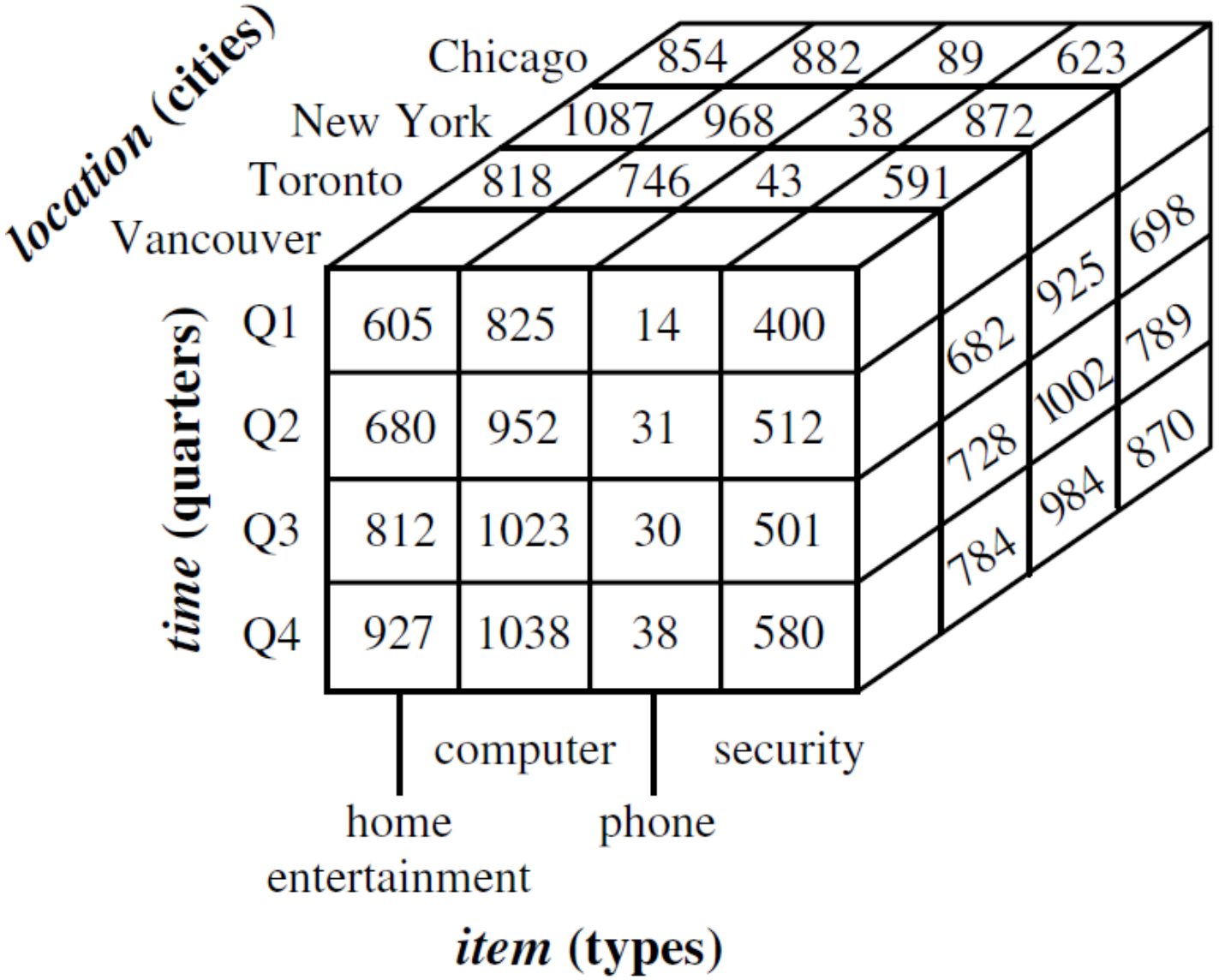


Figure: A 3-D data cube representation of the data in the table above, according to the dimensions *time*, *item*, and *location*. The measure displayed is *dollars sold* (in thousands).

Question?

Suppose that we would now like to view our sales data with an additional fourth dimension, such as *supplier*.

What should we do??

Any Solution???

Solution!!

Viewing things in 4-D becomes tricky. However, we can think of a 4-D cube as being a series of 3-D cubes as shown below:

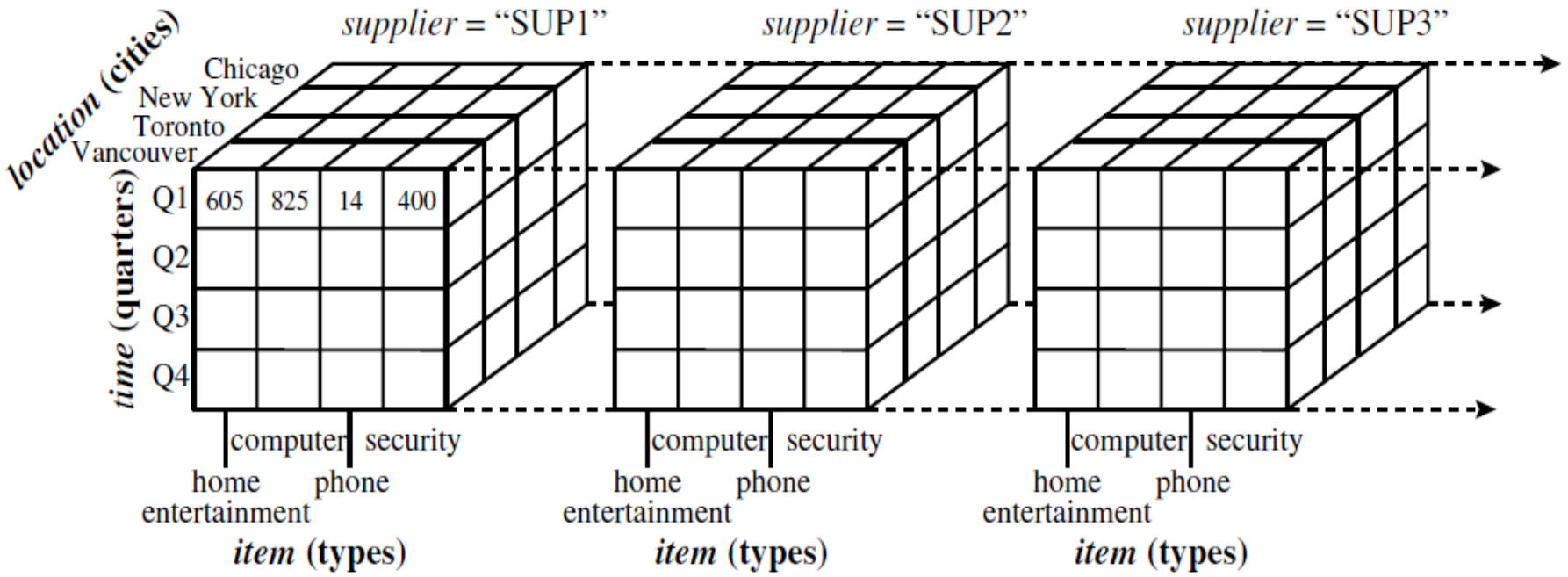


Figure: A 4-D data cube representation of sales data, according to the dimensions *time*, *item*, *location*, and *supplier*. The measure displayed is *dollars sold* (in thousands). For improved readability, only some of the cube values are shown.

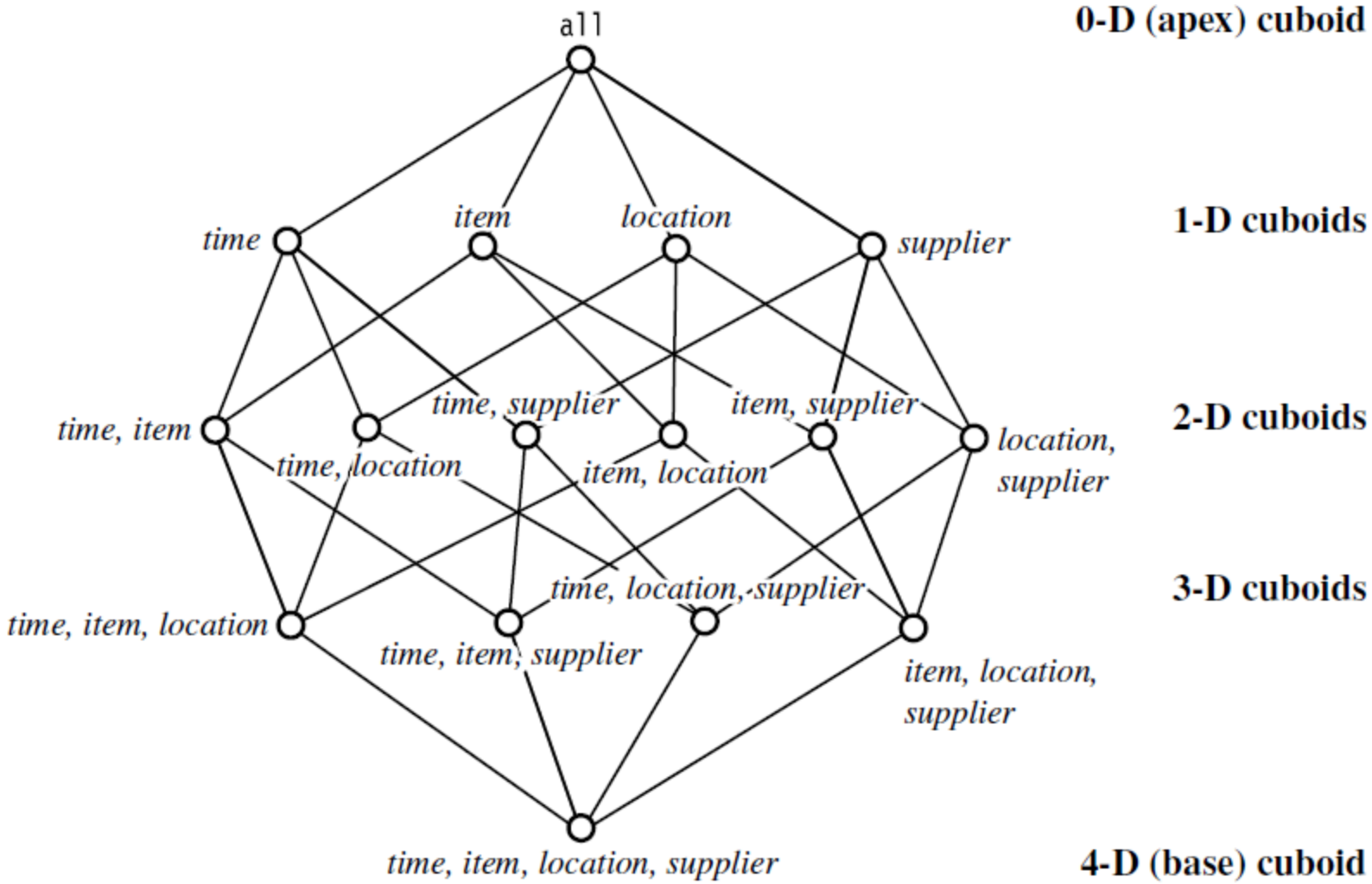


Figure: Lattice of cuboids, making up a 4-D data cube for the dimensions *time*, *item*, *location*, and *supplier*. Each cuboid represents a different degree of summarization.

Measures: Their Categorization and Computation

"How are measures computed?"

A data cube *measure* is a numerical function that can be evaluated at each point in the data cube space.

A measure value is computed for a given point by aggregating the data corresponding to the respective dimension-value pairs defining the given point.

Measures can be organized into three categories (i.e., distributive, algebraic, holistic), based on the kind of aggregate functions used.

Distributive Measure

- A measure is *distributive* if it is obtained by applying a distributive aggregate function.
- An aggregate function is *distributive* if it can be computed in a distributed manner.
- Example: count(), sum(), min(), max().

Algebraic Measure

- A measure is *algebraic* if it is obtained by applying an algebraic aggregate function.
- An aggregate function is *algebraic* if it can be computed by an algebraic function.
- Example: avg(), min_N(), standard_deviation().

Holistic Measure

- A measure is *holistic* if it is obtained by applying a holistic aggregate function.
- An aggregate function is *holistic* if there is no constant bound on the storage size needed to describe a sub-aggregate.
- Example: median(), mode(), rank().

Concept Hierarchies

- A concept hierarchy defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts.

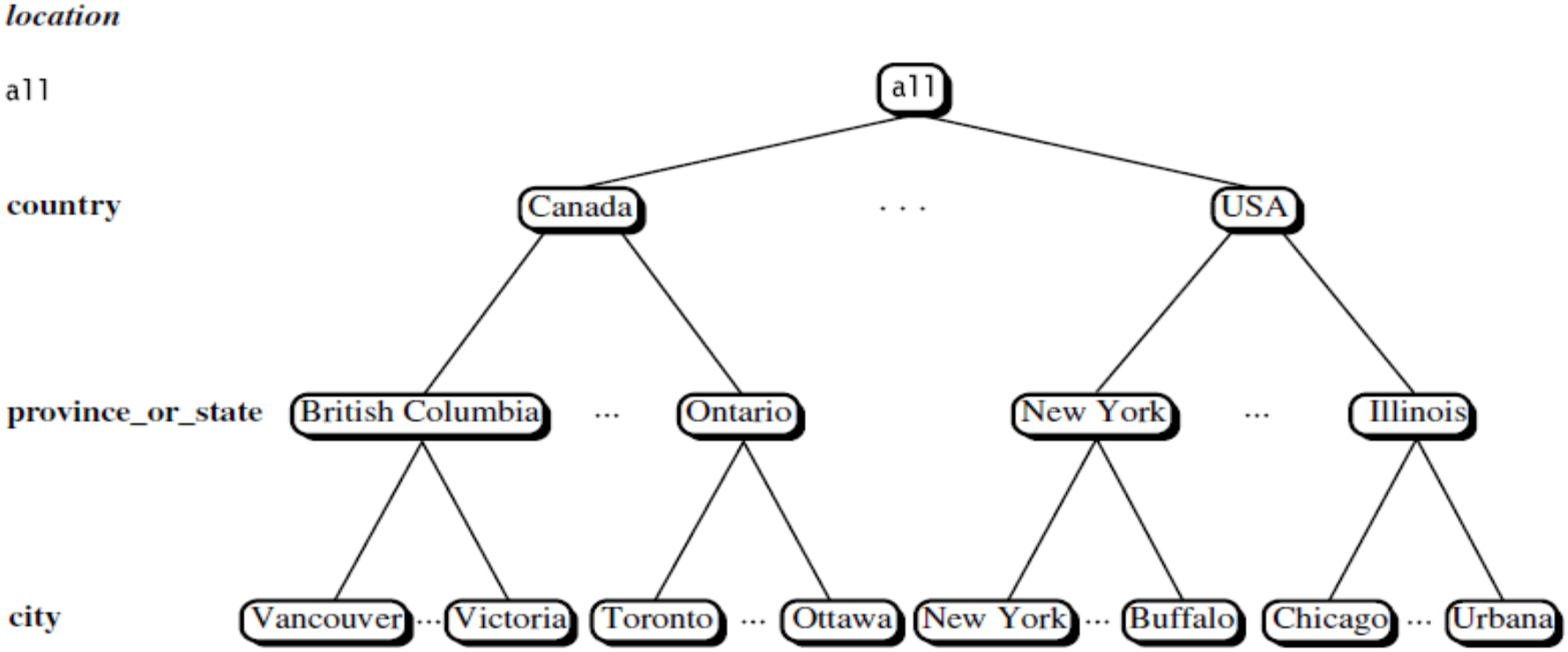


Figure: A concept hierarchy for the dimension *location*.

In the above figure we have considered a concept hierarchy for the dimension *location*. City values for *location* include Vancouver, Toronto, New York, and Chicago. Each city, however, can be mapped to the province or state to which it belongs.

For example, Vancouver can be mapped to British Columbia, and Chicago to Illinois. The provinces and states can in turn be mapped to the country to which they belong, such as Canada or the USA.

These mappings form a concept hierarchy for the dimension *location*, mapping a set of low-level concepts (i.e., cities) to higher-level, more general concepts (i.e., countries).

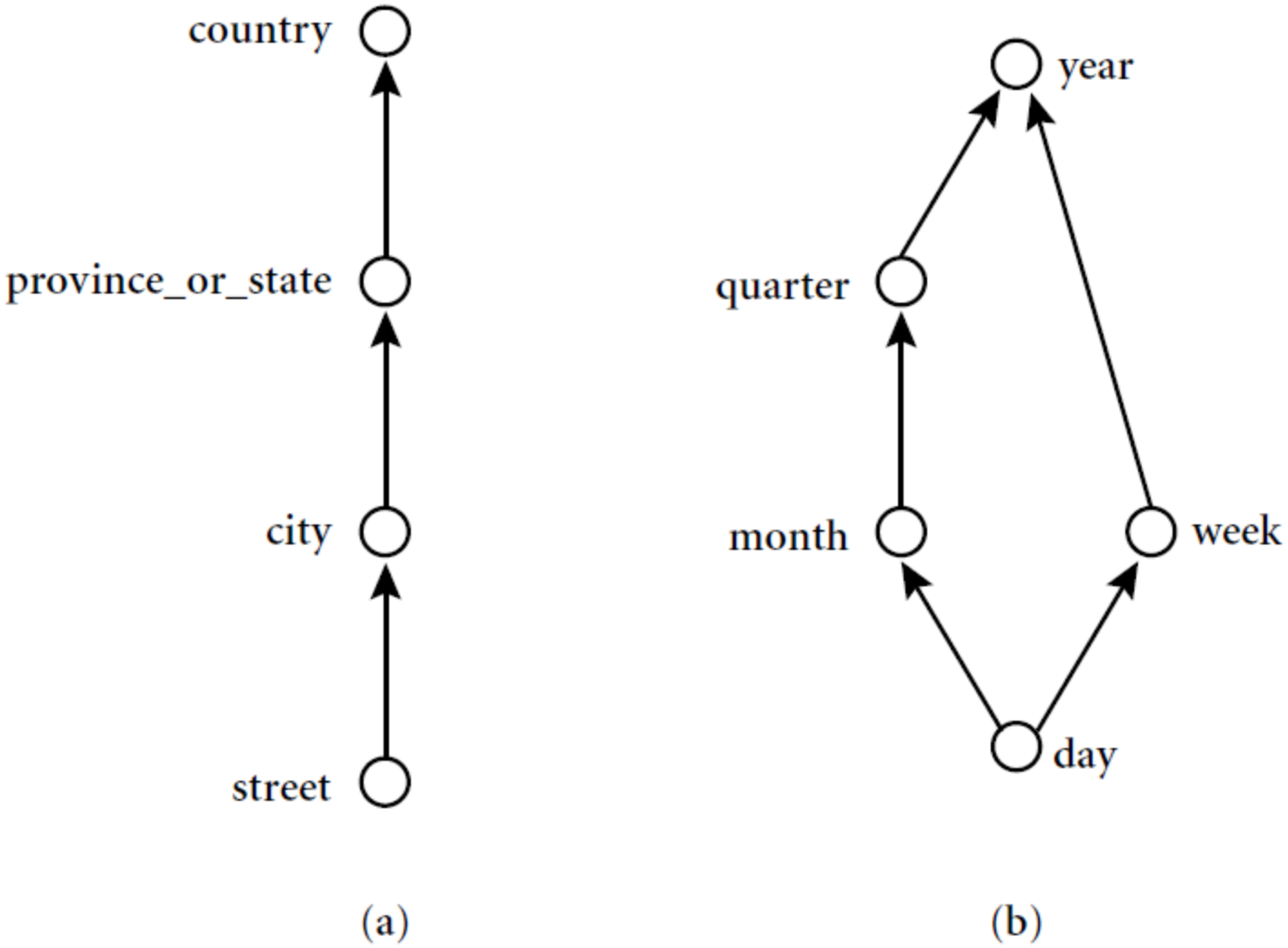


Figure: Hierarchical and lattice structures of attributes in warehouse dimensions: (a) a hierarchy for *location*; (b) a lattice for *time*.

Many concept hierarchies are implicit within the database schema. For example, suppose that the dimension *location* is described by the attributes *number, street, city, province or state, zip code*, and *country*.

These attributes are related by a total order, forming a concept hierarchy such as "*street* < *city* < *province or state* < *country*". This hierarchy is shown in the figure (a) above.

Alternatively, the attributes of a dimension may be organized in a partial order, forming a lattice.

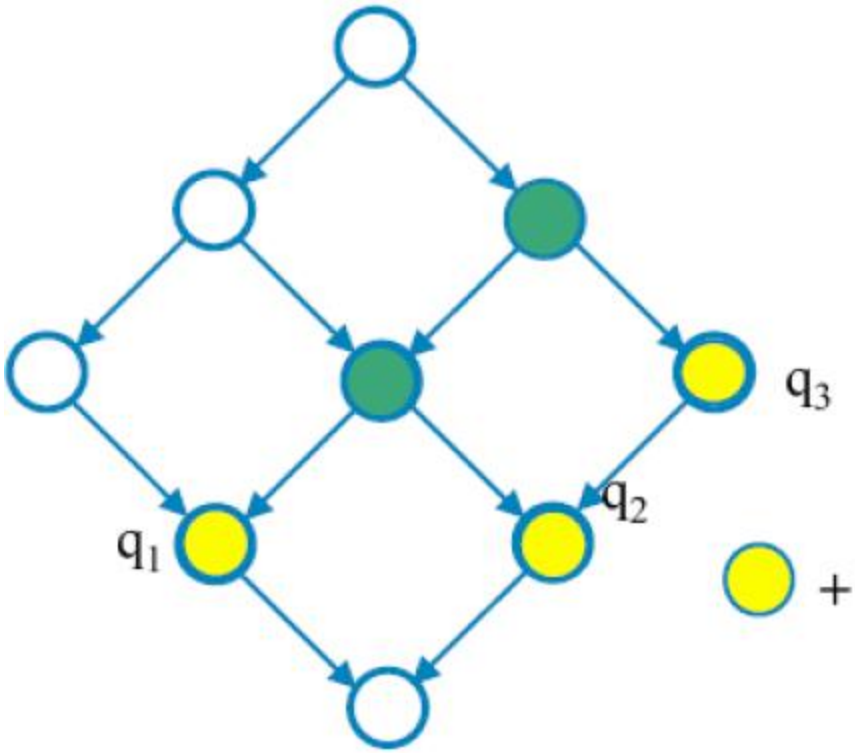
An example of a partial order for the *time* dimension based on the attributes *day, week, month, quarter*, and *year* is "*day* < {*month* <*quarter; week*} < *year*".

This lattice structure is shown in the figure (b) as above.

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.

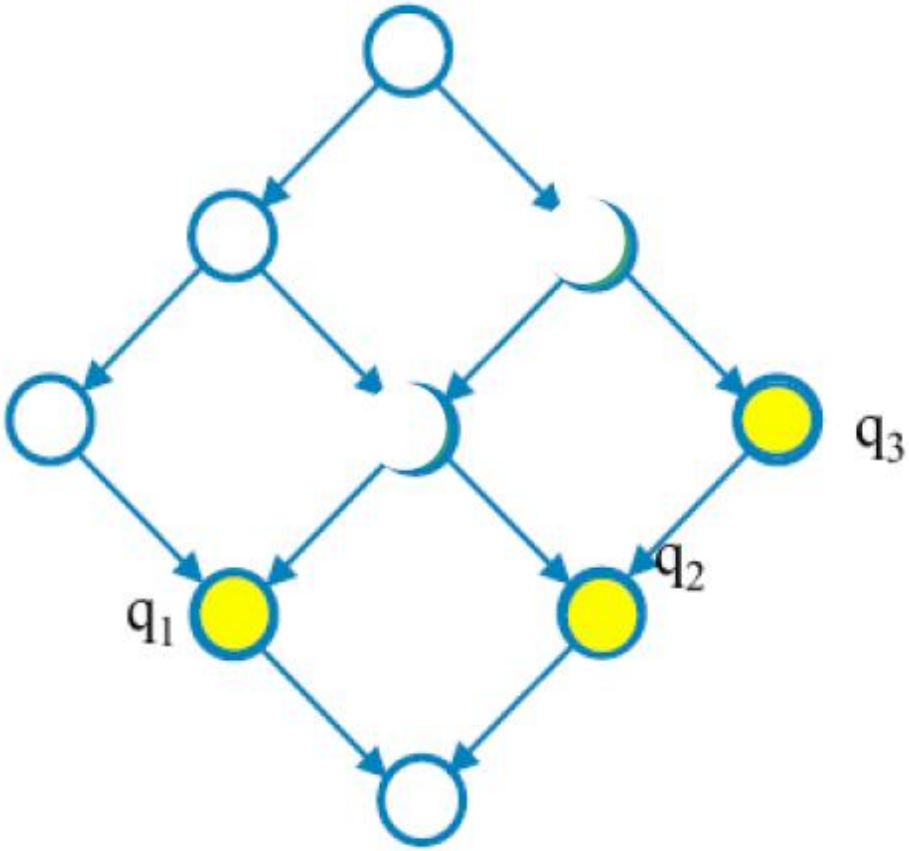
Materialized views can be best explained by Multidimensional lattice.

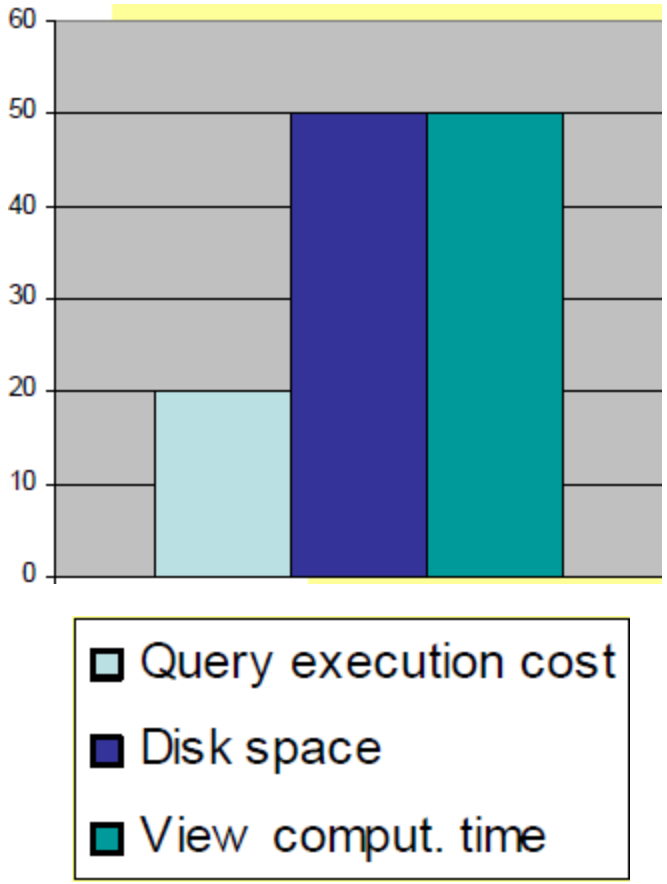


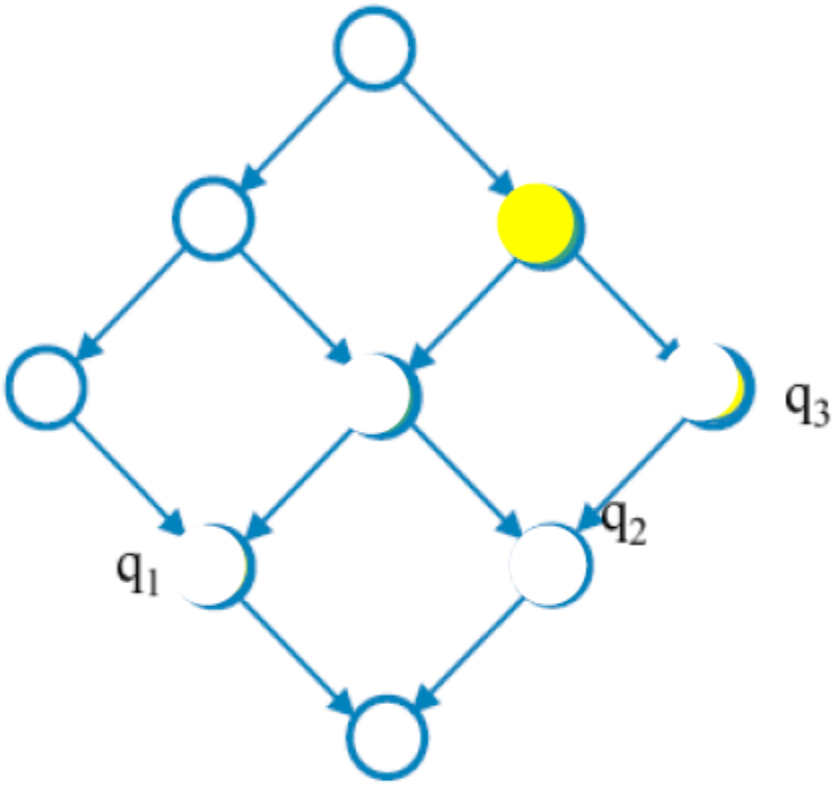
= **exact views:** They solve exactly the queries

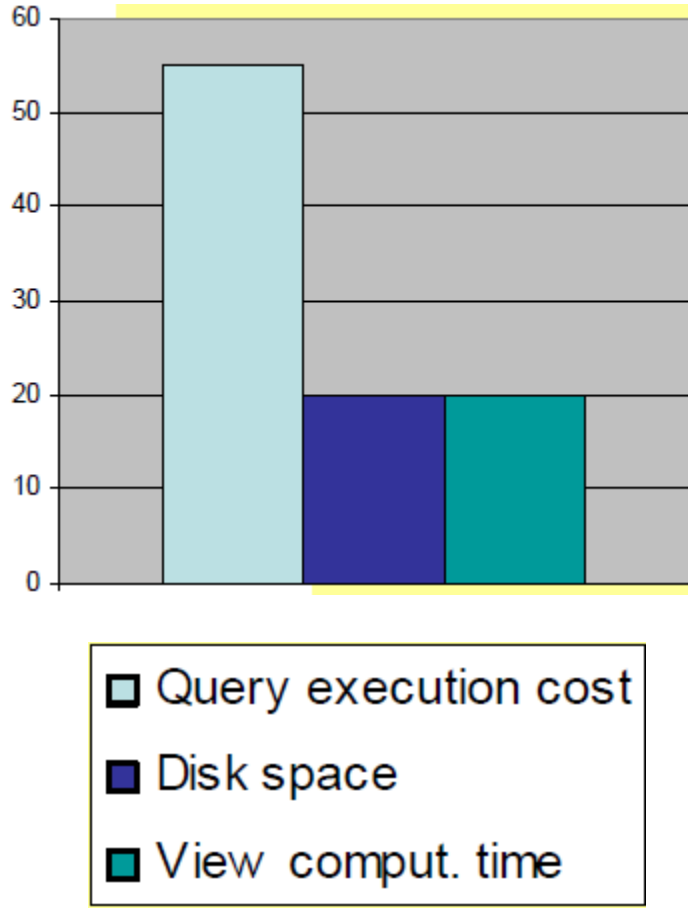
= **less aggregate views:** They solve more than one query

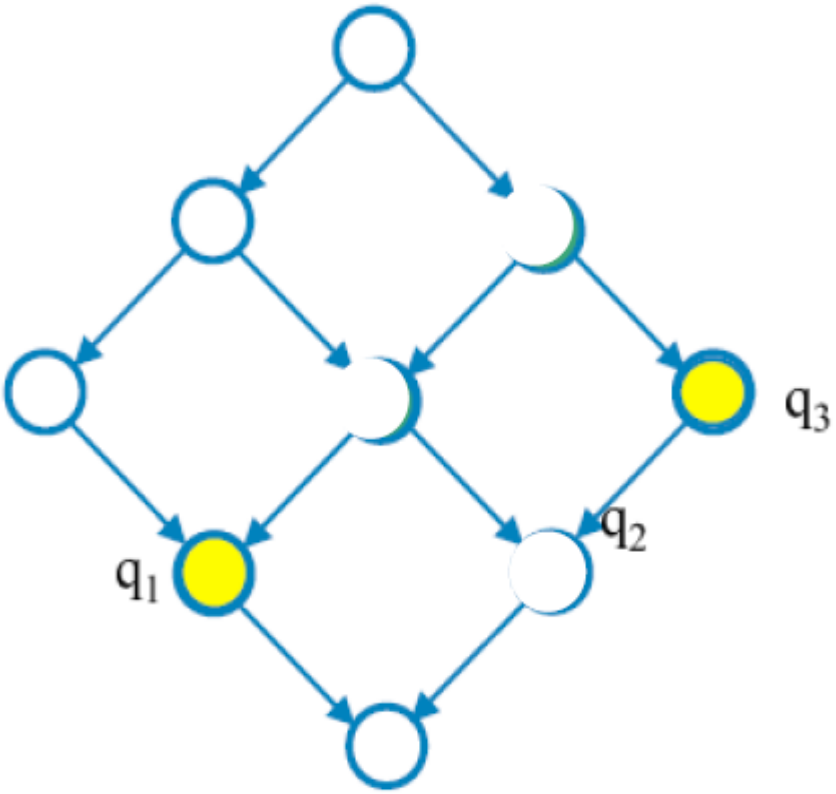
+ = **candidate views:** They could reduce elaboration costs

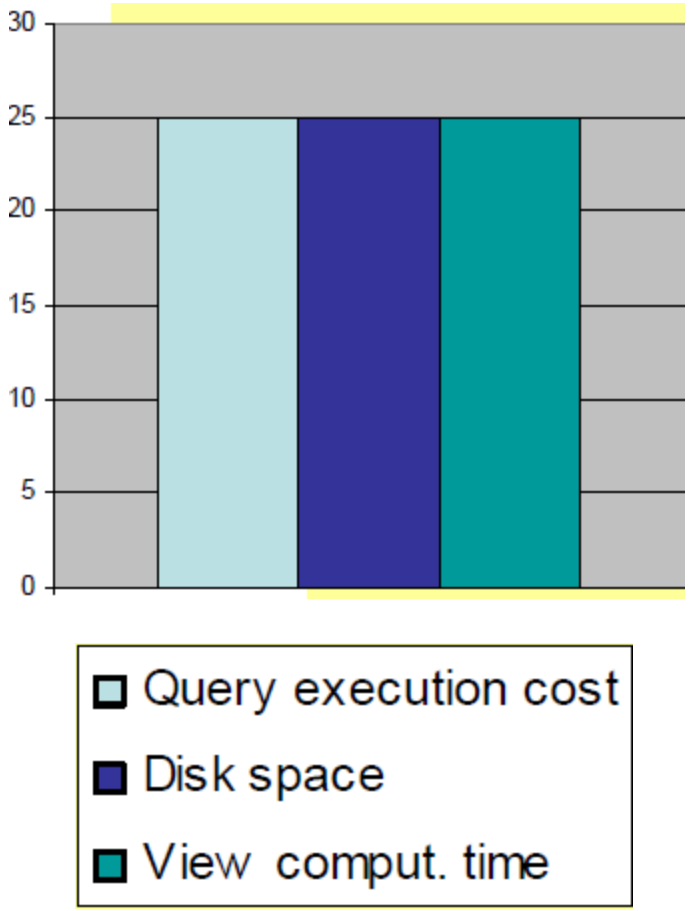

q3
q2
q1


60
50
40
30
20
10
0
Query execution cost
Disk space
View comput. time


q_3
q_2
q_1


60
50
40
30
20
10
0
Query execution cost
Disk space
View comput. time


q3
q2
q1


30
25
20
15
10
5
0
Query execution cost
Disk space
View comput. time

It is useful to materialize a view when:

- It directly solves a frequent query
- It reduce the costs of some queries

It is not useful to materialize a view when:

- Its aggregation pattern is the same as another materialized view
- Its materialization does not reduce the cost

Class Assignment (1)

Suppose that a data warehouse consists of the three dimensions *time, doctor*, and *patient*, and the two measures *count* and *charge*, where *charge* is the fee that a doctor charges a patient for a visit.

(a) Enumerate three classes of schemas that are popularly used for modeling data warehouses.

(b) Draw a schema diagram for the above data warehouse using one of the schema classes listed in (a).

Class Assignment (2)

Suppose that a data warehouse for *XYZ University* consists of the following four dimensions: *student, course, semester*, and *instructor*, and two measures *count* and *avg grade*.

When at the lowest conceptual level (e.g., for a given student, course, semester, and instructor combination), the *avg grade* measure stores the actual course grade of the student. At higher conceptual levels, *avg grade* stores the average grade for the given combination.

(a) Draw a *snowflake schema* diagram for the data warehouse.

Class Assignment (3)

A data warehouse can be modeled by either a *star schema* or a *snowflake schema*. Briefly describe the similarities and the differences of the two models, and then analyze their advantages and disadvantages with regard to one another.

Give your opinion of which might be more empirically useful and state the reasons behind your answer.

Class Assignment (4)

Let us consider the case of a real estate agency whose database is composed by the following tables:

OWNER (IDOwner, Name, Surname, Address, City, Phone)

ESTATE (IDEstate, IDOwner, Category, Area, City, Province, Rooms, Bedrooms, Garage, Meters)

CUSTOMER (IDCust, Name, Surname, Budget, Address, City, Phone)

AGENT (IDAgent, Name, Surname, Office, Address, City, Phone)

AGENDA (IDAgent, Data, Hour, IDEstate, ClientName)

VISIT (IDEstate, IDAgent, IDCust, Date, Duration)

SALE (IDEstate, IDAgent, IDCust, Date, AgreedPrice, Status)

RENT (IDEstate, IDAgent, IDCust, Date, Price, Status, Time)

Design a Star Schema or Snowflake Schema for the DW.

Hints:

The following ideas will be used during the solution of the exercise:

- supervisors should be able to control the sales of the agency

  FACT Sales

  MEASURES OfferPrice, AgreedPrice, Status

  DIMENSIONS EstateID, OwnerID, CustomerID, AgentID, TimeID

- supervisors should be able to control the work of the agents by analyzing the visits to the estates, which the agents are in charge of

  FACT Viewing

  MEASURES Duration

  DIMENSIONS EstateID, CustomerID, AgentID, TimeID

Solution for Class Assignment (4)

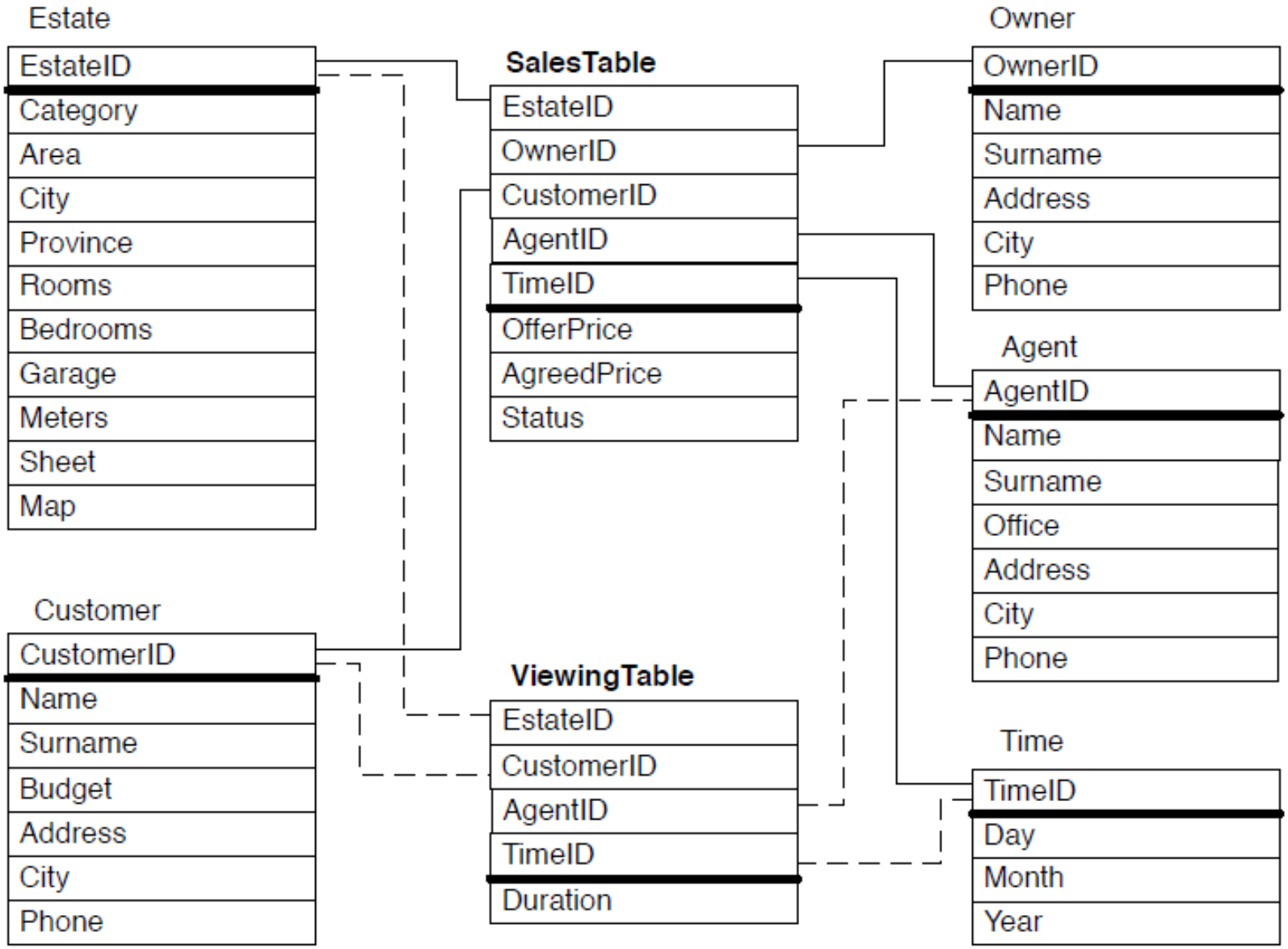


Figure: Star schema

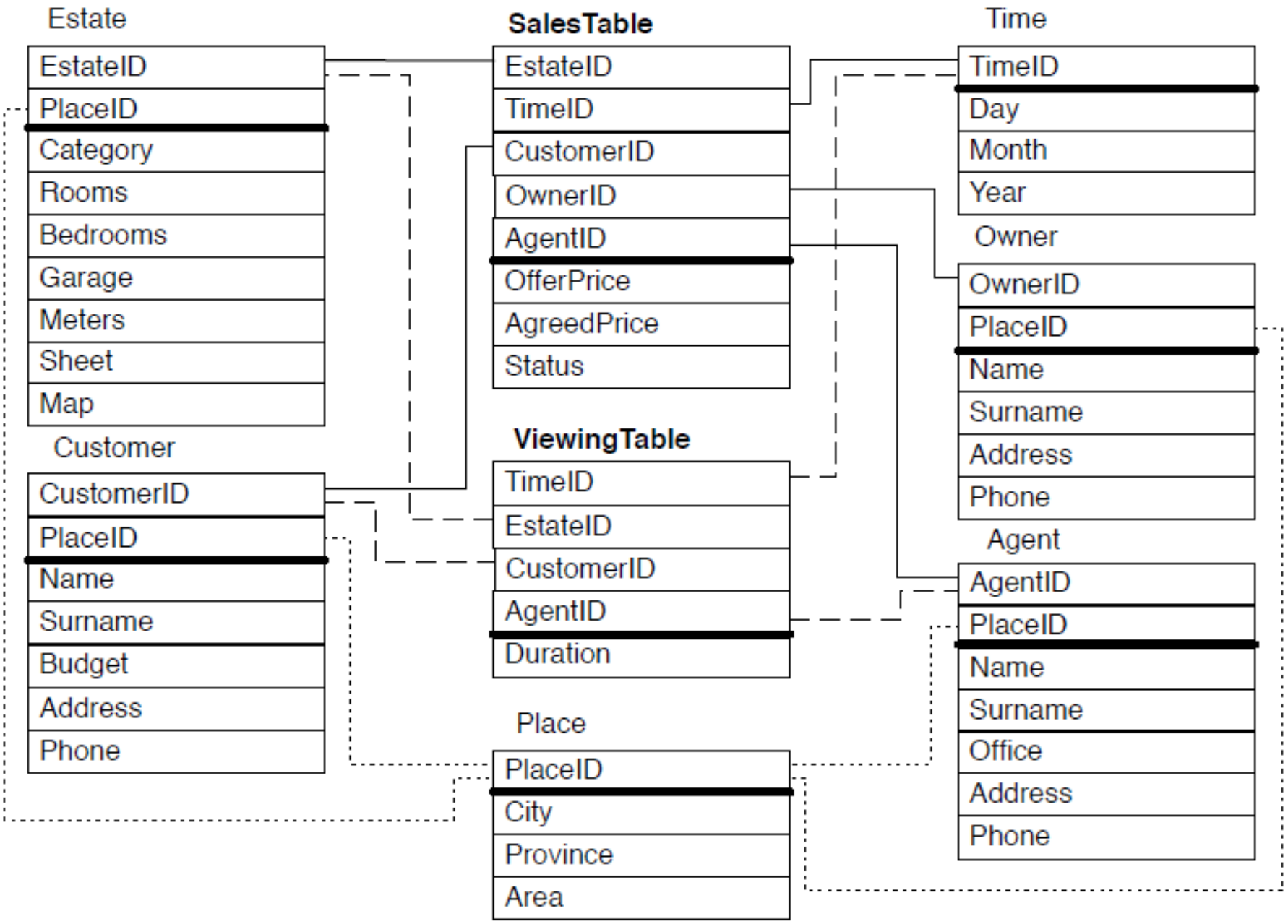


Figure: Snowflake schema

Class Assignment (5)

An online order wine company requires the designing of a data warehouse to record the quantity and sales of its wines to its customers. Part of the original database is composed by the following tables:

CUSTOMER (Code, Name, Address, Phone, BDay, Gender)

WINE (Code, Name, Type, Vintage, BottlePrice, CasePrice, Class)

CLASS (Code, Name, Region)

TIME (TimeStamp, Date, Year)

ORDER (Customer, Wine, Time, nrBottles, nrCases)

Note that the tables represent the main entities of the ER schema, thus it is necessary to derive the significant relationships among them in order to correctly design the data warehouse.

Construct Snowflake Schema.

Solution for Class Assignment (5)

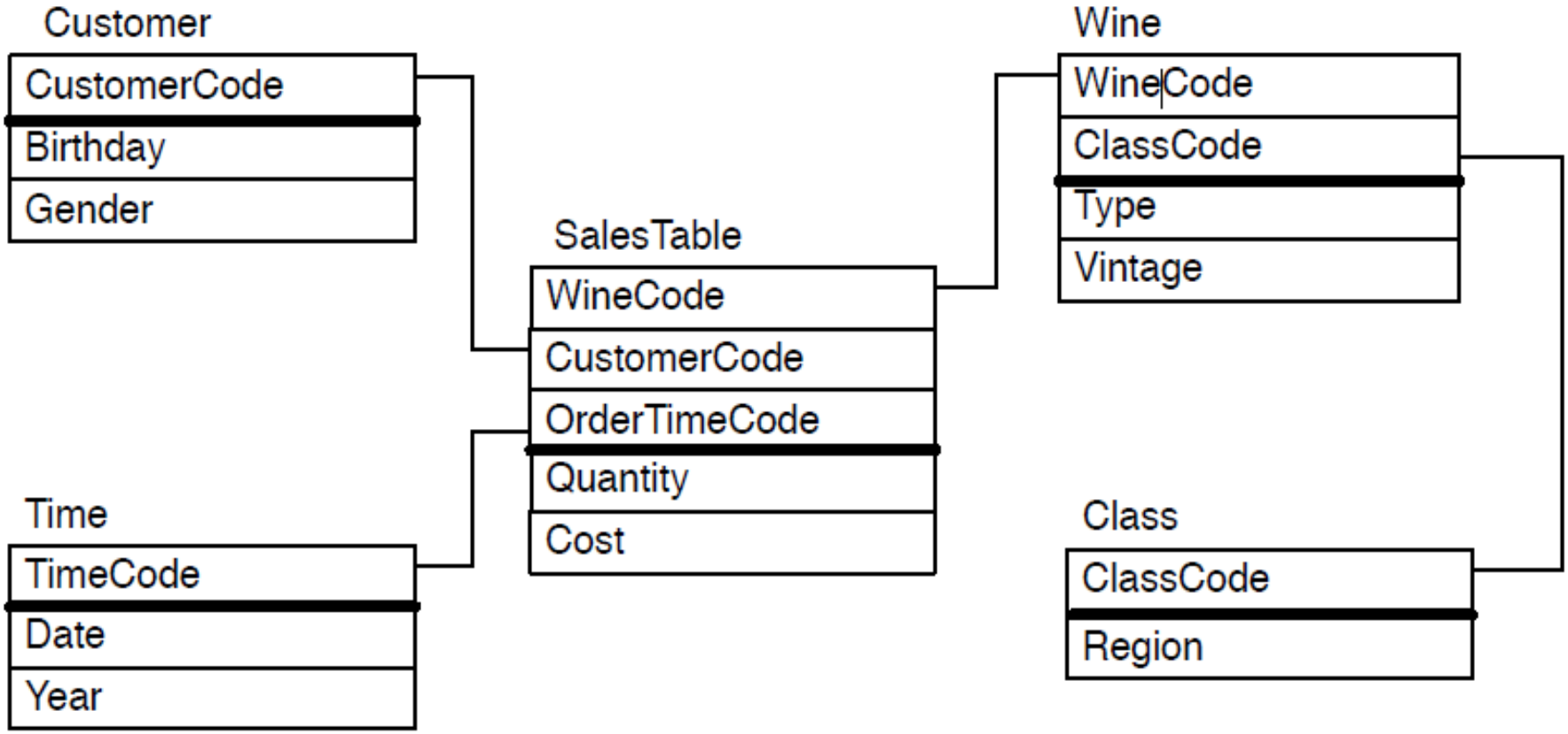
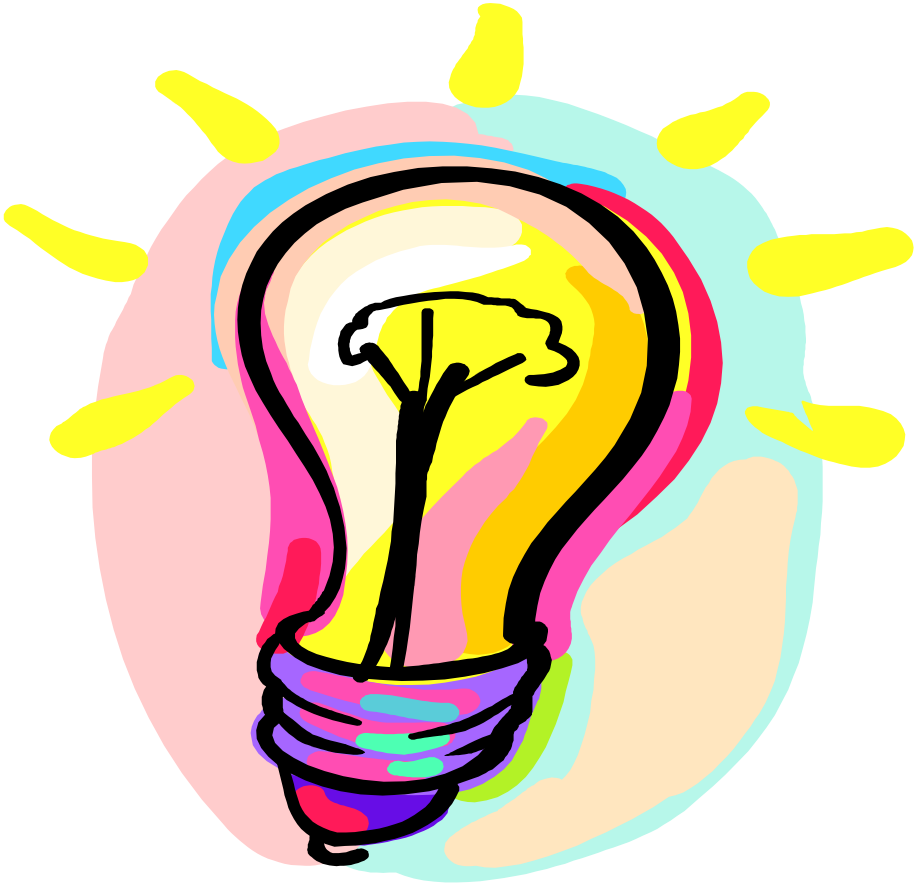


Figure: Snowflake schema

FACT Sales
MEASURES Quantity, Cost
DIMENSIONS Customer, Area, Time, Wine ⟶ Class

Questions?



References

1. Sam Anahory, Dennis Murray, “Data warehousing In the Real World”, Pearson Education.
2. Kimball, R. “The Data Warehouse Toolkit”, Wiley, 1996.
3. Teorey, T. J., “*Database Modeling and Design: The Entity-Relationship Approach”*, Morgan Kaufmann Publishers, Inc., 1990.
4. “An Overview of Data Warehousing and OLAP Technology”, S. Chaudhuri, Microsoft Research
5. “Data Warehousing with Oracle”, M. A. Shahzad

End of Unit 2

Thank you !!!