CHAPTER 11
Advanced Dimension Modeling

- Chapter Objectives
  - Slowly changing dimensions
  - Dealing with large dimensions
  - Snowflake schemas
  - Aggregate fact tables
  - Families of stars

Slowly Changing Dimensions

- Most dimensions are constant over time.
- Some dimensions change slowly.
- Product key of source record does not change.
- Some attribute values change (*description*)
- Source OLTP system, new value overwrites.
- Overwriting is not always the best in dimension tables.
- The type of change determines the method.

Slowly Changing Dimension transformation

The Slowly Changing Dimension transformation supports four types of changes: changing attribute, historical attribute, fixed attribute, and inferred member.

- Changing attribute changes overwrite existing records. This kind of change is equivalent to a Type 1 change. The Slowly Changing Dimension transformation directs these rows to an output named **Changing Attributes Updates Output**.

- Historical attribute changes create new records instead of updating existing ones. The only change that is permitted in an existing record is an update to a column that indicates whether the record is current or expired. This kind of change is equivalent to a Type 2 change. The Slowly Changing Dimension transformation directs these rows to two outputs: **Historical Attribute Inserts Output** and **New Output**.

- Fixed attribute changes indicate the column value must not change. The Slowly Changing Dimension transformation detects changes and can direct the rows with changes to an output named **Fixed Attribute Output**.

- Inferred member indicates that the row is an inferred member record in the dimension table. An inferred member exists when a fact table references a dimension member that is not yet loaded. A minimal inferred-member record is created in anticipation of relevant dimension data, which is provided in a subsequent loading of the dimension data. The Slowly Changing Dimension transformation directs these rows to an output named **Inferred Member Updates**. When data for the inferred member is loaded, you can update the existing record rather than create a new one.

Changing Dimensions (Cont)

- Type 1 Changes – correcting errors
  - Overwrite the old attribute value
- Type 2 changes – preserving history
  - Add new record to dimension table – with new surrogate key
  - Include effective date field
  - Old record does not change
- Type 3 changes – soft revisions
  - Add an attribute for old value
  - Keep same surrogate key

Type 1 Changes – Correction of Errors

- Type 1 Changes – correcting errors
  - Overwrite the old attribute value.
  - Sometimes the change in the source system has no significance.
  - Old value in the source system be discarded.
  - Change in the source system not preserved in data warehouse.
- Type 1 Changes – Data Warehouse
  - Overwrite attribute value in dimension table row.
  - Old value of attribute is not preserved.
  - No other changes made in dimension table row.
  - Key of dimension table are not affected.
  - This type is easiest to implement.

Type 1 Changes

STORE KEY	BEFORE	AFTER
Customer Key:	33154112	33154112
Customer Name:	Kristin Daniels	Kristen Danils
Customer Code:	K12356	K12456
Marital Status:	Single	Single
Address:	733 …	733 …
State:	NY	NY
Zip:	11510	11510

Type 2 Changes – Preserve history

- Type 2 Changes – Preserve history
  - Relate to true changes in source system.
  - Need to preserve history in the data warehouse.
  - This type partitions the history in the data warehouse.
  - Every change for the same attribute must be preserved.
- Type 2 Changes – Data Warehouse
  - Add new dimension table row with the new value of the changed attribute
  - Effective date field included in the dimension table
  - No changes to original row in the dimension table
  - Key of dimension table are not affected
  - New row inserted with a new surrogate key

Type 2 Changes

	BEFORE	AFTER	AFTER
Customer Key:	33154112	51141234	52789342
Customer Name:	Kristin Daniels	Kristin Daniels	Kristin Botha
Customer Code:	K12356	*K12456*	*K12456*
Marital Status:	Single	Single	MARRIED
Address:	733 …	733 …	147 Ninth S
State:	NY	NY	LA
Zip:	11510	11510	11510
Effective Date	-	01/10/2002	01/11/2002

Type 3 Changes – Tentative Soft Changes

- Type 3 Changes – Tentative Soft Changes
  - What if you need to count the orders after a cu-off date in both time groups? Type 2 will not work for this
  - Need to keep track of history with old and new values of the changed attribute
  - Provides ability to track forward and backward
- Type 3 Changes – Data Warehouse
  - Add an “old column” in the dimension table
  - Push value of “current” field to “old” field
  - Keep the new value always in the “current” field
  - Add “current” effective date field for the attribute
  - The key of the row is not effected
  - No new dimension row is needed

Type 3 Changes

	BEFORE	AFTER
Customer Key:	33154112	33154112
Customer Name:	Kristin Daniels	Kristin Daniels
Customer Code:	K12356	K12456
Marital Status:	Single	Single
Address:	733 …	714 Ninth S
	Boston	New York
Old Address:		733 South Street
		Boston
State:	USA	USA
Zip:	11510	11510
Effective Date	01/01/2001	01/11/2002

Slowly Changing Dimensions

- Type 1: Overwrite the dimension record
- Type 2: Create new dimension record
- Type 3: Create an ‘old’ field in the dimension record
- Type 4: Add a *valid_from* and *valid_until* field in the dimension record

Ad. Type 2: *requires* surrogate keys, but in general, one should *always* use these because of performance and flexibility

Ad. Type 4: Kimball only recognizes 3 types SCD’s

Always Use Surrogate Keys

- Allows DWH to assign new key versions for SCD's (type 2)
- Higher performance with numeric keys than with long, alphanumeric keys

Large Dimensions

- May be very wide and very deep
- Use type 2 changes as far as possible
- If dimension table gets too large:
  - Break dimension table into two tables.
  - Place all the fast changing attributes in one table
- Tend to have multiple hierarchies
- Issues to address for very large dimensions
  - Effective design methods
  - Proper indexing strategies
  - Optimisation techniques (caching, partitioning)

Rapidly Changing Dimensions

- With Type 2, you create an additional dimension table row the the new value – preserve history
- This can lead to many additional rows in a table
- What if dimension table is too large and is changing too rapidly?
  - Slow response time for queries
  - Need to break large dimension table into more tables
  - Place all the fast changing attributes in one table
  - I.e split the customer dimension table into 2 dimension tables
  - Figure 11-6

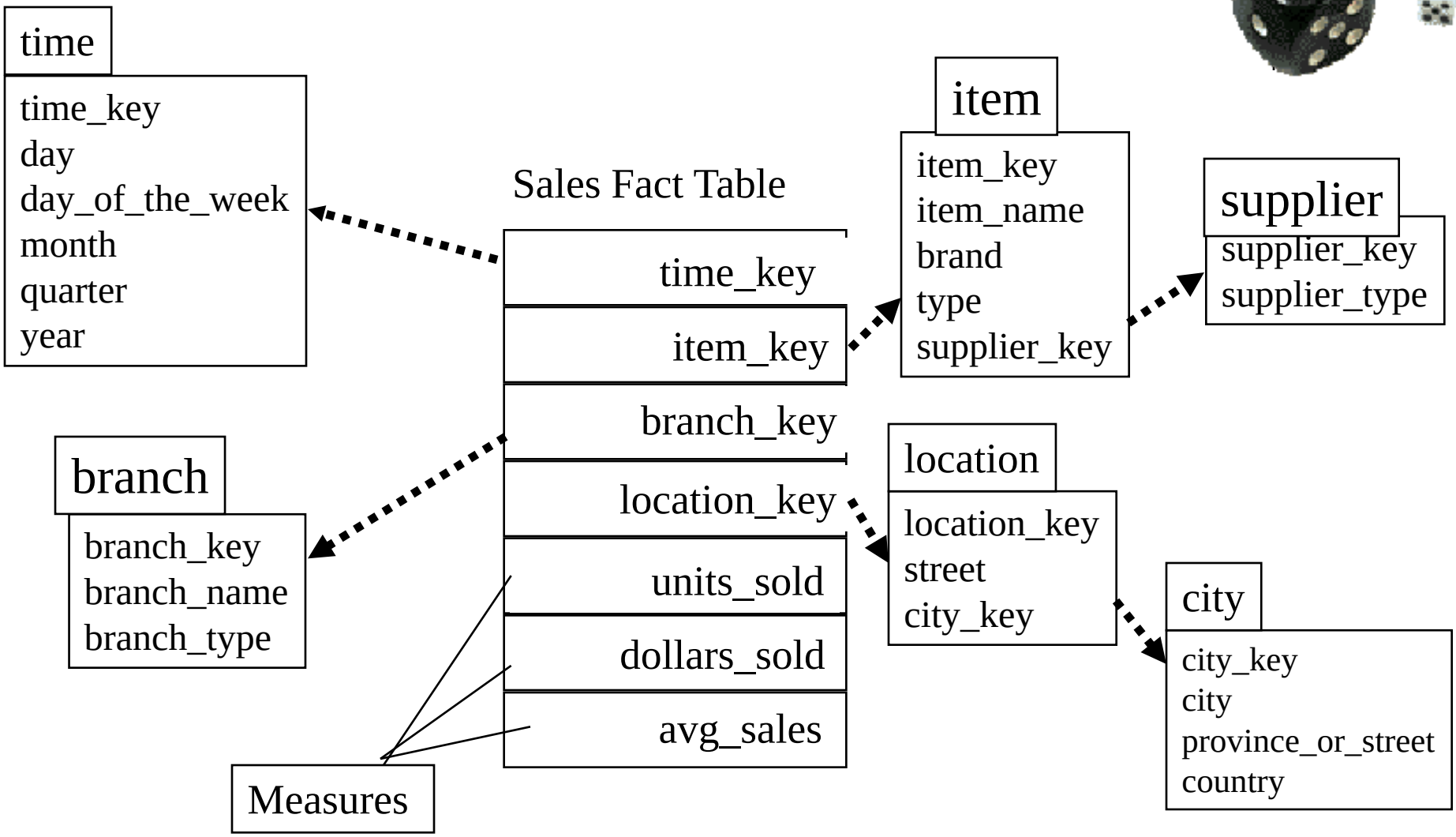
Junk Dimensions

- Yes/no flags, textual codes and free form texts.
- Options:
  - Discard all flags and text – could be throwing away valuable information
  - Place flags and text unchanged in fact table – could lead to a very large, inefficient fact table
  - Make each a separate dimension table – could lead to large number of dimension tables in the STAR schema
  - Keep only those flags and text that are meaningfull; group all useful ones in one junk dimension

Snowflake Schema

- A ***Snowflake Schema*** is a Star schema with fully normalized dimensions—it gets its name because it forms a shape similar to a snowflake.
- The advantage of the snowflake structure is that it explicitly shows the structure of each ***Dimension*** rather than appearing as an unstructured collection of data items.
- Multiple 'parent' ***Dimension***s may be defined in the snowflake schema
  - A 'parent' ***Dimension*** is one the “1 side” of a 1:M relationship with another '***Dimension***

Example of Snowflake Schema



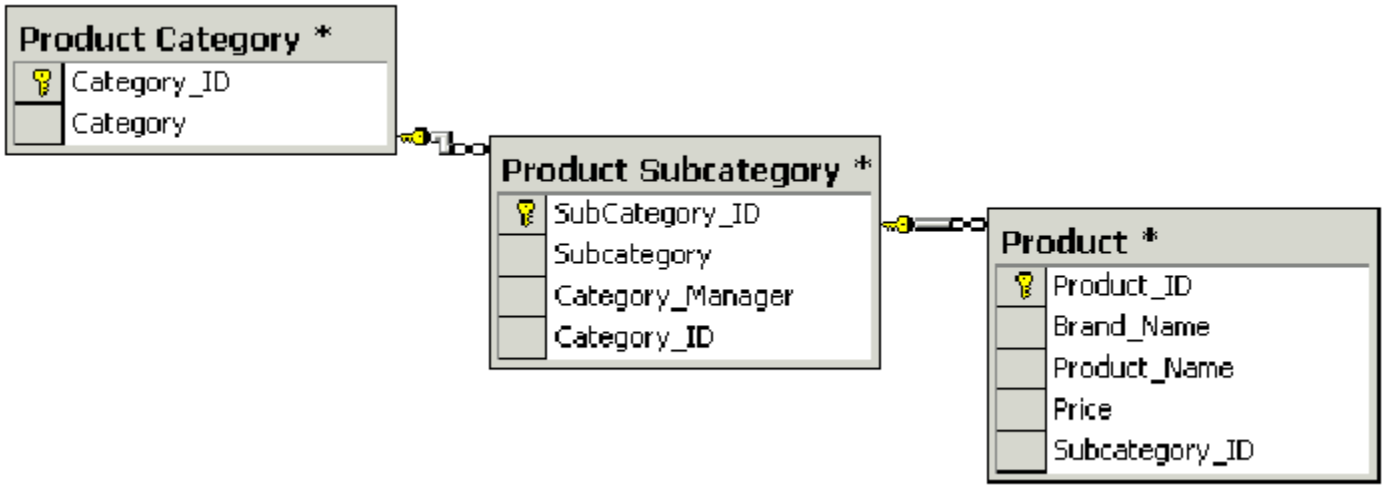
Snowflake Schemas

- Dimensional model uses unnormalized dimensions.
- If dimensions are partially or fully normalized – snowflaking.
  - Partially normalize only a few dimension tables
  - Partially/fully normalize only a few dimension tables
  - Partially normalize every dimension table
  - Fully normalize every dimension table
- Low cardinality attributes are put into new tables.
- When to snowflake – demographic attributes with different granularity.
  - You have some attributes that belong to another “grain”

Snowflake Schemas (Cont…)

- Advantages
  - Small savings in storage space
  - Normalized structures are easier to update and maintain
- Disadvantages
  - Schema less intuitive and end-users are put off
  - More difficult to browse contents
  - Degraded query performance, hampering performance
  - Snowflaking is not generally recommended for a Data Warehouse

Snowflake Example



- Defines hierarchs by using **multiple dimension tables**
- More **normalized** than a single dimension table
- **Supported by MS Analysis Services**

Aggregate Fact Tables

- Aggregates: pre-calculated summaries of most granular data at higher levels along dimension hierarchies
- Fact table size
  - Time=1825 rows, Store=300 rows, Fact=547500 rows
- Need for aggregates
  - Fewer rows than base tables
  - Better performance
  - Aggregating fact tables (one-way ,two-way, etc)

Aggregate Fact Tables (Cont…)

- One-Way Aggregates (one dimension)
  - Product category by store, by date
- Two-Way Aggregates (two dimensions)
  - Product category by territory, by date
- Three-Way Aggregates (three dimensions)
  - Product category by territory, by month
- Figure 11-14
- Effect of sparsity on Aggregation
- Figure 11-15

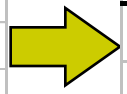
Aggregate Fact Tables (Cont…)

- Aggregation Strategy
  - Do not get bogged down with too many aggregates.
  - Try to cater to a wide range of user groups.
  - Go for aggregates that do no increase the overall usage of storage.
  - Keep the aggregates hidden from the end-user.
  - Keep the impact on the data staging process as less intensive as possible.

Aggregates (Example)

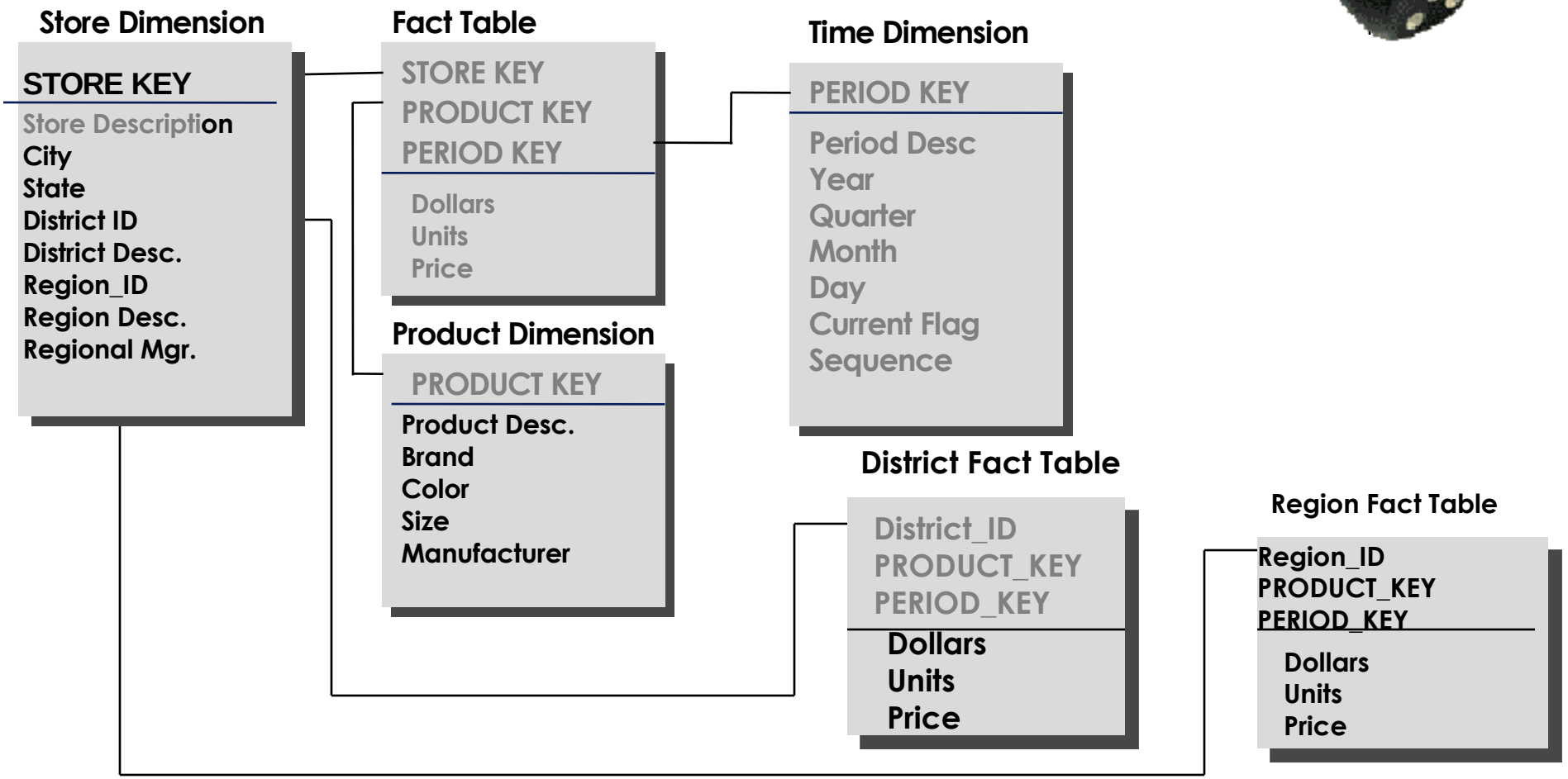
- Add up amounts by day, product
- In SQL: SELECT date, sum(amt)
  FROM SALE
  GROUP BY date, prodId
- Operators: sum, count, max, min, median, avg
- "Having" clause

sale	prodId	storeId	date	amt
	p1	s1	1	12
	p2	s1	1	11
	p1	s3	1	50
	p2	s2	1	8
	p1	s1	2	44
	p1	s2	2	4



sale	prodId	date	amt
	p1	1	62
	p2	1	19
	p1	2	48

Aggregates (Example)



Families of Stars

- Reasons for existence of families:
  - Specific purpose of each star
  - Aggregate fact tables
  - Snapshot and transaction tables
    - In-time snapshot (bank balance)
    - Transaction (your savings account statement)
  - Core and custom tables
- Conforming dimensions
  - Sharing dimensions among fact tables
- Standardizing facts
  - Resolve synonyms, guarantee same algorithm used for any derived unit in each fact table
  - Make sure each fact uses the right unit of measurement

Example of Fact Constellation

time

time_key
day
day_of_the_week
month
quarter
year

item

item_key
item_name
brand
type
supplier_type

Shipping Fact Table

time_key
item_key
shipper_key
from_location
to_location
dollars_cost
units_shipped

Sales Fact Table

time_key
item_key
branch_key
location_key
units_sold
dollars_sold
avg_sales

branch

branch_key
branch_name
branch_type

location

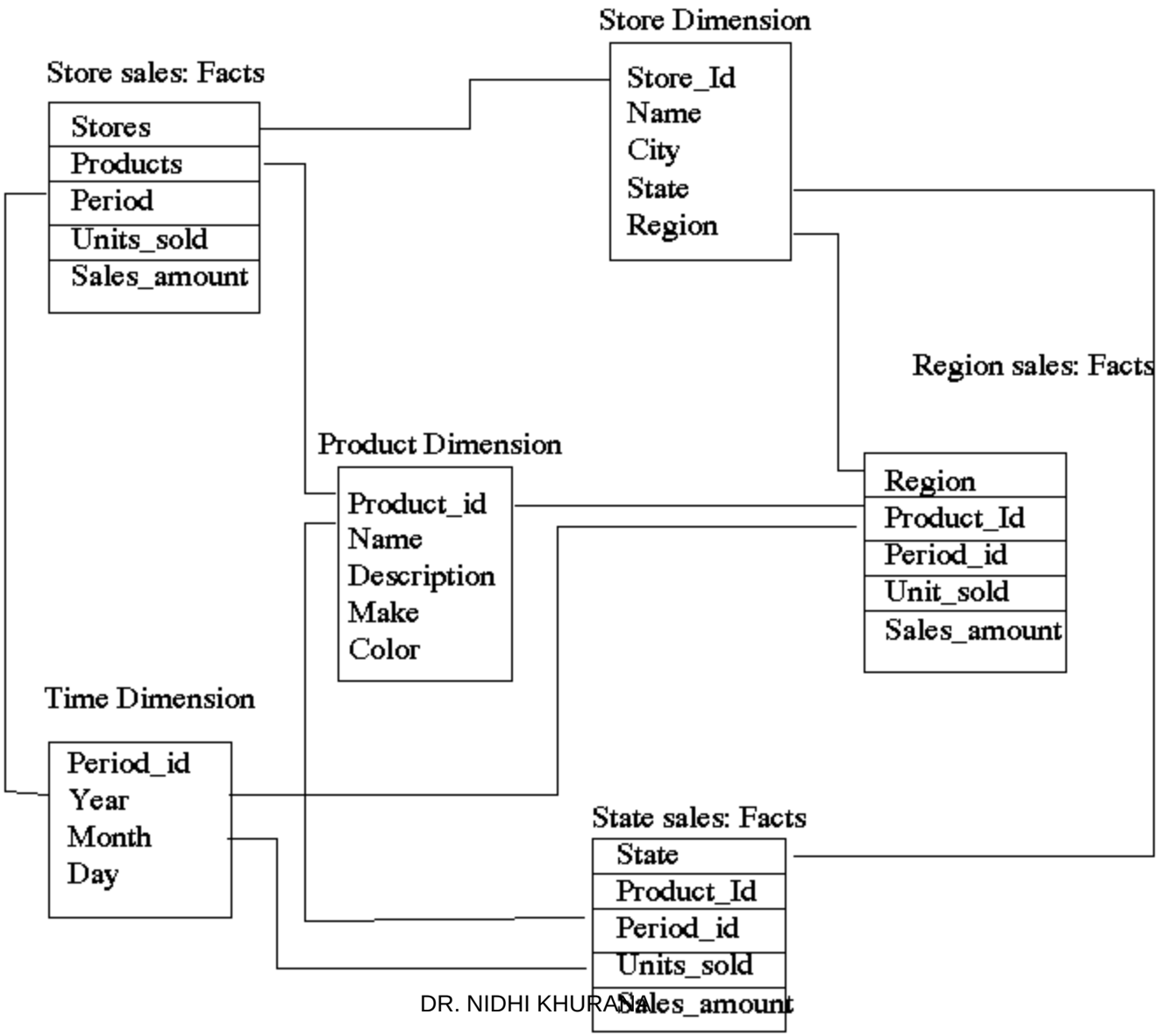
location_key
street
city
province_or_street
country

shipper

shipper_key
shipper_name
location_key
shipper_type

Measures

Example of the Fact Constellation Schema



ROLAP: Dimensional Modeling Using Relational DBMS

- Special schema design: *star, snowflake*
- Special indexes: bitmap, multi-table join
- Special tuning: maximize query throughput
- Proven technology (relational model, DBMS), tend to outperform specialized MDDB especially on large data sets
- Products
  - IBM DB2, Oracle, Sybase IQ, RedBrick, Informix

MOLAP: Dimensional Modeling Using the Multi Dimensional Model

- MDDB: a special-purpose data model
  - Multidimensional database
- Facts stored in multi-dimensional arrays
- Dimensions used to index array
- Sometimes on top of relational DB
- Products
  - Hyperion Essbase, Microsoft Analysis Server, Cognos PowerPlay, Business Objects, Oracle OLAP