



Logical Design in Relational OLAP



COMP323 Chapter 3

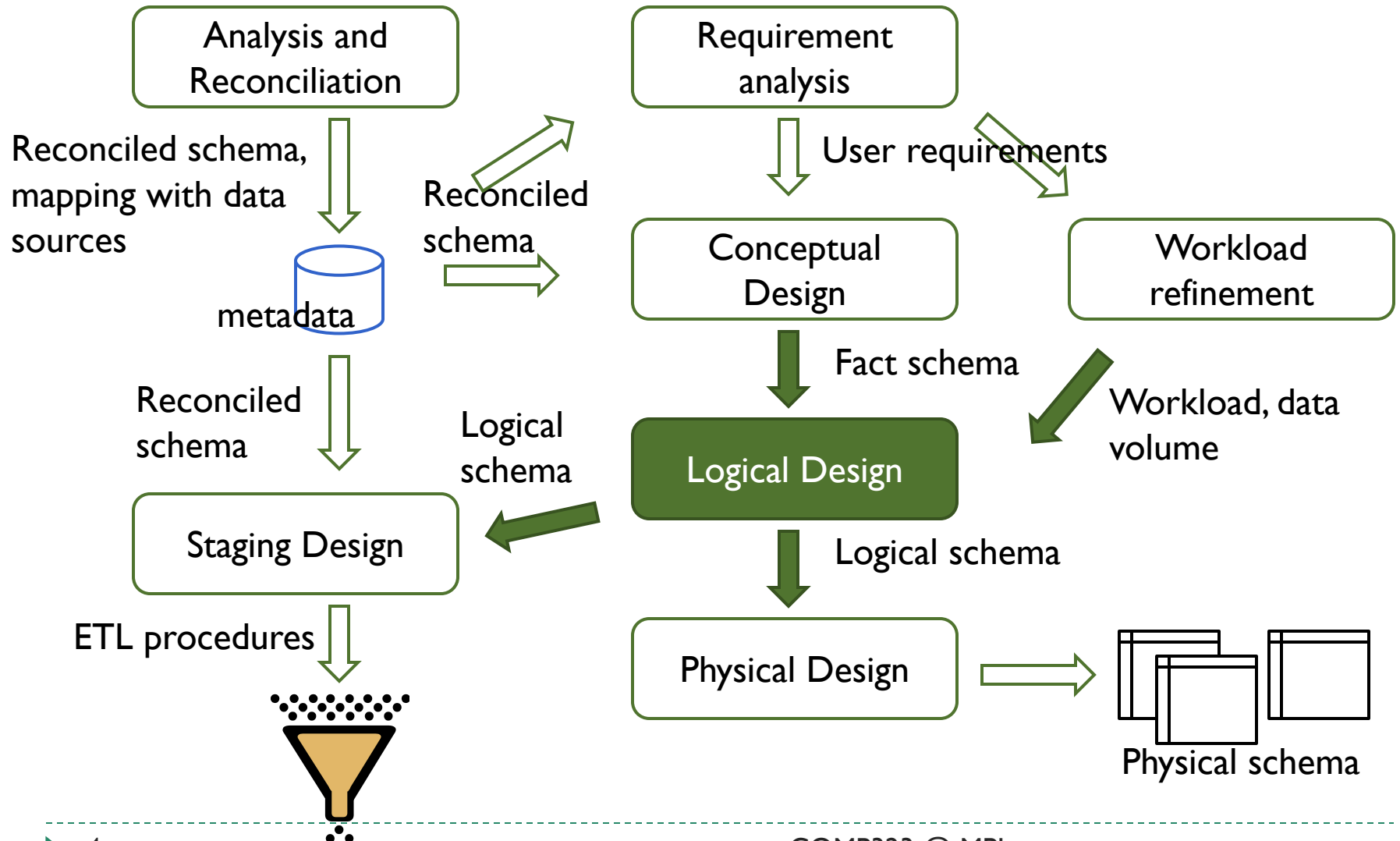
Outline

- ▶ **A. Star schema**
 - ▶ Fact and dimension tables
 - ▶ From fact schema to star schema
 - ▶ Writing analysis queries in SQL
- ▶ **B. Advanced features**
 - ▶ Snowflake schema
 - ▶ Shared hierarchies
 - ▶ Degenerate dimensions
 - ▶ Slowly changing dimensions (SCD)
- ▶ **C. Aggregate fact table**
 - ▶ Aggregate fact table
 - ▶ Relational schema for aggregate fact tables
 - ▶ Query rewrite

Part A. Star Schema

- ▶ Logical design:
 - ▶ Relational OLAP (this chap)
 - ▶ MOLAP and HOLAP (brief intro)
- ▶ Star schema basics:
 - ▶ Fact table, compound key
 - ▶ Dimension tables, surrogate key, why?
 - ▶ Shape and content of fact and dimension tables
 - ▶ Logical design: from fact schema to star schema
- ▶ Analysis query in SQL
 - ▶ Using JOIN, GROUP BY, and WHERE
 - ▶ OLAP extension in SQL-99

Design Methodology

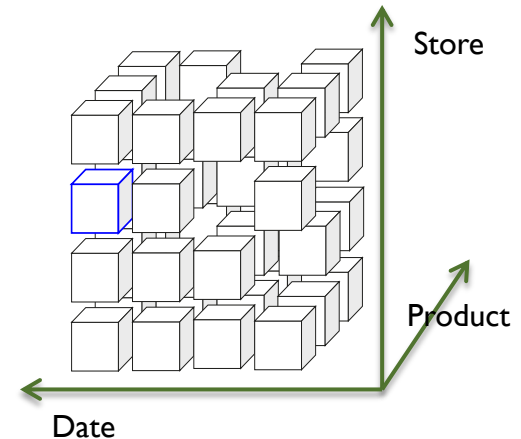


Logical Design

- ▶ In logical design, a data mart designer has to select a logical model to implement the conceptual schema.
- ▶ Three common approaches
 - ▶ Relational OLAP (ROLAP) uses relational model in common DBMS to represent the multidimensional data
 - ▶ Multidimensional OLAP (MOLAP) implements the multidimensional data structure natively in a database
 - ▶ Hybrid OLAP (HOLAP) hybrid model stores a part of the data in multidimensional data structure

Multidimensional OLAP

- ▶ MOLAP uses multidimensional array to store data
- ▶ Benefits:
 - ▶ good performance. No need to simulate via complex SQL queries
- ▶ Problems:
 - ▶ Waste of space to store empty cells. Data sparsity indicates that just a small part of the cells in a multidimensional cube actually include information.
 - ▶ Lack of standards of logical models in MOLAP
- ▶ Example: icCube
- ▶ We'll discuss MOLAP in next chapter...



Relational OLAP

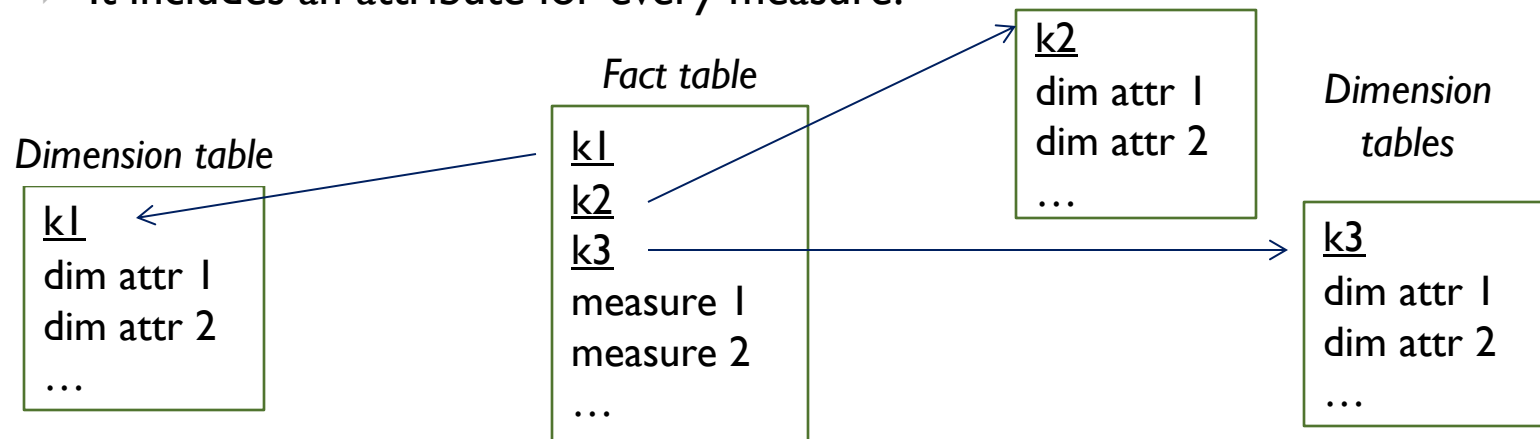
- ▶ ROLAP adopts the well-known relational model to represent multidimensional data.
- ▶ Benefits:
 - ▶ Relational systems are industry standard. Widely available skills
 - ▶ Highly sophisticated and optimized tools (30yr vs. 15yr of MOLAP)
 - ▶ No sparsity in relational systems. More scalable.
- ▶ Examples:
 - ▶ Pentaho, Oracle

Hybrid OLAP

- ▶ HOLAP stores part of the data in multidimensional data structure, and others in relational database
 - ▶ Vertical partitioning: stores aggregates in MOLAP for faster query performance, and detail data in ROLAP
 - ▶ Horizontal partitioning: stores some slices of data in MOLAP, e.g. the more recent data (as filtered by the Date dimension), and other data in ROLAP
- ▶ Benefits:
 - ▶ Combine the best of both ROLAP and MOLAP
- ▶ Examples:
 - ▶ Microsoft SQL server Analysis Services

Star Schema

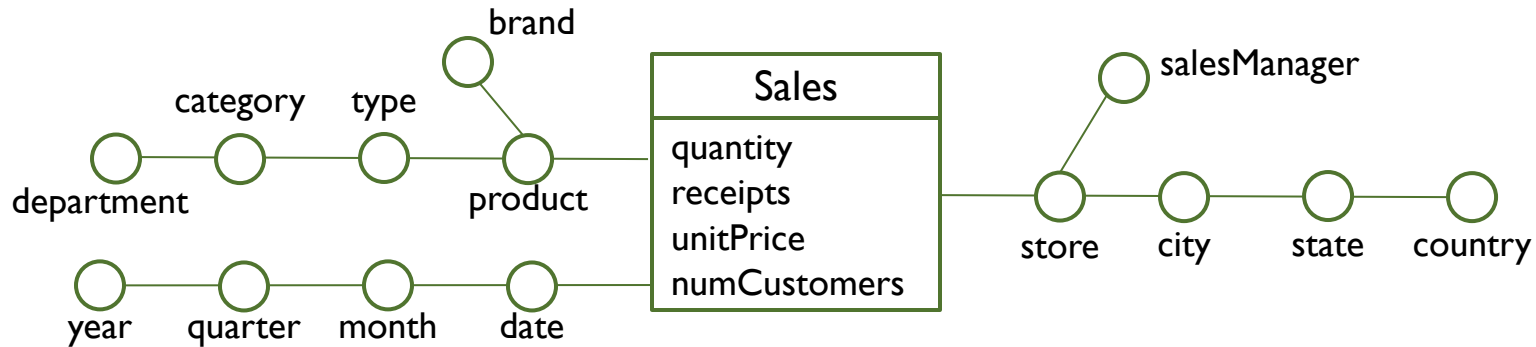
- ▶ ROLAP uses a star schema to represent multidimensional data.
A **star schema** consists of the following:
 - ▶ One **dimension table** for each dimension
 - ▶ Each dimension table features a primary key (k_i) and a set of attributes describing its dimension instance at different aggregation levels
 - ▶ One **fact table** referencing all the dimension tables
 - ▶ Its primary key is the composition of the set of foreign keys (k_1 through k_n) referencing the dimension tables.
 - ▶ It includes an attribute for every measure.



Logical Design

- ▶ We obtained a fact schema in conceptual design. In logical design, we need to create a star schema from the fact schema.
- ▶ To create a star schema from the fact schema,
 - ▶ Create a dimension table for each dimension
 - ▶ Add a surrogate key to each dimension table
 - ▶ Add an attribute for each dimensional attributes (i.e. levels) in the dimension
 - ▶ Create a fact table for the fact
 - ▶ The composite key of the fact table consists of foreign keys to each dimension tables
 - ▶ Add an attribute for each stored measure

Example: Star Schema



/* Fact table */

SALES (keyS: STORE, keyD: DATE, keyP: PRODUCT,
quantity, receipts, unitPrice, numCustomers)

/* Dimension tables */

STORE (keyS, store, city, state, country, salesManager)

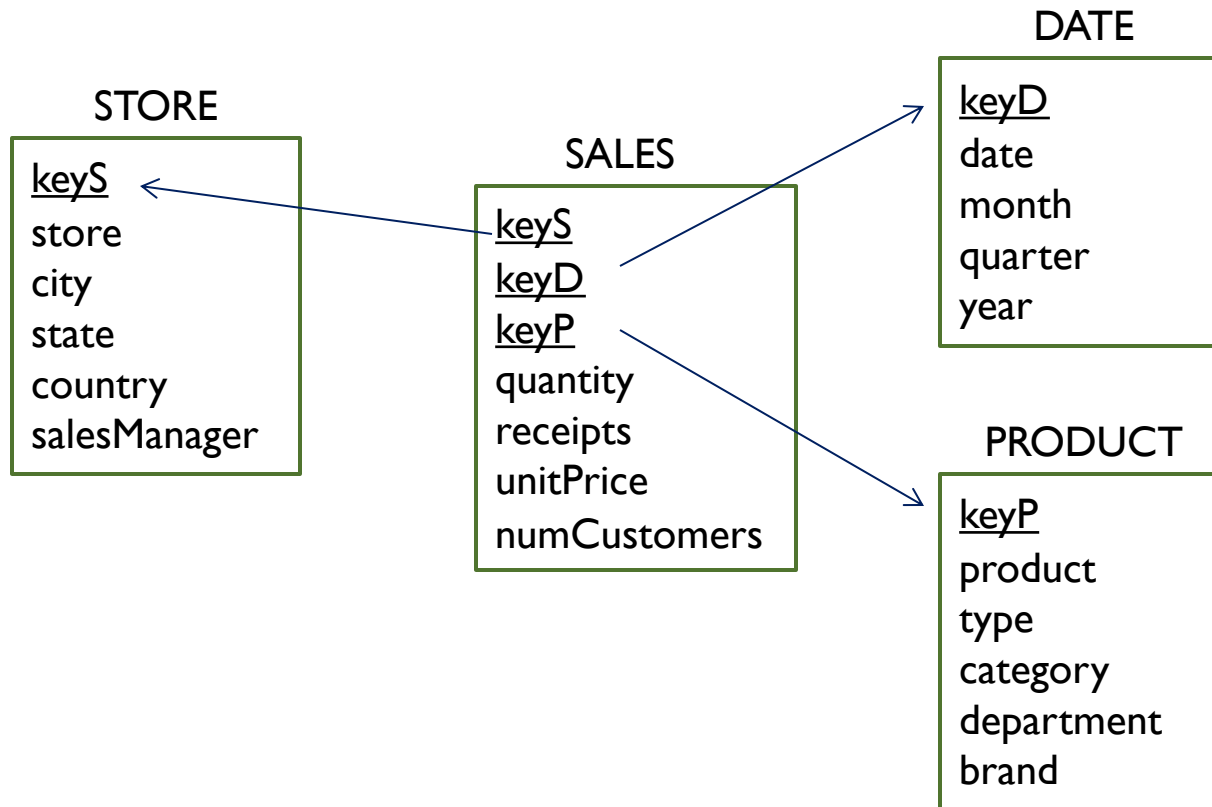
PRODUCT (keyP, product, type, category, department, brand)

DATE (keyD, date, month, quarter, year)

Measures

Dimensional attributes /
levels

Example: Star Schema



Sample data in fact and dimension tables

<u>keyS</u>	<u>keyD</u>	<u>keyP</u>	quantity	receipts	...
1	1	1	170	85	...
2	1	2	320	160	...
3	2	3	412	412	...
...

Fact table: SALES

STORE

<u>keyS</u>	store	city	state	...
1	evermore	Columbus	Ohio	...
2	Profit	Austin	Texas	...
3	evermore2	Austin	Texas	...
...

PRODUCT

<u>keyP</u>	product	type	category	brand	...
1	Milk	Dairy	Food	Healthy	...
2	Yogurt	Dairy	Food	Healthy	...
3	Cheese	Dairy	Food	Tasty	...
...

DATE

<u>keyD</u>	date	month	quarter	year
1	1/1/2011	1/2011	Q1 2011	2011
2	2/1/2011	1/2011	Q1 2011	2011
3	3/1/2011	1/2011	Q1 2011	2011
...

Dimension tables

Surrogate Key in Dimension Tables

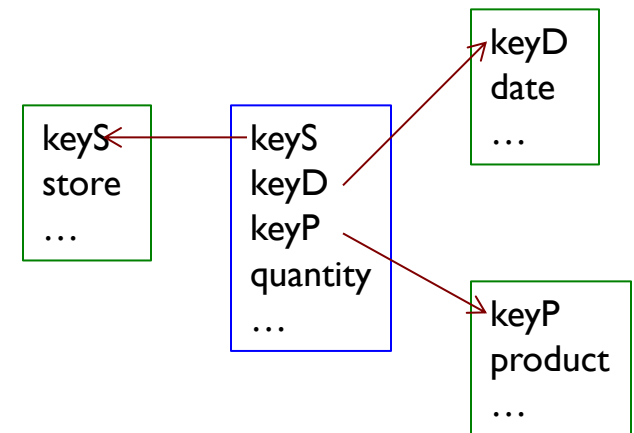
- ▶ A **dimension table** uses a system-generated sequence number as the primary key. This is known as **surrogate key**.
- ▶ The primary key (natural key) from operational database is not suitable for data warehouse
 - ▶ The key may have built-in meaning. E.g. a product code begins with two letters that denotes the warehouse where the product is usually stored. Such key changes if the product is now stored in a different warehouse.
 - ▶ The key may be recycled. E.g. the customer number of a discontinued customer may be assigned to a new customer

Primary Key of Fact Table

- ▶ The **primary key** of the fact table is the concatenation of the **foreign keys** to the dimension tables
 - ▶ Dimensions are the coordinates to locate a cell (primary event). We can locate a primary event by selecting a row in each dimension table.
 - ▶ E.g. PK of Sales is (keyS, keyD, keyP)

```
/* Fact table */  
SALES (keyS: STORE, keyD: DATE, keyP: PRODUCT,  
       quantity, receipts, ...)
```

```
/* Dimension tables */  
STORE (keyS, store, city, ... )  
PRODUCT (keyP, product, type, ...)  
DATE (keyD, date, month, ...)
```



Foreign keys in a star schema

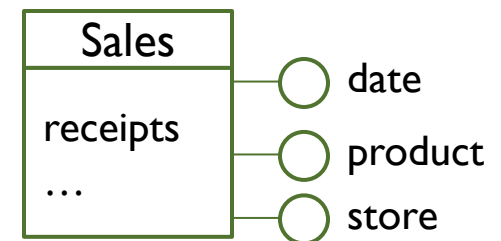
JOINing the fact and dimension tables

- ▶ A multidimensional view of data is obtained when you join the fact table to its dimension tables

```
SELECT D.date, P.product, S.store, F.receipts
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
  JOIN PRODUCT P    ON F.keyP = P.keyP
```

Date	Product	Store	Receipts
1/1/2011	milk	evermore	85
1/1/2011	yogurt	profit	160
2/1/2011	cheese	evermore2	412
...

```
SELECT D.date, P.product, S.store, F.receipts
FROM SALES F, STORE S, DATE D, PRODUCT P
WHERE F.keyS = S.keyS
AND F.keyD = D.keyD
AND F.keyP = P.keyP
```



Same query without using the 'JOIN' syntax

Pros and Cons of Surrogate Keys

► Advantages

- Require less space in fact tables
- Provide quicker access to data because query execution can use simple indexes based on a single numeric attribute
- Offer independence of any changes of identifier values applied to operational sources
- Are able to represent many versions of an individual instance in slowly changing dimension (SCD).

► Disadvantages

- Cause an increase in size of dimension tables if natural keys are also included in dimension tables
- Force you to create additional UNIQUE index based on a natural key to check for dimension tables to be free from duplicates

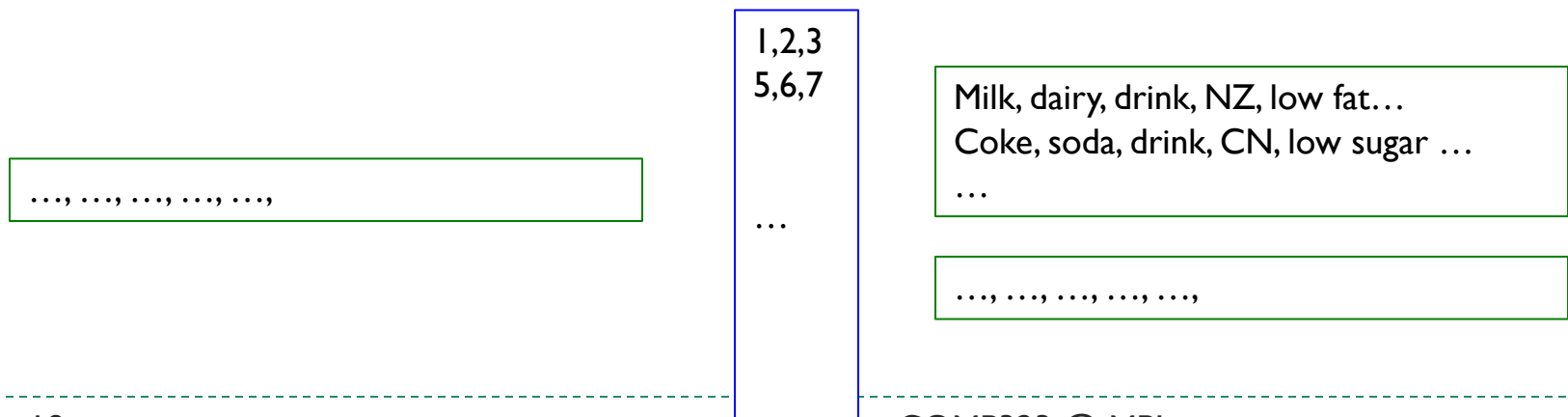
Content in Fact and Dimension Tables, 1

► Fact table

- **Deep, but not wide**: it contains few attributes, but a large number of records.
- Most attributes are **numerical** measures that can be aggregated

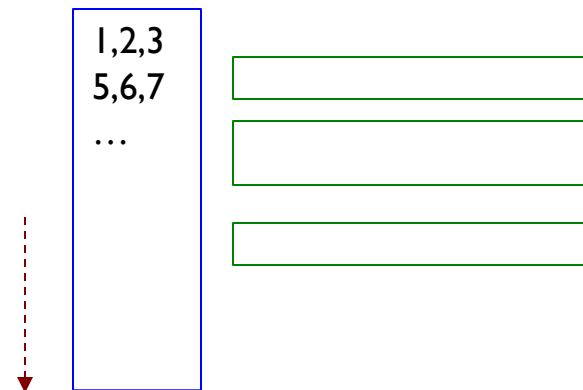
► Dimension table

- **Wide, but not deep**: it contains many attributes, but fewer records than the fact table
- Most attributes are **textual** attributes that are used to filter or sort data in fact table



Content in Fact and Dimension Tables, 2

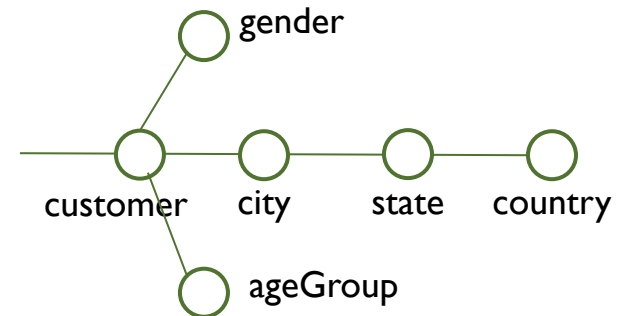
- ▶ New records are appended to the fact table often
 - ▶ Fact table grows fast
 - ▶ Existing records rarely change
- ▶ Records in dimension tables are seldom changed or inserted
 - ▶ Changes are known as slowly changing dimension
- ▶ Therefore, fact tables occupy most storage in a data mart
- ▶ Sparsity is not an issue because fact tables just store records for events that have taken place.



Observation on Dimension Tables

► Dimension tables

- are **denormalized**: some attribute value appear repeatedly. Some attributes are not directly related.
- usually contains attributes related by many-to-one relationship. These define a hierarchy which supports drill-down and rollup operators
- may contain more than one hierarchies
- We can perform cross fact analysis if the fact tables shared conformed dimensions



Restriction and Aggregation in SQL

- ▶ **WHERE** selects events. Fix value for dimensional attributes.
- ▶ **GROUP** defines aggregation with a set of dimensional attributes.
- ▶ **SUM** is the aggregation operator
- ▶ Example: What's the **total** sales receipt **per city** **per product type** in **Jan 2012**?

		Product type	
		Dairy	Drink
City	Macau	100	150
	HK	300	400

```
SELECT S.city, P.type, SUM(F.receipts)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
  JOIN PRODUCT P    ON F.keyP = P.keyP
WHERE D.month = 'Jan 2012'
GROUP BY S.city, P.type;
```

City	Type	Rcpt
Macau	Dairy	100
Macau	Drink	150
HK	Dairy	300
HK	Drink	400

Query examples

- ▶ Q1. What is the total sales receipts in each year?
- ▶ Q2. What is the monthly sales receipts in 2012?
- ▶ Q3. What is the sales receipts by store city and product brand in 2012?
- ▶ Q4. Track the changes in average price of soft drink in 2012. (What is the average price of products in the type 'soft drink' for each month?)
- ▶ Q5. What are the best selling soft drink in Macau?

```
/* Fact table */  
SALES (keyS: STORE, keyD: DATE, keyP: PRODUCT,  
       quantity, receipts, unitPrice, numCustomers)  
  
/* Dimension tables */  
STORE (keyS, store, city, state, country, salesManager)  
PRODUCT (keyP, product, type, category, department, brand)  
DATE (keyD, date, month, quarter, year)
```

Review questions

- ▶ A dimension table is wide; the fact table is deep. Explain.
- ▶ Describe the composition of the primary keys for the dimension and fact tables.
- ▶ Design a star schema for the conceptual schema at the case study at the end of last chapter.
- ▶ What is an empty fact schema? Design a star schema for the empty fact schema of 'class attendance' in last chapter.

Part B. Advanced topics in Dimension Tables

- ▶ **Snowflake schema**
 - ▶ More table join, more surrogate keys
 - ▶ Snowflake vs. star schema
 - ▶ Analysis query in SQL
- ▶ Implement shared hierarchy
- ▶ Other advanced features

Normalization in operational and analysis databases

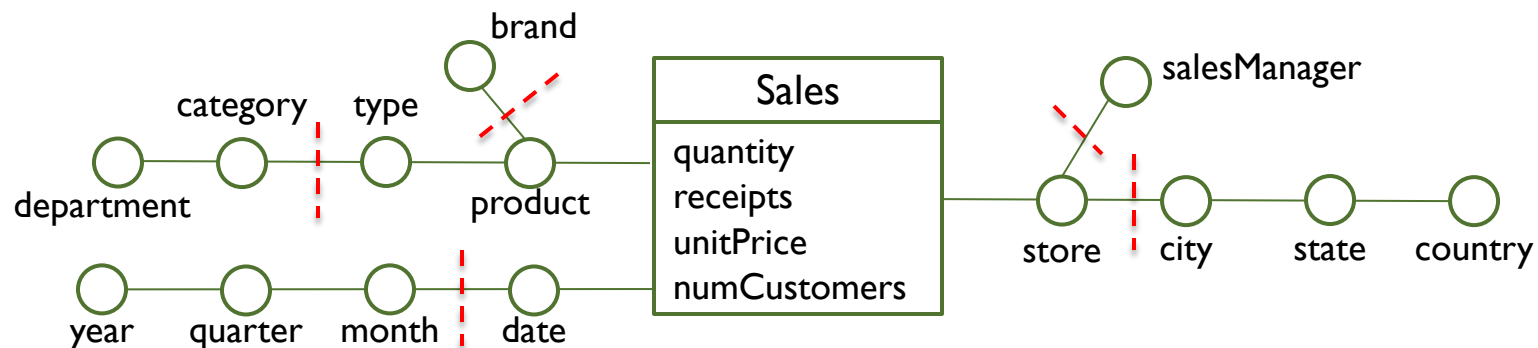
- ▶ Normalization in operational database is important to ..
 - ▶ Reduce redundancy and save storage
 - ▶ Maintain integrity in case of updates
- ▶ But normalization of dimension tables is not as important, because
 - ▶ Fact tables are usually much larger than dimension tables. So normalization does not save much space in data marts.
 - ▶ Insertion into dimension tables are handled by ETL process, which can maintain consistency of redundant data easily
 - ▶ Updates to dimension tables are not often. (More discussion in SCD topic)

Benefits of Denormalization in Star Schema

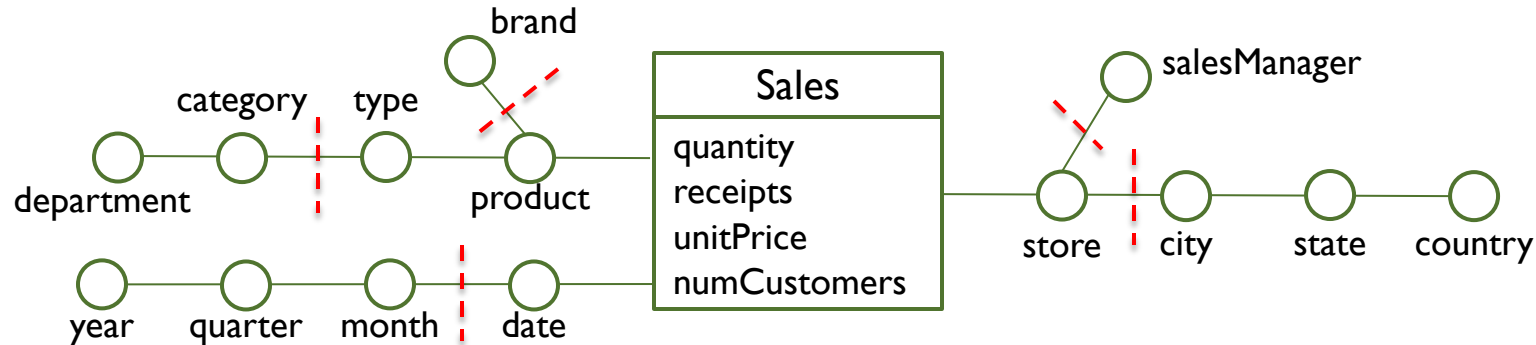
- ▶ In star schema, all dimensional attributes of a dimension are stored in a relational table
 - ▶ Dimension tables are **denormalized** (i.e. not normalized)
- ▶ Some benefits:
 - ▶ Star schema is easier to understand
 - ▶ Redundancy in the data set simplifies the ETL process. (less primary key / foreign key relationship to maintain)
 - ▶ Better query performance because of smaller number of table joins
- ▶ *Yet sometimes we need to partially normalize dimension tables....*

Snowflake Schema

- ▶ **Snowflaking** is a method of (partially) normalizing the dimension tables in a star schema.
 - ▶ No need to do full normalization (e.g. 3NF)
 - ▶ The result is known as **snowflake schema**
- ▶ Break transitive functional dependency by moving some attributes to separate tables and relate these tables with new surrogate keys



Example: Snowflake Schema



SALES (keyS: STORE, keyD: DATE, keyP: PRODUCT, quantity, receipts, ...)

/* Dimension tables */

STORE (keyS, store, keyCity: CITY, keySalesMgr: SALES_MGR)

CITY (keyCity, city, state, country)

SALES_MGR (keySalesMgr, salesManager)

PRODUCT (keyP, product, type, keyCat: CATEGORY, keyBrand: BRAND)

CATEGORY (keyCat, category, department)

BRAND (keyBrand, brand)

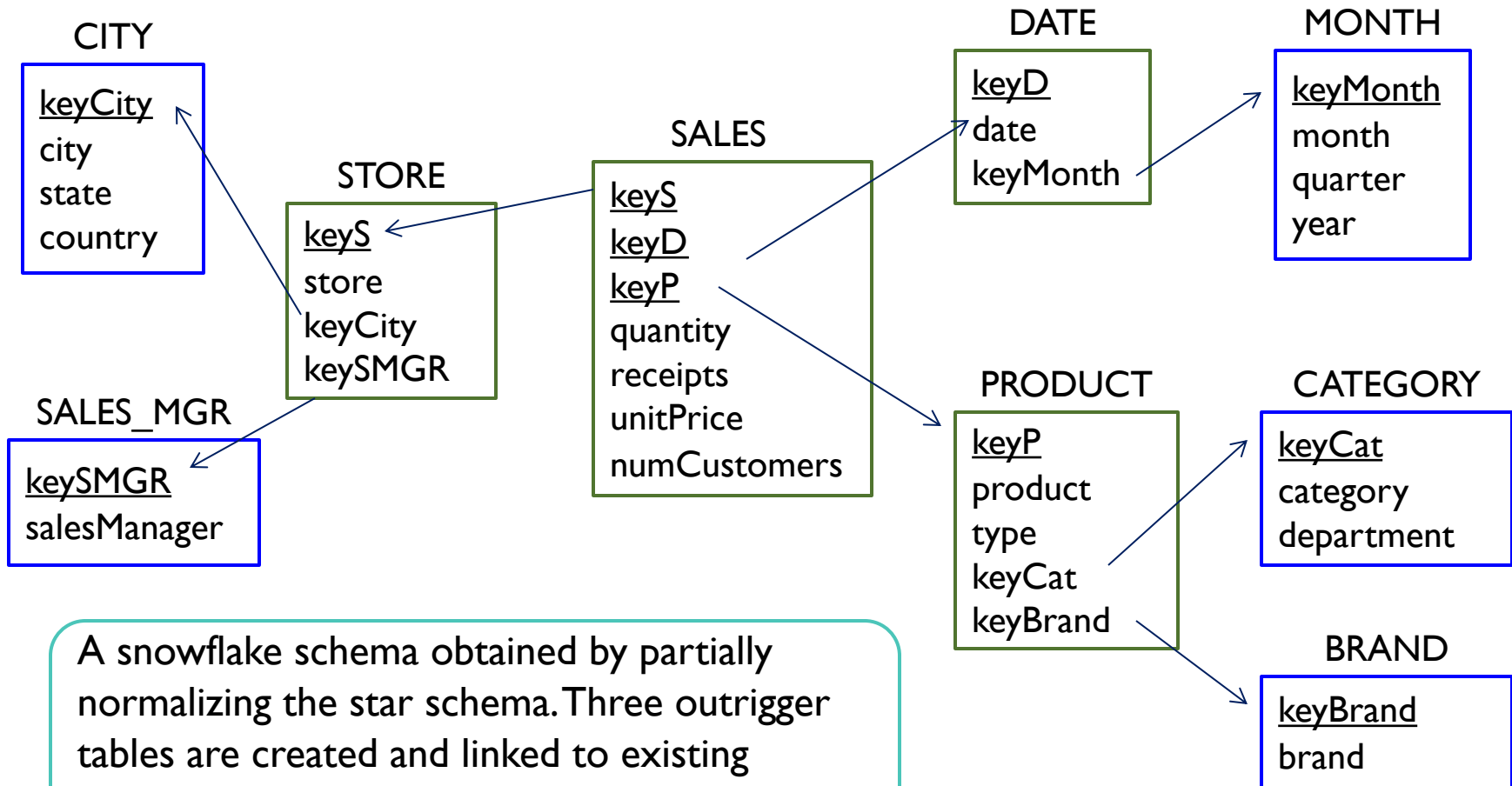
DATE (keyD, date, keyMonth: MONTH)

MONTH (keyMonth, month, quarter, year)

New keys created by
snowflaking

New tables in snowflaking are
sometimes called **outriggers** or
secondary dimension tables.

Example: Snowflake Schema



Example: Query on snowflake schema

- ▶ More table joins are required in queries for a snowflake schema
 - ▶ Possibly slower query processing

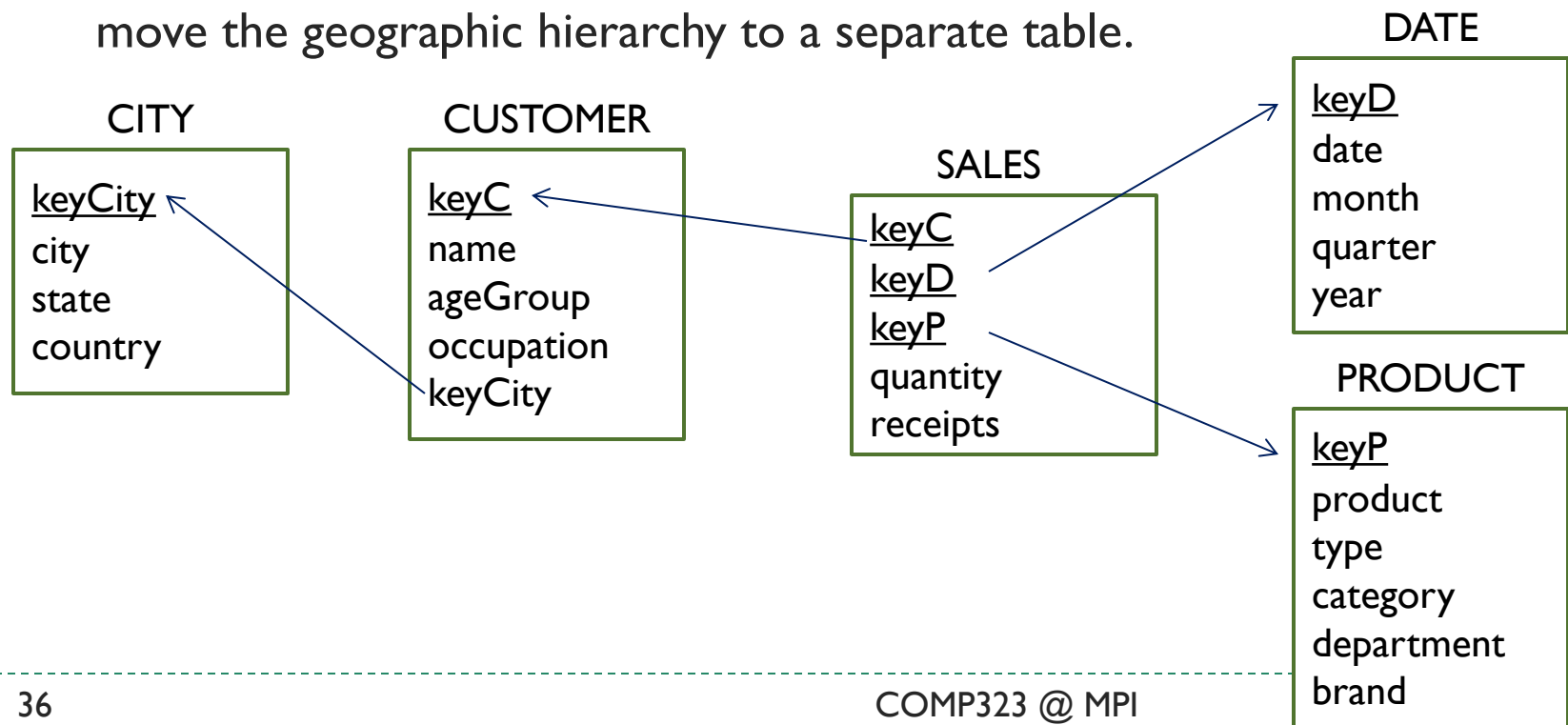
```
SELECT C.city, P.type, Sum(F.receipts)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
  JOIN PRODUCT P    ON F.keyP = P.keyP
  JOIN MONTH M      ON D.keyMonth = M.keyMonth
  JOIN CITY C       ON C.keyCity = S.keyCity
WHERE M.month = 'Jan 2012'
GROUP BY C.city, P.type;
```

When to Snowflake, 1

- ▶ Some OLAP tools and reporting tools function better under the more normalized conditions of a snowflake design
- ▶ Snowflake schema makes explicit some functional dependency relationship between dimensional attributes. These may be necessary for a tool to support aggregates and drill-across.
- ▶ Some tools are optimized for snowflake schema

When to Snowflake, 2

- ▶ Separate a hierarchy from a very large dimension to save storage space
 - ▶ E.g. the Customer dimension in the Sales fact schema of a large online shopping mall may have over 100m rows. Attributes in geographic hierarchy (e.g. city, state, country) are repeated many times. We can move the geographic hierarchy to a separate table.



Example: Eliminating repeating groups with outriggers

DAY

day_key
day_of_week_name
day_of_month
holiday_flag
weekday_flag
weekend_flag
month_name
quarter
year
fiscal_period
fiscal_year

SALESREP

salesrep_key
salesperson_name
hire_day_key
last_review_day_key
territory
region_name
region_manager
reporting_location_key
work_location_key

ORDERS (fact)

...

CUSTOMER

PRODUCT

LOCATION

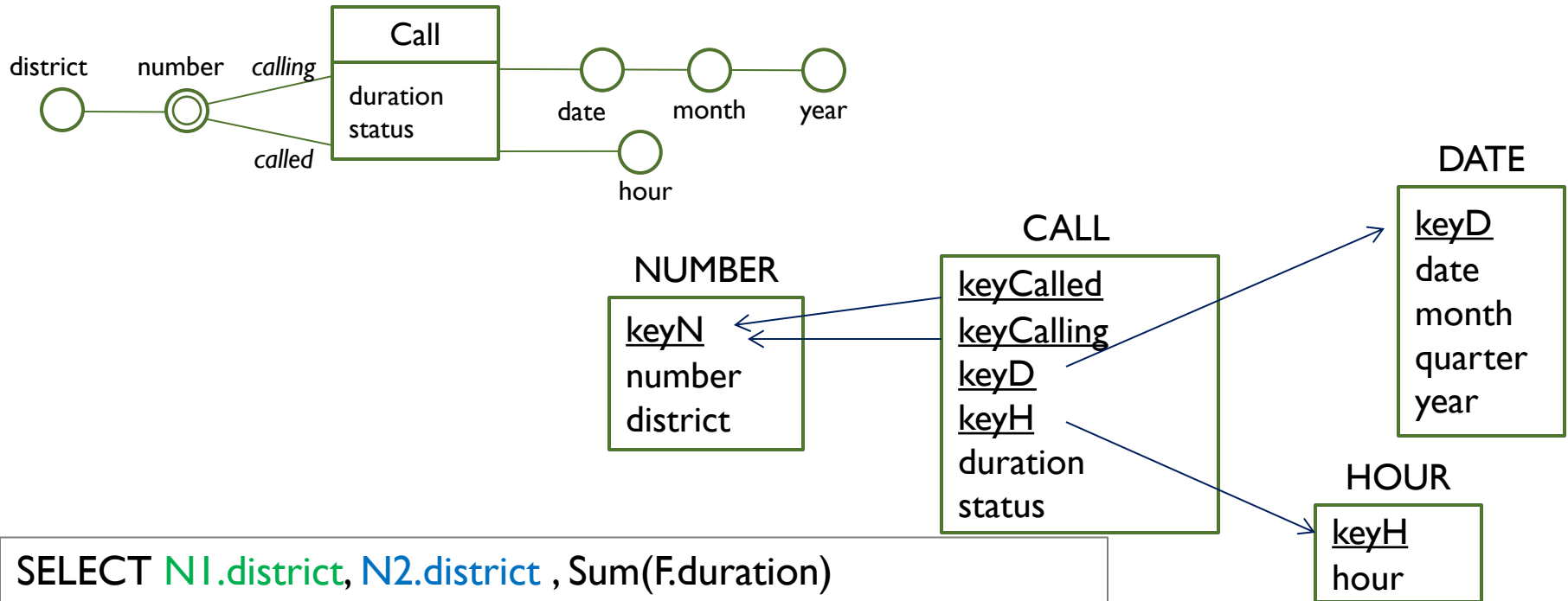
location_key
location_name
location_city
location_state
location_country

The two outrigger tables prevent repeating a long list of attributes for 'hire day', 'last review day', 'reporting location' and 'work location' in the SALESREP dimension table.

When to Snowflake, 3

- ▶ **Sharing a hierarchy among dimensions**
 - ▶ E.g. the geographic hierarchy may be shared by the shipping address of orders, and location of stores and warehouses.
- ▶ **Two types:**
 - ▶ Total sharing – two hierarchies contain exactly the same attributes used with different meanings.
 - ▶ Partial sharing – two hierarchies share some of their attributes
- ▶ **Sharing (part of) a hierarchy among aggregate fact tables and base fact table.** (to be discussed in next topic)

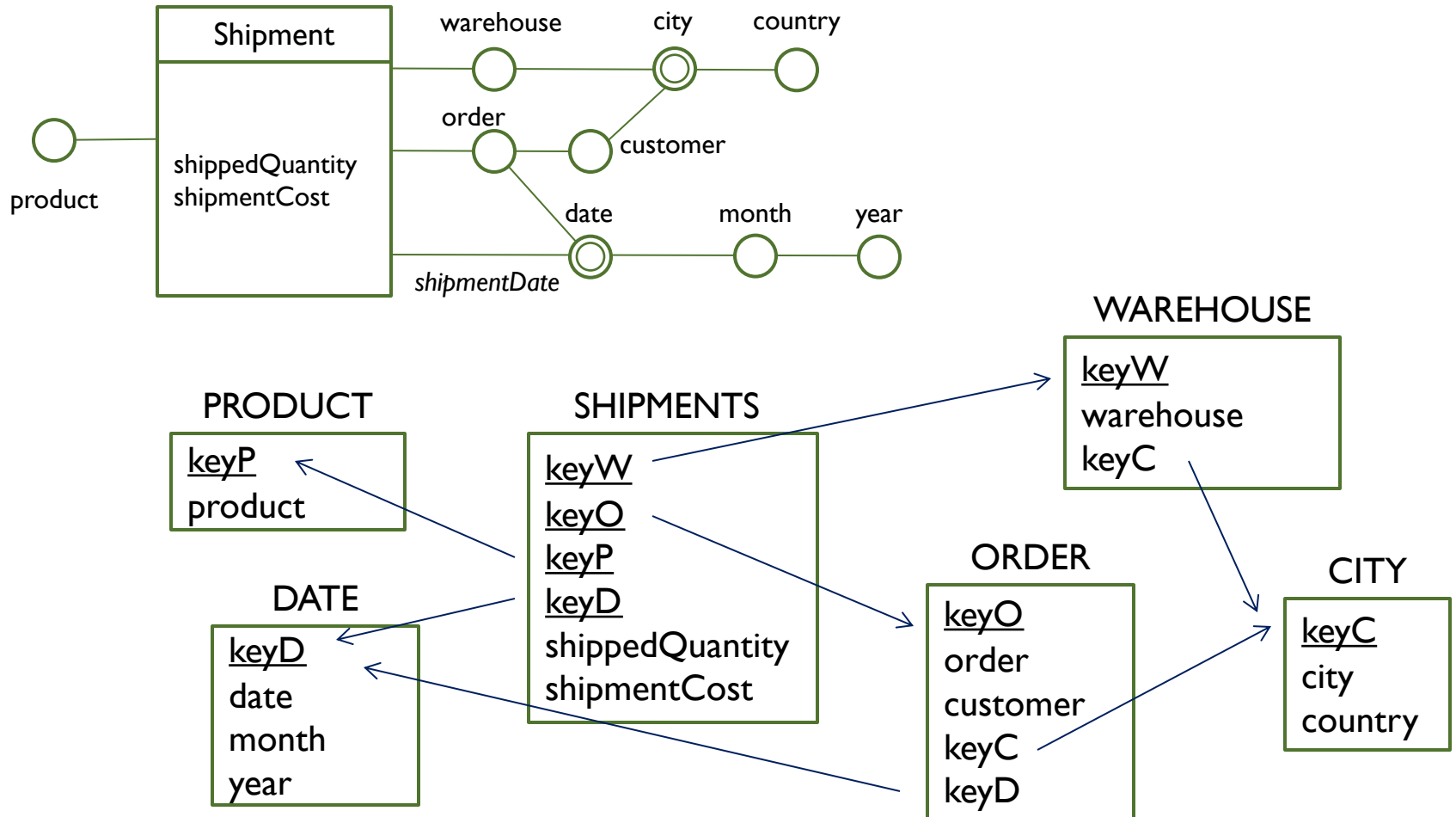
Example: Total Sharing



```

SELECT N1.district, N2.district , Sum(F.duration)
FROM CALL F
  JOIN NUMBER N1      ON F.keyCalled = N1.keyN
  JOIN NUMBER N2      ON F.keyCalling = N2.keyN
  JOIN DATE D         ON D.keyD = F.keyD
WHERE D.month = 'Jan 2012'
GROUP BY N1.district, N2.district
    
```

Example: Partial Sharing



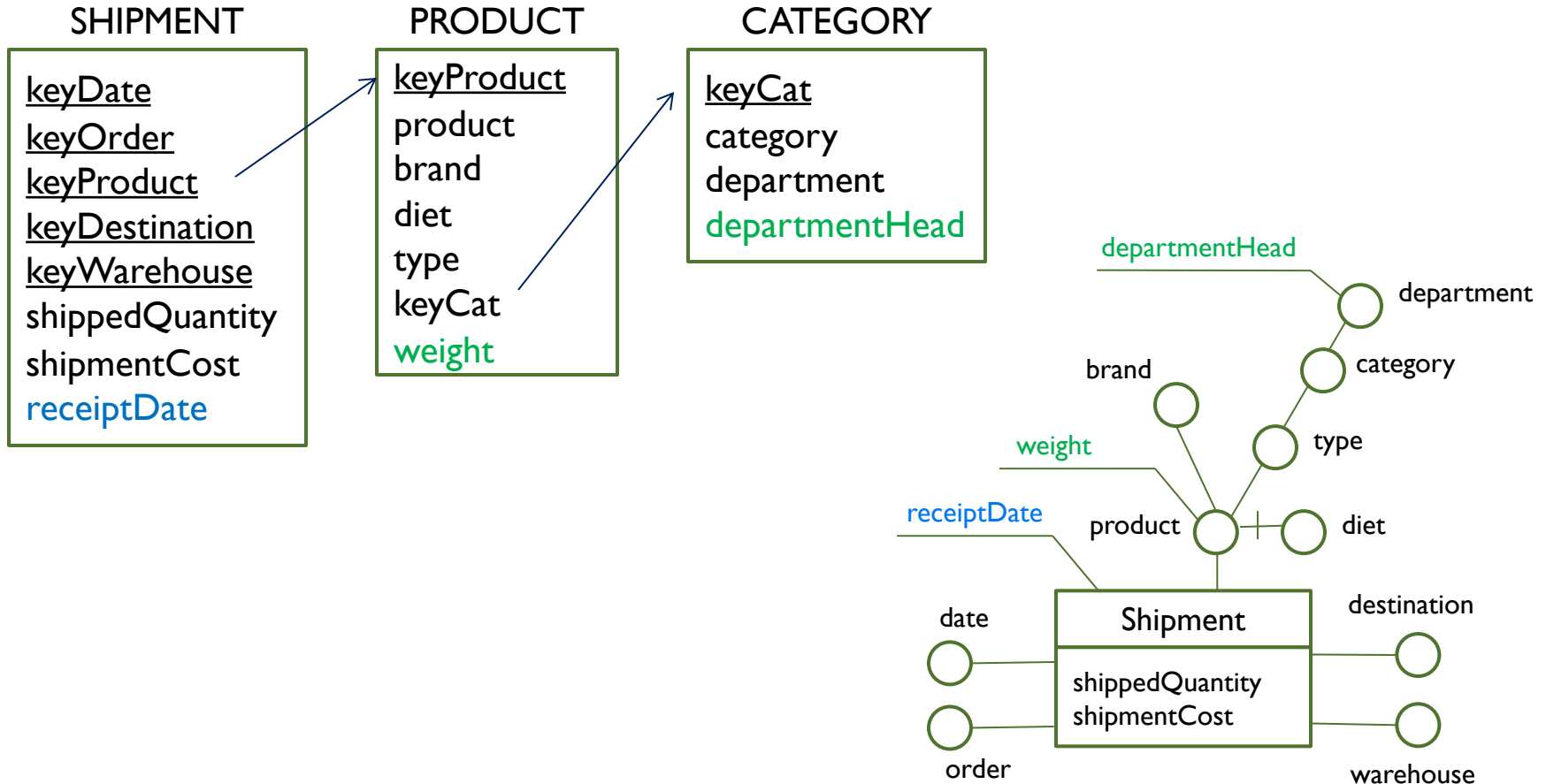
Query exercise

- ▶ Q1. What is the average shipment cost from the warehouses in the city 'A'?
- ▶ Q2. What is the total shipment quantity for the product 'X' to customers in the city 'B'?
- ▶ Q3. How many shipments are ordered in 2012, but shipped in 2013?
- ▶ Q4. What is the shipment costs in 2012? Break down the numbers by warehouse city and customer city.

Descriptive attributes

- ▶ A descriptive attribute contains information that cannot be used for aggregating, but that is considered useful to retain
- ▶ A descriptive attribute linked to a dimensional attribute must be included in the dimension table for the hierarchy that contains it.
- ▶ A descriptive attribute linked to a fact must be included in the fact table.

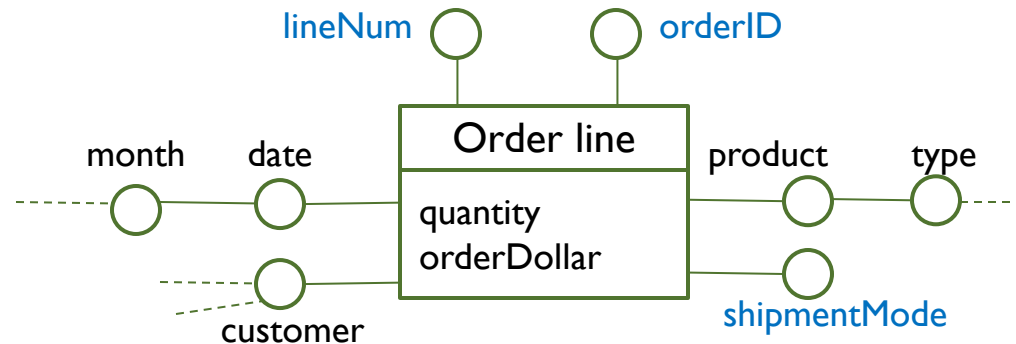
Example



Degenerate dimensions

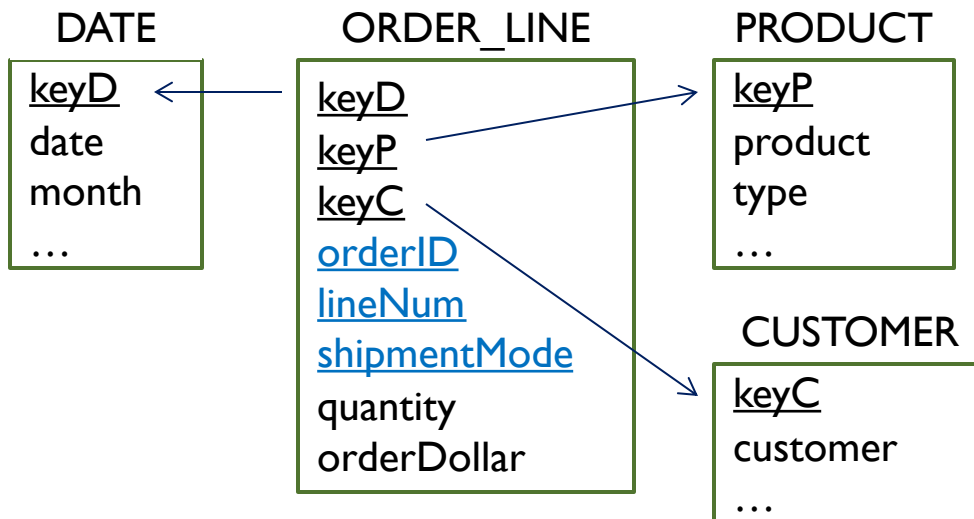
- ▶ Some dimensions have only one attribute
- ▶ If the size of the attribute is smaller than a surrogate key (e.g. ≤ 4 bytes), or if the cardinality of the attribute is very large (e.g. transaction ID),
 - ▶ creating a separate dimension table waste storage space and requires a table join in query processing
 - ▶ degenerate dimension is a better choice!
- ▶ In **degenerate dimension**, we store the single attribute of the dimension directly in the fact table
 - ▶ There is no foreign key and surrogate key for degenerate dimension
 - ▶ This attribute is part of the composite primary key of the fact table
 - ▶ Such degenerate dimension can still be used to filter and group data in fact.

Example: Degenerate Dimension



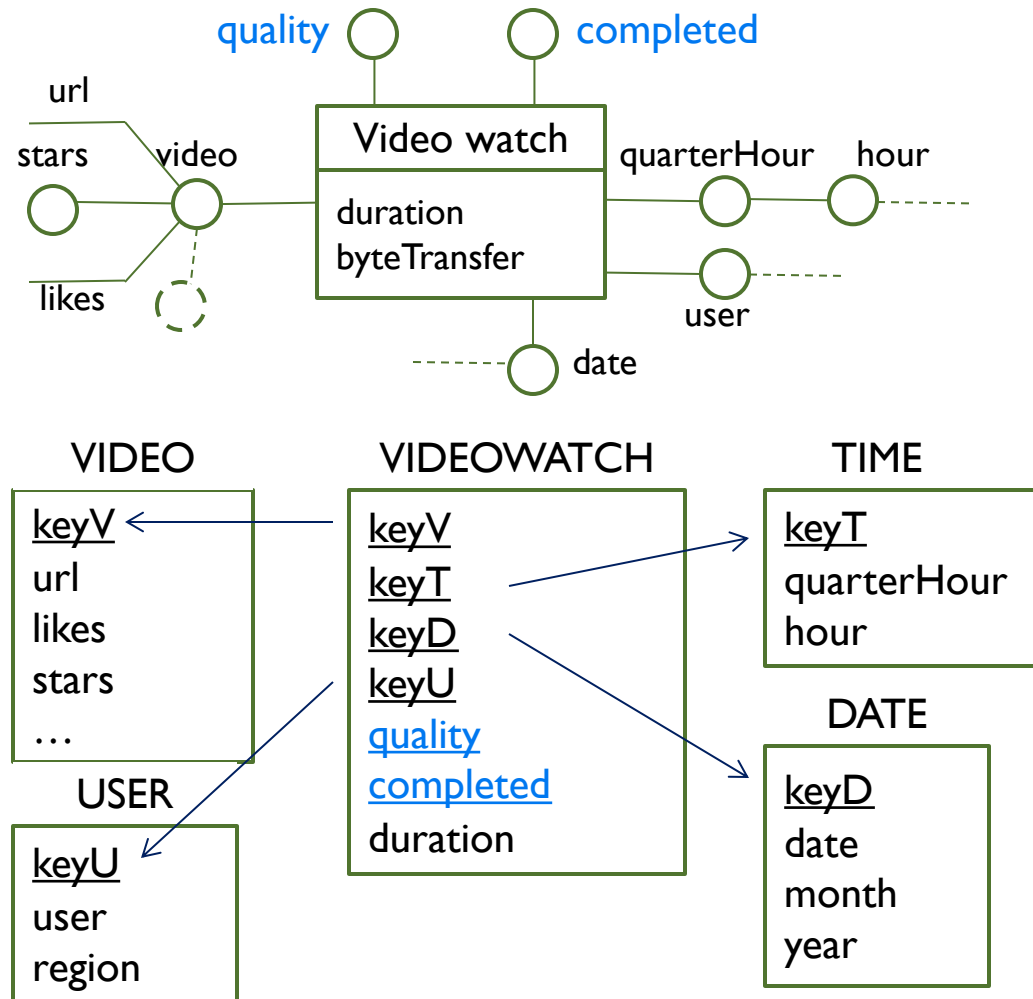
```
SELECT F.orderID, Sum(F.orderDollar)
FROM ORDER_LINE F
JOIN DATE D ON D.keyD = F.keyD
WHERE D.month='2013 Apr'
AND F.shipmentMode='reg'
GROUP BY F.orderID
```

What is the total order dollar of each order in Apr 2013? Restrict the data to orders with shipment mode 'regular'.



Example: Online video watching

- ▶ A fact that supports analysis of video watch duration and count by time of day, date, video attributes and user attributes.
- ▶ In addition, statistics can be filtered and/or grouped by
 - ▶ Quality – 'high', 'low'.
Version of video watched
 - ▶ Completed – 'all', 'part':
whether the video is watched fully.
 - ▶ Discussion: why degenerate dimensions?



Exercise

► Write SQL to answer these analysis queries

- 1) Which video is the most popular in Mar 2013? Restrict your statistics to cases where the videos are watched to the end.
- 2) What are the total duration of video served to users in Mar 2013? Break down the numbers by video quality and user region.
- 3) Compare the percentage of videos watched in high quality during peak hours (e.g. 8:00pm-1:00am) with that in non-peak hours.

Slowly-Changing Dimensions

- ▶ ETL process constantly inserts new rows to fact tables. Occasionally, it also needs to insert new rows to dimension tables
 - ▶ E.g. new products, new customers
- ▶ In general, existing rows in fact tables do not change
- ▶ Sometimes, changes in operational databases may result in changes in some dimensional attributes. E.g.
 - ▶ correcting an error in the birthday of a customer
 - ▶ a customer moves to a new address
- ▶ The problem of how to update the dimension tables to reflect such changes is known as **Slowly-Changing Dimensions**

	Action	Effect on events
Type 1	Update dimension	Restates history
Type 2	Insert new row in dimension table	Preserves history

Example

- ▶ On 1/1/2009, Smith became the sales manager of a new store 'NEW'. Johnson became the sales manager for the store 'A'.

Status on 1/1/2008

store	salesManager
A	Smith
B	Johnson
C	Johnson

Status on 1/1/2009

store	salesManager
A	Johnson
B	Johnson
C	Johnson
NEW	Smith

Sales events

store	date	quantity
A	8/2/2008	100
B	18/10/2008	100
C	25/12/2008	100
A	8/2/2009	100
NEW	5/7/2009	100
B	18/10/2009	100
C	25/12/2009	100

Type 1 Change

- ▶ Type 1 overwrites the attribute value in the dimension table row.
- ▶ Preexisting events have a new context. All the events including past ones are always interpreted from the viewpoint of the current value dimension attributes.
- ▶ Used when no need to preserve the old value. E.g. in case of correction of errors in operational systems

Example: Type 1 change

Dimension table STORE, as on 1/1/2008

<u>keyS</u>	store	salesManager	...
1	A	Smith	...
2	B	Johnson	...
3	C	Johnson	...

Dimension table STORE, as on 1/1/2009

<u>keyS</u>	store	salesManager	...
1	A	Johnson	...
2	B	Johnson	...
3	C	Johnson	...
4	NEW	Smith	...

Status on 31/12/2008

	Year
Sales Manager	2008
Johnson	200
Smith	100

Status on 31/12/2009

	Year	
Sales Manager	2008	2009
Johnson	300	300
Smith	-	100

Amount of items sold by various sales managers

Type 2 Change

- ▶ Type 2 change inserts a new row to store new values into the dimension table.
- ▶ New events are associated with the new row while old events still link to the original row.
- ▶ Historic context of events are preserved. An event is associated with the dimension row that was valid when the event took place
- ▶ May result in large dimension table in case of frequent changes.
 - ▶ Snowflaking may partially solve the problem
- ▶ Surrogate keys support adding a new row that share the same natural keys for two versions of a dimension instance

Dimension and Fact tables in Type 2

Dimension table STORE

<u>keyS</u>	store	salesManager	...
1	A	Smith	...
2	B	Johnson	...
3	C	Johnson	...
4	NEW	Smith	...
5	A	Johnson	...

keyS in the fact table that corresponds to the sales events

keyS	store	date	quantity
1	A	8/2/2008	100
2	B	18/10/2008	100
3	C	25/12/2008	100
5	A	8/2/2009	100
4	NEW	5/7/2009	100
2	B	18/10/2009	100
3	C	25/12/2009	100



Notice the new event for the store 'A' in the fact table links to the new row in the dimension table.

Example: Type 2 change

Dimension table STORE, as on 1/1/2008

<u>keyS</u>	store	salesManager	...
1	A	Smith	...
2	B	Johnson	...
3	C	Johnson	...

Dimension table STORE, as on 1/1/2009

<u>keyS</u>	store	salesManager	...
1	A	Smith	...
2	B	Johnson	...
3	C	Johnson	...
4	NEW	Smith	...
5	A	Johnson	...

Status on 12/31/2008

	Year
Sales Manager	2008
Johnson	200
Smith	100

Status on 31/12/2009

	Year	
Sales Manager	2008	2009
Johnson	200	300
Smith	100	100

Amount of items sold by various sales managers

Aggregation by an attribute that has / has not changed

- ▶ If a query selects one value from an attribute that was changed, this approach can accurately determine the events related to that specific value.

```
SELECT S.salesManager, D.year, SUM(F.quantity)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
GROUP BY S.salesManager, D.year
```

Amount of items sold by sales managers by year

- ▶ All the events are selected without any distinction if a query constrains only the attributes whose values show no previous change.

```
SELECT S.city, D.year, SUM(F.quantity)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
GROUP BY S.city, D.year
```

Amount of items sold by city by year

Discussion

- ▶ Determine whether to use Type 1 or Type 2 response, and describe changes to the dimension table
 - ▶ Membership changes from 'bronze' to 'silver'
 - ▶ Customer moves to 'Los Angeles', 'CA'
 - ▶ Customer's gender corrected to 'M'
 - ▶ Customer moves to 'San Diego', 'CA'
- ▶ What if these changes are done in sequence?

Customer dimension table

<u>keyC</u>	name	gender	city	state	membership
100	Peter	F	Reno	NV	bronze

Review questions

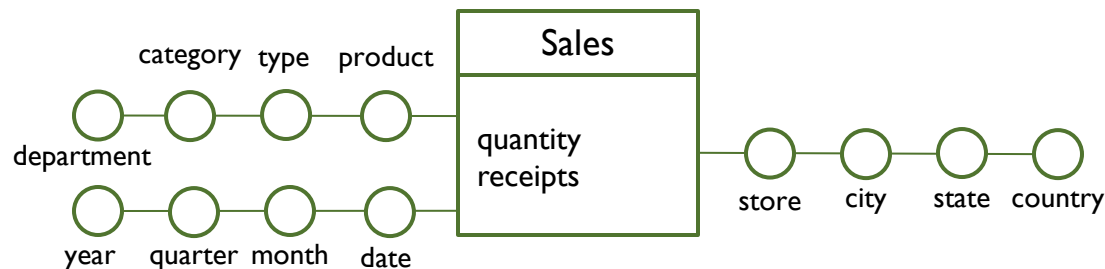
- ▶ How does a snowflake schema differ from a star schema? Name some advantages and disadvantages of the snowflake schema.
- ▶ Describe how you can handle optional arc and dimension in logical design. Why you cannot use a NULL for an optional dimension in the fact table?
- ▶ When should you use degenerate dimensions?
- ▶ Describe slowly-changing dimensions. What are the common ways to handle SCD changes?

Part C. Aggregate Fact Tables

- ▶ Why aggregate before query?
- ▶ How to create? Choose aggregation levels
- ▶ Logical schema to implement aggregate fact tables
 - ▶ Importance of conformance
- ▶ Rewriting query: SQL example

Low level of granularity: reason and consequence

- ▶ Many queries require detail data on one or more dimensions, but only summary totals based on the other dimensions.
- ▶ Detail data at the lowest level of granularity is required in the base fact table.
- ▶ Analysis queries retrieve a large number of rows in fact tables! Slow query processing!



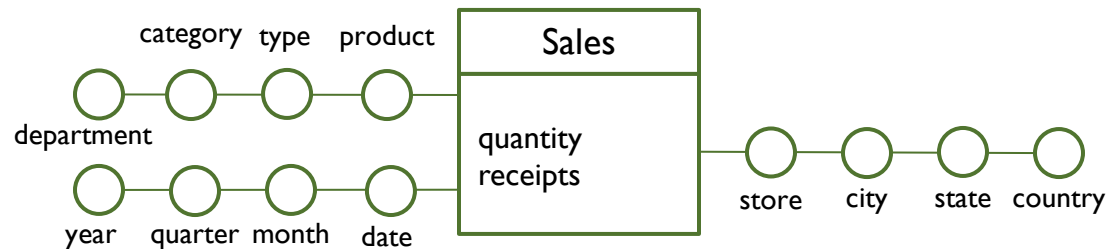
How did the three new stores in Wisconsin perform during the last three months compared to the national average?

What is daily trend of sales on meat and poultry during summer?

What are the best selling dairy products in last quarter?

Aggregate Fact Tables

- ▶ An **aggregate fact table** is a summary of the base fact table at higher levels along the dimension hierarchies.
- ▶ pre-calculated in ETL process and stored as a table in the database, or
- ▶ materialized view: database system support automatic update with the base fact table



An aggregate fact table for 'sales per product type per store per month' stores a summary of the base fact table.

```
CREATE MATERIALIZED VIEW Sales_Aggr_I
AS
SELECT P.type, S.store, D.month, Sum(F.receipts), Sum(F.quantity)
FROM SALES F
  JOIN STORE S          ON F.keyS = S.keyS
  JOIN DATE D           ON F.keyD = D.keyD
  JOIN PRODUCT P        ON F.keyP = P.keyP
GROUP BY P.type, S.store, D.month;
```


Query performance gain from aggregates

- ▶ Aggregate fact tables are usually smaller than the base fact tables.
- ▶ Aggregate fact tables can highly improve the performance of DW queries if the queries are run against the aggregate fact table instead of the base fact table.
 - ▶ E.g. consider calculating the total sales in 2013 by product type

<u>keyS</u>	<u>keyD</u>	<u>keyP</u>	quantity	receipts
I	I	I	15	85
...

Fact table: SALES

<u>keyS</u>	<u>keyMonth</u>	<u>keyType</u>	quantity	receipts
I	10	20	17000	98000
...

Aggregate fact table: Sales_Aggr_I:
SALES per product type per store per month

Date dimension: 30 days
Store dimension: 300 stores reporting daily sales
Product dimension: 40000 products in each store
(about 4000 sell in each store daily)
Sparsity: 10%
Max number of rows in base fact table for 1 week:
 $30 \times 300 \times 40000 \times 10\% = 3.6e7$


Assume there are 500 products per product type, and there is at least a sale for each type in each store in a month. (**Sparsity = 100%**). The max number of rows in the aggregate fact table for 1 month is only $(40000/500) \times 300 \times 1 = 2.4e4$

Query rewrite for materialized view

- ▶ Some DBMSes (e.g. Oracle, MS SQL server) support automatic query rewrite to take advantage of aggregate fact table.
- ▶ The DBMSes choose the aggregate fact tables with the least number of rows and rewrite a query that uses the base fact table.

```
SELECT P.type, Sum(F.receipts), Sum(F.quantity)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
  JOIN PRODUCT P    ON F.keyP = P.keyP
WHERE D.month IN ('2013 Jan', '2013 Feb', '2013 Mar')
GROUP BY P.type;
```

The aggregate fact table Sales_Aggr_I summarizes SALES per product type per store per month

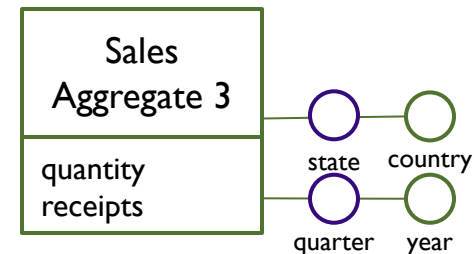
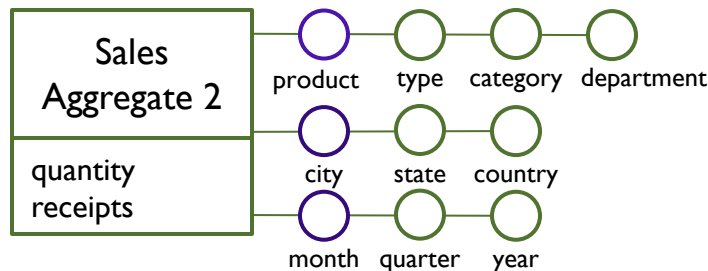
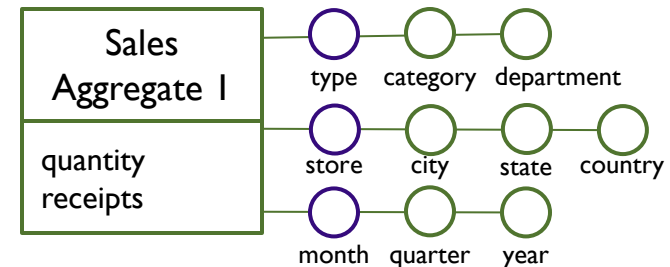
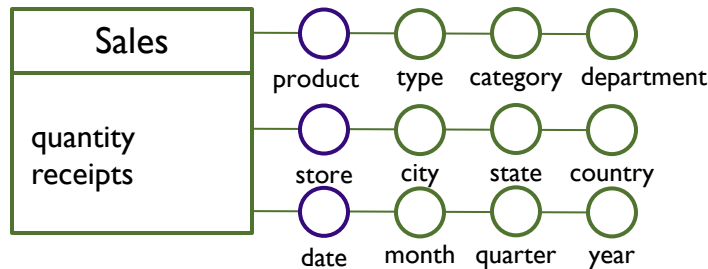


Query rewritten to run on the aggregate fact table

```
SELECT P.type, Sum(F.receipts), Sum(F.quantity)
FROM Sales_Aggr_I F
WHERE D.month IN ('2013 Jan', '2013 Feb', '2013 Mar')
GROUP BY P.type,
```

Variety of aggregate fact tables

- ▶ You can create an aggregate fact table by choosing a dimensional attribute along each dimension. You may also aggregate the total of 'all products', 'all stores' and 'all dates' by not including any attributes for the dimension.



Exercise

- ▶ Discuss which aggregate fact tables can be used to answer the following queries:
 - 1) Total receipts by store city by product type for last month
 - 2) Total receipts by product type in each store in the city 'Macao' for last month
 - 3) Which products generate the highest sales receipt in each city for last month?
 - 4) Which products generate the highest sales receipt in each city during weekends (Saturdays and Sundays)
 - 5) Total sales receipts by quarter by state
 - 6) Total sales receipts by quarter by city

Conformance of Aggregate Fact Tables

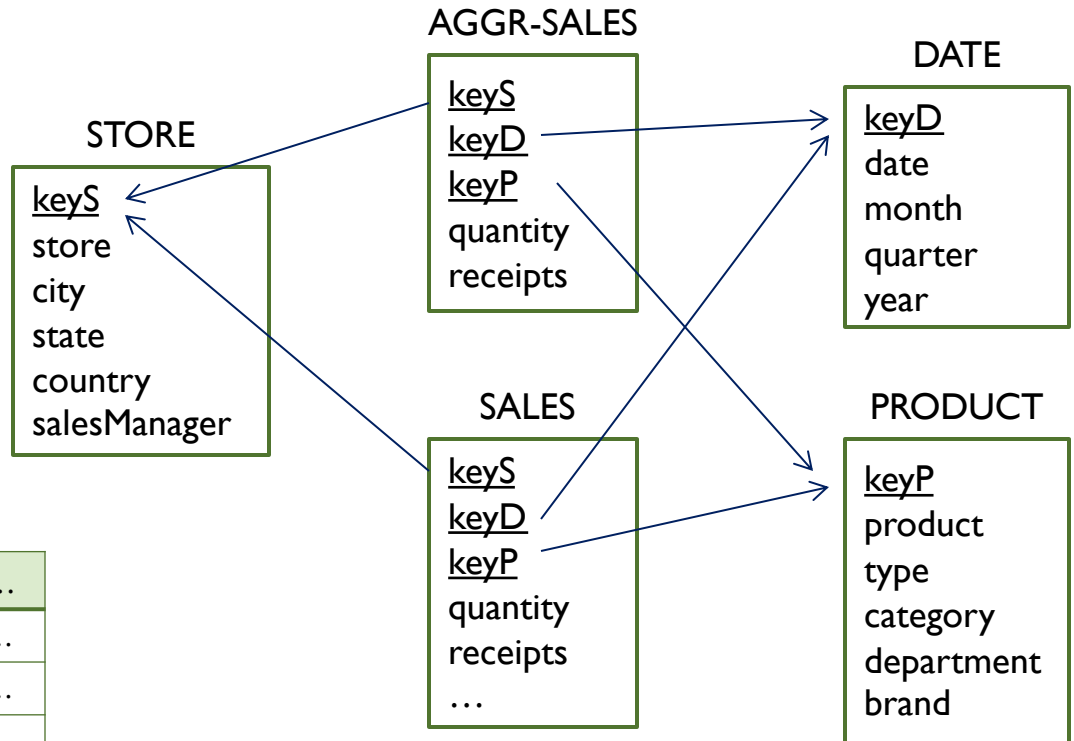
- ▶ To facilitate query writing, aggregate fact tables should use conformed dimensions with the base fact table.
 - ▶ Dimension tables have same structure and content
- ▶ In addition, the summarized measures in aggregate fact table should be consistent with the base fact table
 - ▶ When new events are inserted in base fact table, the aggregate fact table must be updated properly

Logical schema for aggregate fact tables

- ▶ Three common ways to define dimension tables
 - ▶ **Constellation schema** – dimension tables contain rows with NULL to represent aggregation levels
 - ▶ Dimension tables grow large when you add rows for multiple levels in a hierarchy
 - ▶ Not recommended
 - ▶ **Multiple star schemata** – separate dimension tables for different aggregation levels. May replicate partial hierarchies
 - ▶ Better performance, but waste some storage.
 - ▶ **Snowflake schema** – apply snowflaking at aggregation levels where aggregate fact tables are materialized.
 - ▶ Not replicating data in dimension tables

Example: Constellation schema

The aggregation table is a summary for sales per state per quarter per product. The dimension table STORE has special rows for states. The dimension table DATE has special rows for quarters.



Dimension table: STORE

<u>keyS</u>	store	city	state	...
1	evermore	Columbus	Ohio	...
2	Profit	Austin	Texas	...
3	evermore2	Austin	Texas	...
101	NULL	Columbus	Texas	...
102	NULL	Austin	Texas	...
1001	NULL	NULL	Ohio	...
1002	NULL	NULL	Texas	...

Special rows in the dimension table to represent the states 'Ohio' and 'Texas'.

Example: Multiple star schemata

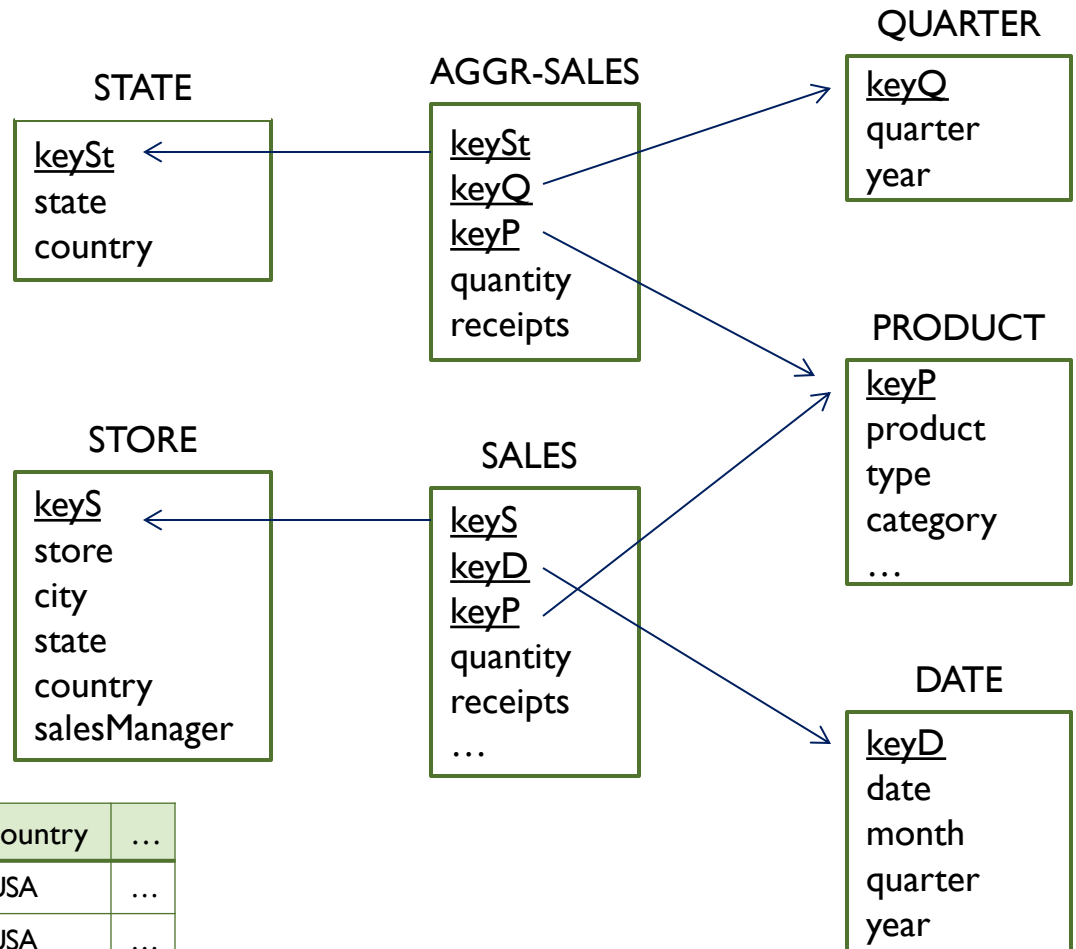
The aggregation table is a summary for sales per state per quarter per product. It links to derived dimension tables QUARTER and STATE.

Dimension table: STATE

<u>keyS</u>	state	country
1001	Ohio	USA
1002	Texas	USA

Dimension table: STORE

<u>keyS</u>	store	city	state	country	...
1	evermore	Columbus	Ohio	USA	...
2	Profit	Austin	Texas	USA	...
3	evermore2	Austin	Texas	USA	...



Example: Snowflake schema

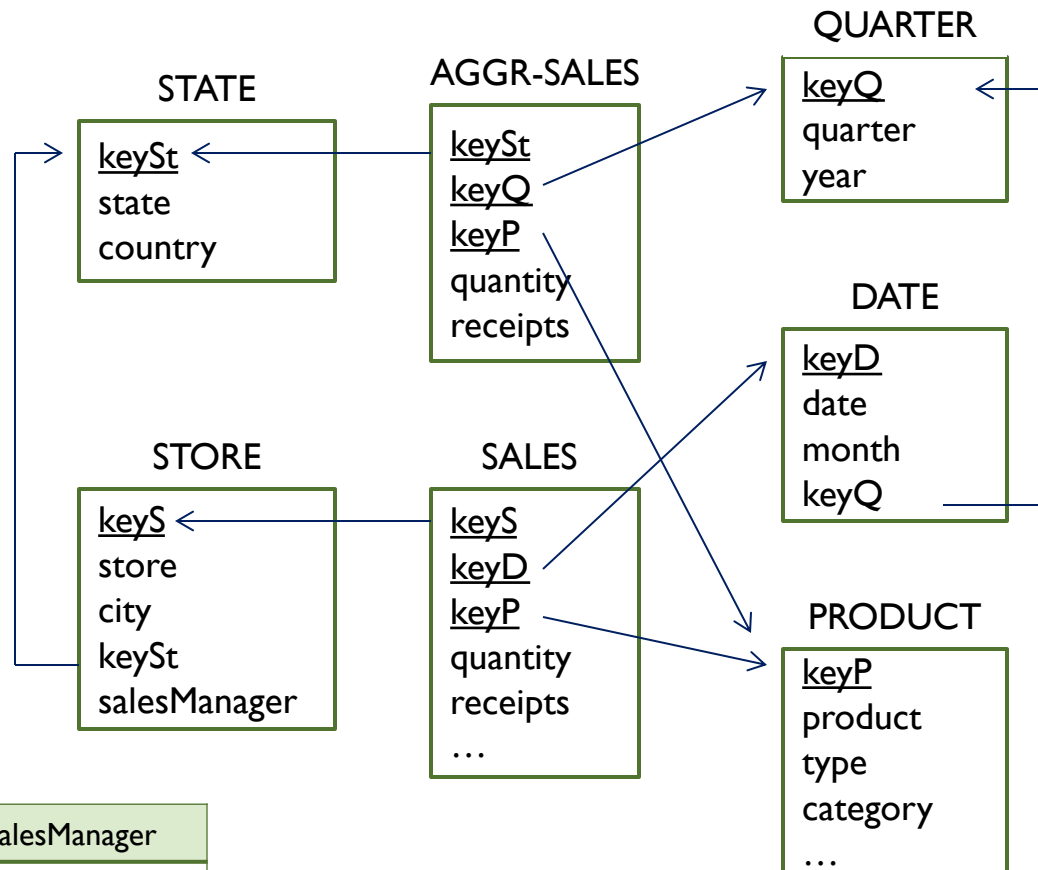
The original dimension table STORE is broken to two tables at the functional dependency $city \rightarrow state$. The secondary dimension table is also used by the aggregate fact table.

Dimension table: STATE

<u>keySt</u>	state	country
1001	Ohio	USA
1002	Texas	USA

Dimension table: STORE

<u>keyS</u>	store	city	keySt	salesManager
1	evermore	Columbus	1001	...
2	Profit	Austin	1002	...
3	evermore2	Austin	1002	...



Simple Query Rewrite for Snowflake schema

- An advantage of the multiple star and snowflake schema is that it is simple to rewrite queries to use aggregate fact tables

```
SELECT P.category, S.state, Sum(F.receipts)
FROM SALES F
  JOIN STORE S          ON F.keyS = S.keyS
  JOIN DATE D          ON F.keyD = D.keyD
  JOIN PRODUCT P       ON F.keyP = P.keyP
WHERE D.year='2013'
GROUP BY P.category, S.state
```

```
SELECT P.category, St.state, Sum(F.receipts)
FROM AGGR-SALES F
  JOIN STATE St         ON F.keySt = St.keySt
  JOIN QUARTER Qt       ON F.keyQ = Qt.keyQ
  JOIN PRODUCT P       ON F.keyP = P.keyP
WHERE Qt.year='2013'
GROUP BY P.category, St.state
```

Review questions

- ▶ What are aggregate fact tables? Why are they needed? Give an example.
- ▶ Describe and compare the three logical designs for aggregate fact tables.
 - ▶ Rewrite an analysis query to use an aggregate fact table

```
SELECT P.category, D.quarter, Sum(F.receipts)
FROM SALES F
  JOIN STORE S      ON F.keyS = S.keyS
  JOIN DATE D       ON F.keyD = D.keyD
  JOIN PRODUCT P    ON F.keyP = P.keyP
WHERE D.year='2013' AND S.country='China'
GROUP BY D.quarter, P.category
```

Further Readings

- ▶ Data warehousing sections in Oracle online library
 - ▶ <http://www.oracle.com/pls/db112/homepage>
- ▶ “Optimizing data warehouse query performance through bitmap filtering”
 - ▶ [http://msdn.microsoft.com/en-us/library/bb522541\(v=sql.105\).aspx](http://msdn.microsoft.com/en-us/library/bb522541(v=sql.105).aspx)
- ▶ Indexed views in MSSQL
 - ▶ <http://gavindraper.com/2012/05/02/sql-server-indexed-views-explained/>
 - ▶ Web search for more
- ▶ Materialized view in Oracle
 - ▶ Oracle Database Datawarehousing Guide