Exercises Data Warehousing
SQL Extensions & DB Explosion Problem

Solutions

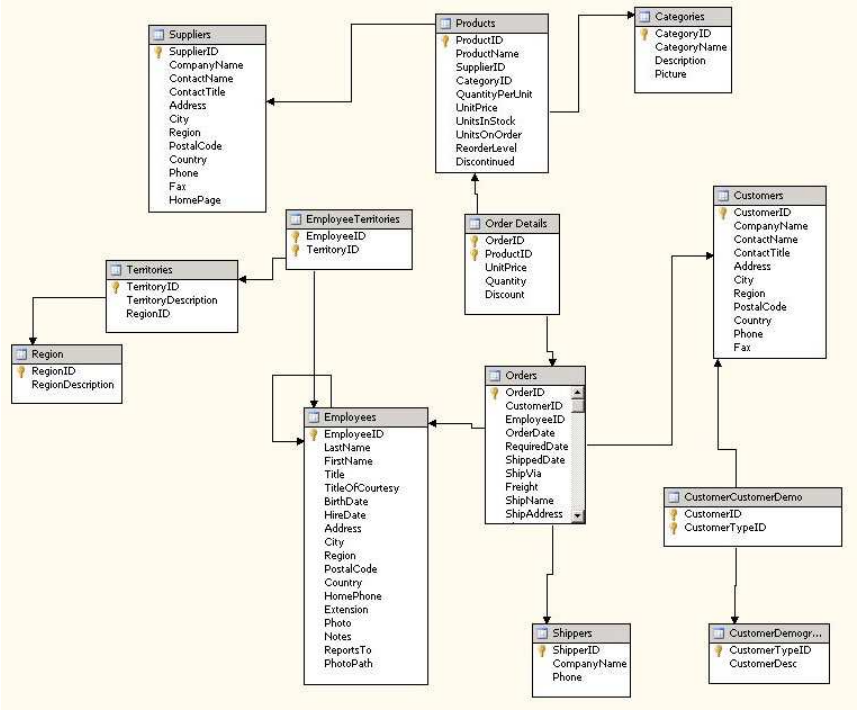


Figure 1: Schema of the Northwind database

1. Consider the Northwind database whose schema is given in Figure 1. This database contains information of orders placed by customers. For every order the detail is given of what products were sold, for what unit price and in what quantity. The employee that secured the order is recorded as well as the date in which the order was inserted. For customers the city they live in etc. is recorded, and for employees their salesdistrict. For this database, create queries to generate the following reports:

   (a) Select the number of sales per category and country.

```
SELECT CategoryName, Country, COUNT(*) AS Count
FROM "Order Details" O, Products P, Categories C, Suppliers S
WHERE O.ProductID = P.ProductID AND P.CategoryID = C.CategoryID AND
      P.SupplierID = S.SupplierID
GROUP BY CategoryName, Country
ORDER BY CategoryName, Country
```

   (b) Select the 3 top-selling categories overall (hint: use "select top 3" construction).

```
SELECT CategoryName, Country, COUNT(*) AS Count
FROM "Order Details" O, Products P, Categories C, Suppliers S
```



```
WHERE O.ProductID = P.ProductID AND P.CategoryID = C.CategoryID AND
      P.SupplierID = S.SupplierID
GROUP BY CategoryName
ORDER BY CategoryName
```

   (c) Produce an overview of sales by month for these categories (hint: get month and year with "month" and "year" functions). Are there countries and product categories for which the trend over time is increasing?

```
WITH Top3Categories AS (
      SELECT TOP 3 C.CategoryId, CategoryName, COUNT(*) AS Count
      FROM "Order Details" O, Products P, Categories C
      WHERE O.ProductID = P.ProductID AND P.CategoryID = C.CategoryID
      GROUP BY C.CategoryId, CategoryName
      ORDER BY Count DESC )
SELECT CategoryName, Country, year(OrderDate) AS Year,
      month(OrderDate) AS Month, COUNT(*) AS Count
FROM Orders O, "Order Details" OD, Products P, Top3Categories C, Suppliers S
WHERE O.OrderID = OD.OrderID AND OD.ProductID = P.ProductID AND
      P.CategoryID = C.CategoryID AND P.SupplierID = S.SupplierID
GROUP BY CategoryName, Country, year(OrderDate), month(OrderDate)
ORDER BY CategoryName, Country, year(OrderDate), month(OrderDate)
```

   (d) List total amount of sales in $ by employee and year (discount in OrderDetails is at UnitPrice level). Which employees have an increase in sales over the three reported years?

```
SELECT FirstName, LastName, year(OrderDate) AS Year,
       FORMAT(SUM((1-Discount)*OD.UnitPrice*Quantity),'C', 'en-us') AS TotalAmount
FROM Orders O, "Order Details" OD, Employees E
WHERE O.OrderID = OD.OrderID AND O.EmployeeID = E.EmployeeID
GROUP BY FirstName, + LastName, year(OrderDate)
ORDER BY FirstName, LastName, year(OrderDate)
```

   (e) Get an individual sales report by month for employee 9 (Dodsworth) in 1997.

```
SELECT month(OrderDate) AS Month,
       FORMAT(SUM((1-Discount)*OD.UnitPrice*Quantity),'C', 'en-us') AS TotalAmount
FROM Orders O, "Order Details" OD
WHERE O.OrderID = OD.OrderID AND O.EmployeeID = 9 AND year(OrderDate) = 1997
GROUP BY month(OrderDate)
ORDER BY month(OrderDate)
```

   (f) Get a sales report by country and month.

```
SELECT Country, year(OrderDate) AS Year, month(OrderDate) AS Month,
       FORMAT(SUM((1-Discount)*OD.UnitPrice*Quantity),'C', 'en-us') AS TotalAmount
FROM Orders O, "Order Details" OD, Products P, Suppliers S
WHERE O.OrderID = OD.OrderID AND OD.ProductID = P.ProductID AND
      P.SupplierID = S.SupplierID
GROUP BY Country, year(OrderDate), month(OrderDate)
ORDER BY Country, year(OrderDate), month(OrderDate)
```

2. The sales department of a supermarket chain wants to have a system to support the strategic planning and evaluation of promotions. To this end, they need sales information over the different stores of the supermarket chain. For their analysis tasks they want to compute average sales and total sales, for different products, either at product level or brand level, for different stores at different levels of granularity: individual store, province where the store is located, and country, and for different time periods: per year, month, quarter, semester and also by day of the week.

   (a) How would you conceptually model the data needed by the sales department as a data cube? E.g., what are the measures, the dimensional attributes, the hierarchies, the aggregations that are needed?
   Solution: The dimensions are as follows

   - Product(Product, Brand, Type),
   - Store(Store, Province, Country), and

- Date(Month, Semester, Year, Weekday).

The measure is sales. The aggregation functions are sum (for total sales) and average (for average sales). The hierarchies are as follows:

Product → Brand
Store → Province → Country
Date → Month → Semester → Year
→ Weekday

(b) Given the cube of (a), explain how you would construct the answers to the following queries with the operations slice-and-dice, pivot, roll-up, and drill-down. If necessary, indicate in which cell(s) of the constructed cube the answer can be found:

i. Give the total overall sales per store.
Solution: Cells (Store, all, all) of the original cube. Slice on Product=all and Date=all. The measure is TotalSales.

ii. Give an overview of the average sales per month per province.
Solution: The measure is AverageSales. Slice on Product=all. Roll-up Store to Province, and Day to Month. Represent it using a pivot-table on dimensions Store and Day.

iii. Give the subcube with only dimensions store at level province and day at level month for the average and total sales for the period 1999 till 2005.
Solution: Slice: date must be in 1999 till 2005. Roll-up Store to Province, Date to Month.

(c) Give an SQL:1999 expression that produces the datacube (i.e., contains all aggregates of the cube using the null value in an attribute to represent aggregation on the corresponding dimension). How do you handle the multiple measures? The hierarchy?

We assume that the base data is stored in the following relational tables:

- Product(ProductID, Brand, Type)
- Store(StoreID, Province, Country)
- Date(Date, Weekday, Month, Semester, Year)
- Sales(ProductID, StoreID, Day, Amount)

```
SELECT P.ProductID, Brand, S.StoreID, Province, Country, D.Day,
      Weekday, Month, Semester, Year, SUM(Amount) AS Total, AVG(amount) AS Average
FROM Product P, Store S, Day D, Sales Sa
WHERE P.ProductID = SA.ProductID and S.StoreID = Sa.StoreID and D.day = Sa.day
GROUP BY ROLLUP(brand,P.ProductID), ROLLUP(country, province, S.StoreID),
         ROLLUP(year, semester, month, D.day), ROLLUP(weekday, D.day);
```

3. Give SQL:1999 expressions for the queries in 2(b).

Solution
Let Cube be the result of the query in 2(c).

- Give the total overall sales per store.

```
SELECT StoreID, Total
FROM Cube
WHERE Brand IS NULL AND Year IS NULL AND Weekday IS NULL AND StoreID IS NOT NULL
```

- Give an overview of the average sales per month per province.

```
SELECT Month, Year, Province, Average
FROM Cube
WHERE Brand IS NULL AND StoreID IS NULL AND Province IS NOT NULL AND
      Date IS NULL AND Month IS NOT NULL AND Weekday IS NULL
```

- Give the subcube with only dimensions store at level province and date at level month for the average and total sales for the period 1999 till 2005.

```
SELECT Month, Year, Province, Average, Total
FROM Cube
WHERE Brand IS NULL AND StoreID IS NULL AND Province IS NOT NULL AND
      Date IS NULL AND  Month IS NOT NULL AND Year >= 1999 AND Year <= 2015 AND
      Weekday IS NULL
```


4. Suppose that we have a relation Sales(Product, Month, Store, Amount). There are five products: P1, P2, P3, P4, P5, 12 months of data, and three stores: S1, S2, and S3.

(a) (Dense setting) Suppose that every product has been sold in every month in every store; i.e., for every combination of a product p, a month m, and a store s, there is a tuple (p, m, s, a) with a non-zero amount.

i. How many tuples does this relation contain?
Solution: $5 \times 12 \times 3 = 180$

ii. How many tuples does a data cube with dimensions Product, Month, Store, and measure Amount contain?
Solution: $6 \times 13 \times 4 = 312$

(b) (Sparse setting) Consider the following (sparse) relation:

Product	Month	Store	Amount
P1	Jan	S1	a1
P1	Jan	S2	a2
P2	Feb	S2	a3
P2	Feb	S3	a4
P3	Jan	S1	a5
P3	Feb	S1	a6
P4	Feb	S1	a7
P5	Jan	S3	a8

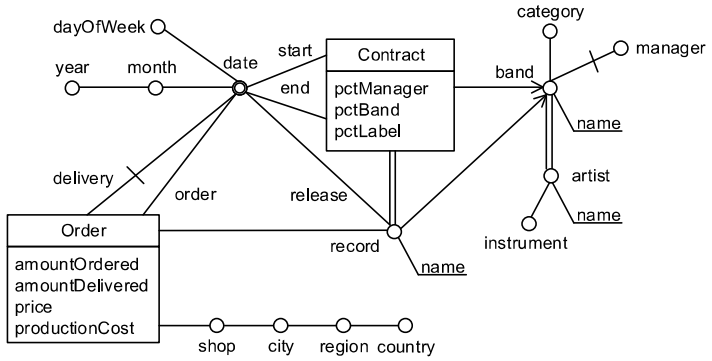
How many non-empty cells does the data cube of this relation contain?
Solution:

Group By	#
PMS	8
PM	6
PS	7
MS	6

Group By	#
P	5
M	2
S	3
()	1

Hence, in total: 38 non-empty cells.

Solution



Common mistakes: dimensional attributes that are incorrectly placed into a hierarchy (for instance, roll-up from band to record); many-to-many relations modeled as one-to-many; models with only one fact usually did not work: contracts are not a property of sales, not vice versa. These are two different subjects in the data warehouse, hence requiring two facts. A third fact, for record, or the production of a record could be considered.

2. **Loan Data Warehouse.** A bank wants to build a data warehouse for storing and analyzing data about all loans issued by them.

   Every loan has one or more borrowers, a starting date, a type (e.g., fixed rate or one of different types of variable rate), the branch of the bank where the loan was issued, the interest rate at the start of the loan, and the amount. For every loan the purpose of the loan is recorded; e.g., to buy a car, a house, a personal loan, ... When a borrower applies for the loan, different discounts on the interest rate may be awarded; e.g., fidelity discount, discount because the borrower also bought some additional insurances, VIP discount, etc. For one loan, multiple discounts may apply. The amount of discount is independent of the branch. Every discount that has been awarded needs to be stored. When the loan ends, this is stored as well, together with an indication if the loan was fully repaid or the borrower defaulted.

   For the borrowers, their date of birth, family status, monthly income, number of children and address is stored.

   The following questions are prototypical for the type of query analysts want to answer based on the data warehouse:

   - Give the average interest rate before discount at the start of the loan, per loan type and branch.
   - For all branches, give the minimum, maximum and average interest rate per loan type and purpose.
   - Give the number of loans per branch and per amount category. The amount category depends on predefined thresholds; amounts are divided into the following classes: `very high`, `high`, `medium`, `low`, and `very low`.
   - Give the percentage of defaulted loans per year and per city of the branch where the loan was issued.

Exercises Data Warehousing Dimensional Modelling

Solutions

To draw the schema, the tool "Indyco Builder" can be used. The link to the tool and personal licenses for all students are available from the course website. The tool is also installed in the computer lab.

1. **Record Label.** A record label wants to keep track of all contracts they have with bands, records they are producing and the sales of these records. Currently they are only keeping data of their ongoing contracts; no historical information is kept. Therefore it is decided to construct a data warehouse for collecting and storing historical information. With the data warehouse the company wants to analyze its sales. Based on conversations with the managers of the company, you were able to compile the following description of the available data.

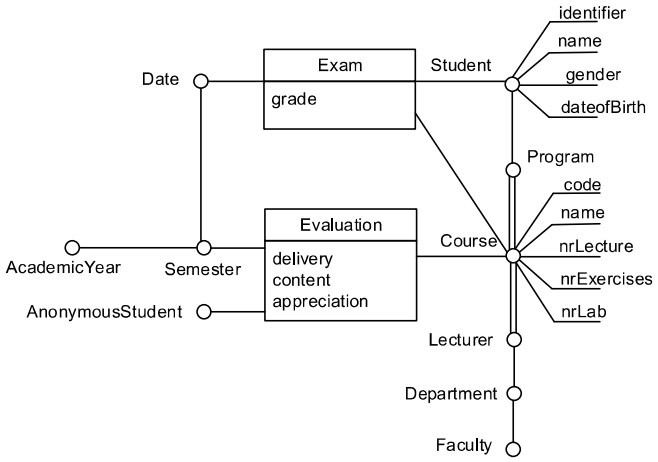
   The record label has several bands under contract. Each band has a name, a main category of music it plays, and one or more artists. A band may have a manager, but does not have to. Bands without a manager must have one of the artists as their main representative. An artist has one main instrument, which can also be his/her voice. Over time, bands can split and attract or replace group members. Bands make records which are physically produced and distributed by the record label. Records, including those that are not yet finished, have a title and are identified by a special international code. For every record the price to produce it and its release date are stored. The record label has contracts with the bands. For every contract start and end data are registered as well as for which records it holds, what percentage of sales goes to the manager of the group (if present), how much to the group, and how much to the record label. A contract can apply to multiple records, but every record falls under exactly one contract. Shops can order records at the record label. The date, amount, and agreed price of these orders are recorded, as well as the number of records that were sent to the shop in the end (sometimes demand outweighs supply) and the date on which the order was fulfilled. For every shop, the record label has information about the spatial location, including: city, country, and region of the shop.

   Some examples of the types of analyses the record label wants to perform based on the data warehouse are the following:

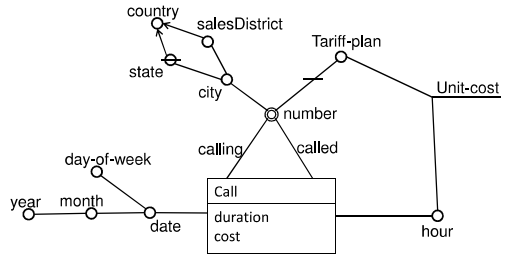
   - Which record/group/artist has given the record label the highest/lowest gain/return on investment per week/month/year.
   - What seller/city/country has the highest number of demands that could not be met?
   - Which record/group/artist has the highest number of demands?

   Make a dimensional fact model for a data warehouse according to the above description.

Solution:



4. Consider the following DFM schema.



How big would the roll-up lattice be for this schema? Draw the roll-up lattice for the dimensions calling and date only.

Solution:
Size of the roll-up lattice:

- calling and called: both 13 grouping sets

  (ALL, country, state, sales-dist, state&sales-dist, city, number, tariff-plan, country&tariff-plan, state&tariff-plan, sales-district&tariff-plan, state&sales-dist&tariff-plan, city&tariff-plan)

- hour: 2 (ALL and hour)
- date: 7

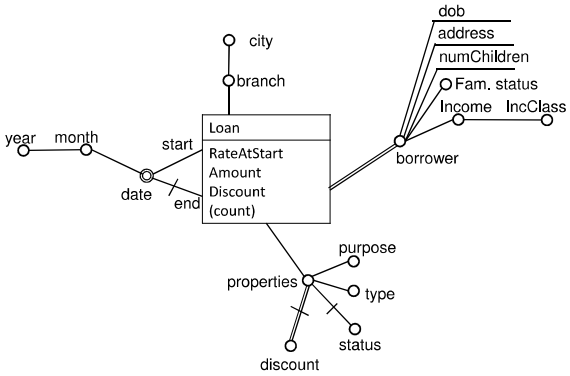
  (All, Year, Day-of-week, Year&Day-of-week, Month, Month&Day-of-week, Date)

Total: $13 \times 13 \times 2 \times 7 = 2366$ ways to roll-up the data

Make a dimensional fact model for a data warehouse according to the above description.

Solution *There is not a single correct answer; different choices may lead to different models. The level of detail given in this explanation was not necessarily expected in your answer.*

A first important observation is what will be the subjects around which we will be building our data warehouse. In this case the most natural choice is "loan." Another option could have been "loan event"; e.g., change in interest rate. This other option, however, has a couple of disadvantages: given the prototypical queries, loan is a more natural choice. Also, it is hard to define meaningful ways to aggregate measures such as amount over these events.



3. **Course Evaluation** A university wants to analyze the results of its students and the quality of its courses over time. For this purpose they want to create a data warehouse that will store the results of all students. A student is registered for a program which consists of several courses taught by one or more lecturers. Every course has a course code, a name, the number of lecturing hours and the number of hours for exercises and lab sessions. A lecturer belongs to a department and a department to a faculty. A student can take one or more exams for a course. In case of multiple exam trials, all results should be stored. For a student, his or her name, student identifier, gender, and date of birth are stored. Furthermore, every course offering is evaluated (anonymously) by the students. This evaluation results in three scores: one for course delivery, one for course content and one for overall appreciation. Based on the data in the data warehouse it should be possible to evaluate the exam results by student, by program, by academic year, by course, by department, etc. Furthermore courses will be evaluated on the basis of their evaluations by year, by lecturer, by program, etc.

Exercises Data Warehousing
From Operational to Multidimensional Databases

Solutions

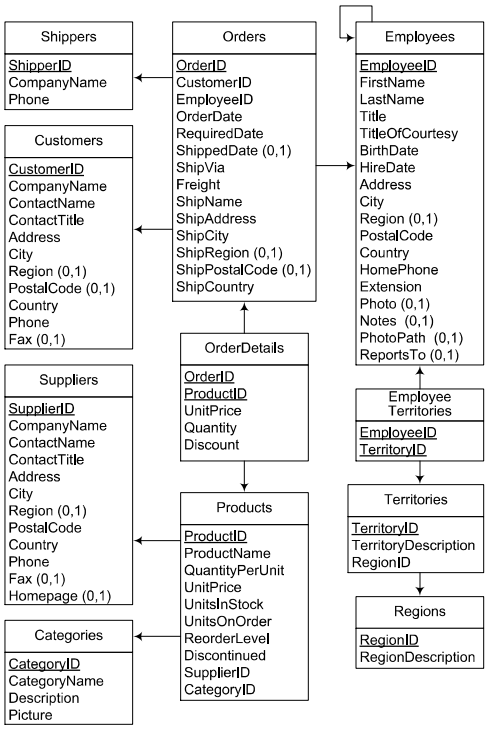


Figure 1: Schema of the Northwind database

Consider the Northwind operational database whose schema is given in Figure 1. This database contains information of orders placed by customers. For every order the detail is given of what products were sold, for what unit price and in what quantity. The employee that secured the order is recorded as well as the date in which the order was inserted. For customers the city they live in etc. is recorded, and for employees their sales district.

For this database, create a data warehouse that will be used for analyzing the sales of the organization. Justify your choices.

One possible conceptual schema for the data warehouse is given in Figure 2 and its relational schema is gien in Figure 3.

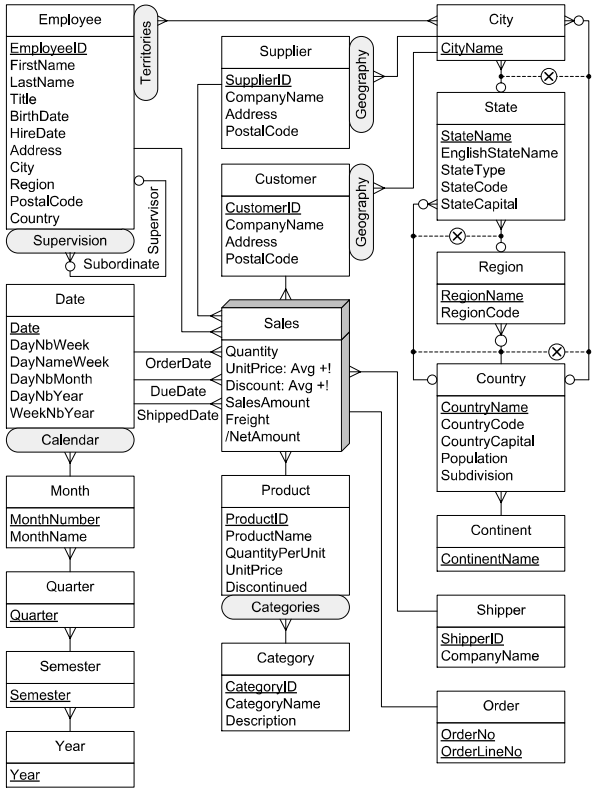
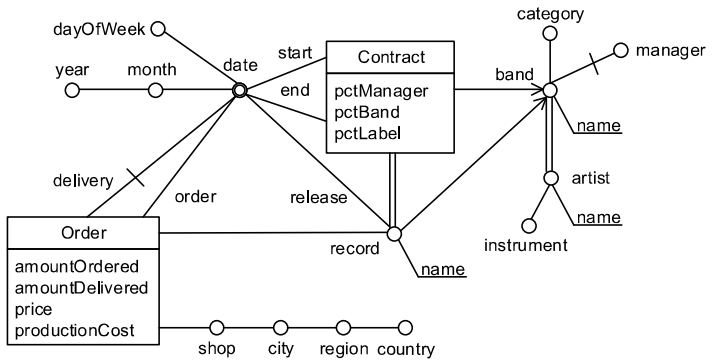


Figure 2: Conceptual schema of the Northwind data warehouse

Exercises Data Warehousing
ROLAP

Solutions

1. Consider the following dimensional fact model for the record label of the last exercise session.



Create a relational model (star or snowflake schema) for this model.

Solution

The relational tables are given in Figure 1. The attributes that together form the primary key have been underlined. The foreign keys are indicated by (FK).

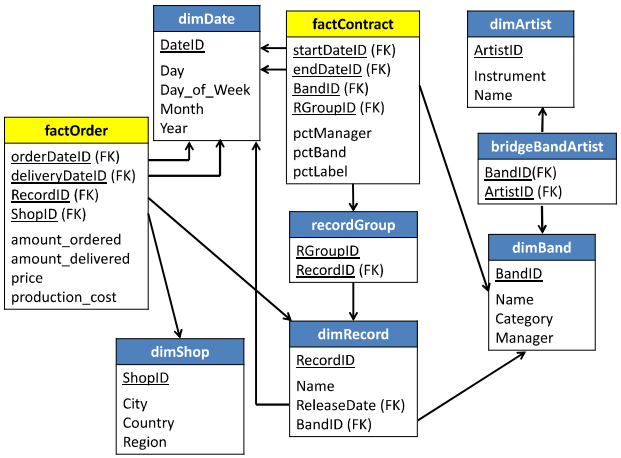


Figure 1: Relational schema of the data warehouse for the record label.

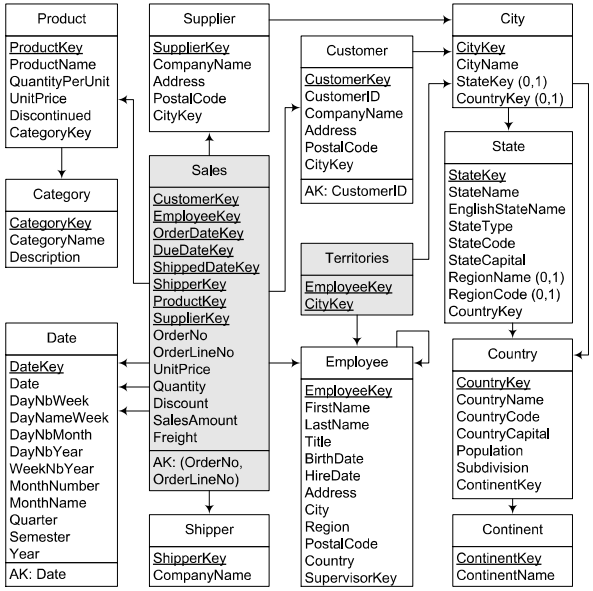


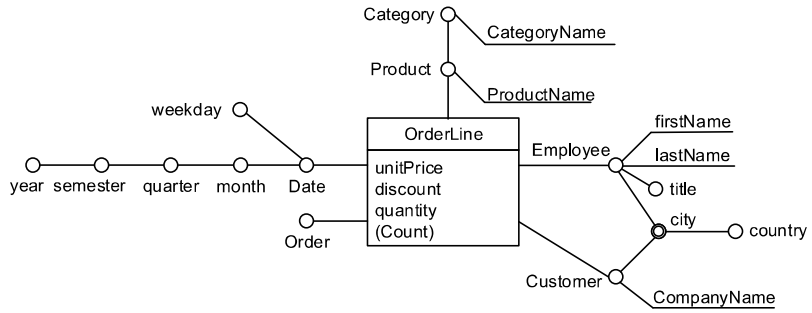
Figure 3: Relational schema of the Northwind data warehouse

```
[Quantity] [smallint]
PRIMARY KEY (OrderID,DateID,CID,EID,ProductID)
);
```

The translation from the DFM to the relational model loses some important information, including for instance the exact form of the hierarchy, what attributes together define the members of a certain level in the hierarchy, what aggregation functions we can apply, for which measures, and over which dimensions, ... Hence, before we can use an OLAP tool to deploy and browse a data cube, we need to add the necessary meta-information. Follow the step-by-step guide to start a MS Visual Studio project for defining the meta-information to be used by SSAS and processing and deploying the data cube. You will find an instantiation of a database based on this relational schema for implementing our DFM on SQL Server (database name: 2014_SimpleNW_DW).

Solution
A BIDS project defining the cube meta-data is available on course website.

2. Consider the following, simple, DFM schema dimensional:



This schema has been implemented in the relational model as follows:

```
CREATE TABLE [dimGeography](
[GID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
[city] [nvarchar](15),
[country] [nvarchar](15)
);

CREATE TABLE [dimCustomer](
[CID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
[CustomerID] [nvarchar](5),
[CompanyName] [nvarchar](40),
[GID] [int] FOREIGN KEY REFERENCES dimGeography(GID),
);

CREATE TABLE [dimDate](
[DateID] [int] NOT NULL PRIMARY KEY,
[Date] [date],
[Weekday] [varchar](50),
[day] [int],
[month] [int],
[monthname] [varchar](50),
[year] [int],
[semester] [int],
[quarter] [int]
);

CREATE TABLE [dimEmployee](
[EID] [int] IDENTITY(1,1) NOT NULL PRIMARY KEY,
[EmployeeID] [int],
[LastName] [nvarchar](20),
[FirstName] [nvarchar](10),
[Title] [nvarchar](30),
[GID] [int] FOREIGN KEY REFERENCES dimGeography(GID),
);

CREATE TABLE [dimProduct](
[ProductID] [int] NOT NULL PRIMARY KEY,
[ProductName] [nvarchar](40),
[CategoryID] [int],
[CategoryName] [nvarchar](15)
);

CREATE TABLE [factOrders](
[OrderID] [int],
[DateID] [int] FOREIGN KEY REFERENCES dimDate(DateID),
[CID] [int] FOREIGN KEY REFERENCES dimCustomer(CID),
[EID] [int] FOREIGN KEY REFERENCES dimEmployee(EID),
[ProductID] [int] FOREIGN KEY REFERENCES dimProduct(ProductID),
[UnitPrice] [money],
[Discount] [real],
```

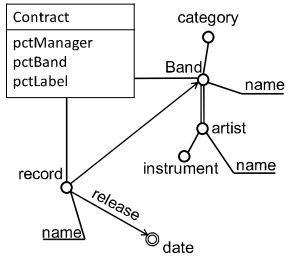
Exercises Data Warehousing
Bridge Tables and Changing Dimensions

Solutions

1. SASS (and most similar tools of other vendors as well) natively supports bridge tables. In SSAS bridge tables are considered to be measureless fact tables. Hence, when creating the cube you need to mark the bridge table as a fact table ("measurement group"). Then, in the "Dimension Usage" tab of the cube explorer you can indicate how the dimension that is connected to the fact table through the bridge, has to be used. The relationship type is "Many-to-Many" and the intermediate measure group (=fact table) will be your bridge table.

   Connect to the database "M2MDB" in BIDS. You will need all tables except Cust_Acc_Bridge. Connect the Customer dimension ("dimCustomer") to your data cube (measurement group "factTransactions").

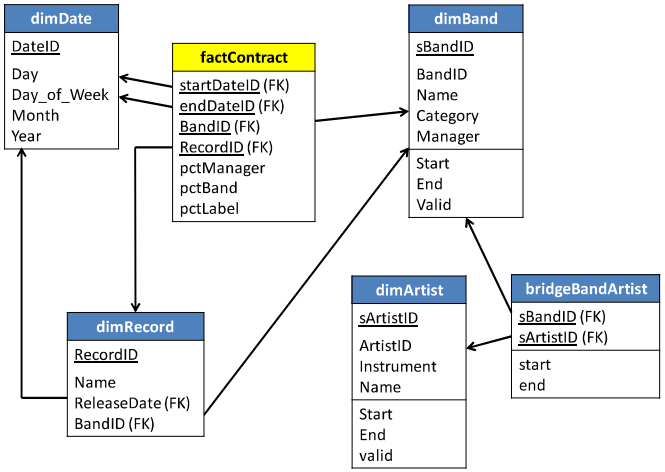
2. Consider the following DFM schema.



   Design a relational schema that implements this model. Ensure that your schema can accommodate the following changes:

   (a) Band name
   (b) Band members
   (c) Instrument

   Solution:



3. Consider the database we worked with last week, Simple_NW_DW. Suppose now that the following dimensional changes need to be accommodated for:

   (a) What changes do we need to make to the schema in order to be able to accommodate the following changes?

   - The company name and city of a customer may change.
   - The name of a product and of a category may be corrected.
   - The title of an employee may change.

   Solution: You will find the solution in Simple_NW_DW_SCD on the MSSS.

   (b) What updates to the database are needed (DELETE, INSERT, UPDATE statements) in order to encode the following changes:

   - Product name "Flotemysost" is corrected to "Flotemisost." (product ID 71)
     Solution: This is a type-1 change. Since Product is not versioned, you only have to update the existing tuple for product 71.
   - The company with ID "GREAL" changes its name back to "Great Lakes Food Market".
     Solution: This is a type-1 change. Create a new version for "GREAL" by copying all attributes from the most recent version of the tuple and changing the name. The most recent version of GREAL can be recognized by the fact that the end date equals NULL. The new version has start date the current date. Furthermore, in the newly created tuple, set the start to the current moment, and end to NULL. This is now the valid version of GREAL. For the previously most recent version of the tuple, set end to the current moment.
   - The first name of employee with ID 1 (Nancy Davolio) is misspelled. Her first name should be "Nancera" instead of "Nancy."
     Solution: This is a type-1 change. Since Employee is versioned, however, it may be the case that multiple versions have to be corrected as they all contain the same spelling mistake. Hence, for all tuples with ID 1 (OLTP key equal to 1, not the surrogate key!), make the change.
   - After Andrew Fuller resigns, Steven Buchanan becomes the new CEO.
     Solution: This is a type-2 change, since Employee is versioned. In both the most recent version of the employee record of Andrew Fuller as that of Steve Buchanan the end is set to the current moment. For Steven Buchanan a new version is made in which the position is changed and the start is the current moment and the end NULL.

4. What changes do we need to make to our data warehouse if we retrospectively learn that for one of the dimensions there was an update on one or more values of one of the tuples? Consider for instance the database of question 2. We learn now that customer "HUNGO" changed location on September 20th, 2009 (day with ID 264) and moved from Cork (GID 19) to Charleroi in Belgium (GID 18) on that day. What changes should we make to the data warehouse in order to retrospectively bring the content of the data warehouse up to date?

   Solution: TBD

5. Consider and model the following situation: we are storing all transactions of a bank. Every transaction is for an account, done by a customer, at a certain date. Furthermore we keep the amount. An account is associated to multiple customers. A bank account has attributes accountNr (=OLTP key) and a policy ("can go below zero but bounded", "can go below zero but unbounded", "cannot go below zero"). A customer is identified by his or her passport number, has a name, and is or is not a VIP customer.

   - Model this in the DFM, and translate to the relational model. Do not yet consider SCDs at this point.
   - Now consider the following changes that may take place:
     – The policy of an account may change;
     – A customer's status may change from non-VIP to VIP or vice-versa;
     – The owners of an account may change.

     How would you change your model such that these changes can be accommodated?

Exercises Data Warehousing
Indexing

Solutions

1. Consider the following relation `Cars`:

Brand	Type	Color	Risk
Opel	Corsa	Grey	Low
Opel	Corsa	Red	Medium
Peugeot	206	Black	Medium
BMW	A	Black	High

   (a) Construct a bitmap index for the attributes `Brand` and `Color` for this table.
   Solution

Brand

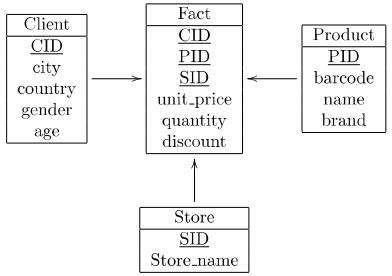
Opel	Peug.	BMW
1	0	0
1	0	0
0	1	0
0	0	1

Color

Grey	Red	Black
1	0	0
0	1	0
0	0	1
0	0	1

   (b) Indicate how these two bitmap indices can be used to answer the query: *Give the total number of red Opel cars with a medium risk score.*
   Solution: First we intersect the bitmap for `Opel` with the bitmap index for `Red` by taking the logical **AND** of the bitmaps, resulting in the bitmap: (0 1 0 0). Then, for all 1-entries, we directly access the corresponding tuple in the relation, check the condition `Risk=medium`, and if this condition is satisfied, we count the tuple. Hence, in this case, we access the second tuple only. As the tuple satisfies the condition, it is counted. The final result 1 is returned.

2. Consider the following Star schema for a ROLAP database. A fact (c, p, s, up, q, d) represents that customer c bought q units of product p in store s at unit price up and received a discount d.



   (a) Discuss the advantages and disadvantages of the following indices; for instance, describe under what circumstances and for which types of queries are these indices interesting.
      i. a bitmap join index for the attribute country from the table Client into the fact table (mapping countries to tuples in the fact table that are about this country);
      Solution: The advantage of this index is that it will speed up queries that slice on a country as it will allow to immediately select the disk blocks in which facts are stored that are needed to answer the query. One caveat, however: if there is only an index on country, it is likely that this index will not be sufficiently selective because, depending on the block size, every block may contain every country. Hence, in such a situation, unless the index can be used in combination with other bitmap or bitmap-join indices, the index will not be helpful. So, in summary, this index is probably only useful in combination with other bitmap indices.
      ii. a bitmap index on table Client for the attribute gender; and,
      Solution: Again a similar argumentation as for the previous index applies, because there are only two possible values for gender. Therefore, a bitmap index on gender will only be useful in combination with other bitmap indices on the same table, such as for instance on city.

      iii. a bitmap index for the attribute unit_price in table Fact.
      Solution: This index is clearly suboptimal as there will be many different values in unit_price. In such a situation it is always better to use for instance a btree index. Furthermore, in this case unit_price is a measure, not a dimension. Hence it will be unlikely that there will be many queries making selections on the basis of unit_price, which renders an index on this attribute useless.

   (b) Consider the following query:

```
SELECT Client.Gender, SUM(Fact.quantity)
FROM Fact, Client
WHERE Fact.CID=Client.CID AND
      (Client.country = "The Netherlands"
       OR Client.country = "Belgium")
group by Client.Gender
```

   Select one of the indices from (a) and explain how this index may help to answer this query efficiently.
   Solution: The only candidate is the first bitmap-join index. This index allows to select only those facts that are about Belgium and The Netherlands. For these facts, however, we will have to lookup the associated clients in the client dimension table because we need to know the gender in order to get access to the gender. Hence, it is unlikely that any of the indices above will actually help, unless the selection (`Client.country = "The Netherlands" OR Client.country = "Belgium"`) is very selective and there is a (BTree) index on CID in the client table that allows to retrieve tuples in the Client table by CID instead of by a full table scan.

3. Deduplication is an important task in data cleaning. To do deduplication it is useful to have a measure of distance between string values such as names, addresses, phone numbers, etc. The edit distance is one such distance measure. Compute the edit distance between "Brussel" and "Bruges".
   Solution:

		B	**R**	**U**	**G**	**E**	**S**
	0	1	2	3	4	5	6
B	1	**0**	1	2	3	4	5
R	2	1	**0**	1	2	3	4
U	3	2	1	**0**	1	2	3
S	4	3	2	1	**1**	2	3
S	5	4	3	2	**2**	2	3
E	6	5	4	3	3	**2**	3
L	7	6	5	4	4	4	**3**

   The edit distance is **3** as indicated in the botton right corner. By following the path in **bold** from the top left corner to the right bottom corner we can identify the required operations. The first three **0** mean that no operation is required since the 3 first letters are equal. The next **1** means that the **S** was replaced by a **G**. The first **2** means that the second **S** was deleted. The second **2** means that no operation was needed since the two characters are equal to **E**. Finally, the last **3** means that the **L** was replaced by an **S**. Notice that the path in bold is just **one** among multiple paths of length **3**.

4. Consider a cube with three dimensions A, B, and C, and one measure M. Suppose that the complete cube needs to be materialized for the aggregation function M. That is, given the base table B(IDA, IDB, IDC, M), we need to compute:

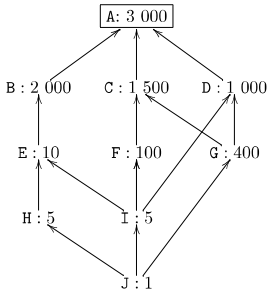
```
SELECT IDA, IDB, IDC, sum(M)
FROM B
GROUP BY CUBE(IDA, IDB, IDC)
```

   Show how you could apply the pipe-sort algorithm to compute the cube. Assume that sorting a relation X takes SORT(X) time, and scanning the table takes FTS(X) (FTS stands for "full table scan"). Express the time needed by the query plan developed by the pipe-sort algorithm with the time needed by a brute-force solution that computes the aggregations for all grouping sets separately. What is the gain? (Express costs and gain in terms of SORT(X) and FTS(X) for X any of the aggregation tables constructed along the way)

Exercises Data Warehousing
View Materialization

Solutions

1. An important technique to speed up analytical queries is by pre-computing and materializing aggregations. Consider the following lattice of views that can be requested by the user, along with the number of rows in each view.



A is the view representing the base relation. The edges indicate the relation "can be computed from."

(a) Suppose that only the top-view A has been materialized. Select three additional views from the views B, C, D, E, F, G, H, I, and J to materialize. Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996)

Solution: Initially we start with the set of materialized views $S = \{A\}$. This top-level view is always in the set of materialized views as none of the other views can be used to generate it. The benefits of the different views are:

B	$5 \times (3\,000 - 2\,000) = 5\,000$
C	$5 \times (3\,000 - 1\,500) = 7\,500$
D	$4 \times (3\,000 - 1\,000) = 8\,000$
E	$\mathbf{4 \times (3\,000 - 10) = 11\,960}$
F	$3 \times (3\,000 - 100) = 8\,700$
G	$2 \times (3\,000 - 400) = 5\,200$
H	$2 \times (3\,000 - 5) = 5\,990$
I	$2 \times (3\,000 - 5) = 5\,990$
J	$1 \times (3\,000 - 1) = 2\,999$

As E gives the highest benefit, this view is selected.

For selecting the second view, we calculate the benefits w.r.t. the set of already selected views $\{A, E\}$:

B	$1 \times (3\,000 - 2\,000) = 1\,000$
C	$\mathbf{3 \times (3\,000 - 1\,500) = 4\,500}$
D	$2 \times (3\,000 - 1\,000) = 4\,000$
F	$1 \times (3\,000 - 100) = 2\,900$
G	$1 \times (3\,000 - 400) = 2\,600$
H	$2 \times (10 - 5) = 10$
I	$2 \times (10 - 5) = 10$
J	$1 \times (10 - 1) = 9$

As now C gives the highest benefit, this view is selected in the second step.

For selecting the third view, we calculate the benefits w.r.t. the set of already selected views $\{A, C, E\}$:

B	$1 \times (3\,000 - 2\,000) = 1\,000$
D	$\mathbf{1 \times (3\,000 - 1\,000) + 1 \times (1\,500 - 1\,000) = 2\,500}$
F	$1 \times (1\,500 - 100) = 1\,400$
G	$1 \times (1\,500 - 400) = 1\,100$
H	$2 \times (10 - 5) = 10$
I	$2 \times (10 - 5) = 10$
J	$1 \times (10 - 1) = 9$

Solution: The pipe-sort algorithm is based on the idea that when a relation is sorted, for example, in attributes ABC it is also sorted on attributes AB and A. Therefore, with a single scan it is possible to compute the aggregation on ABC, AB, A and {} in a pipe-line fashion.

For the GROUP BY CUBE we need to compute the aggregation for all $2^n = 8$ subsets of ABC. By sorting on ABC we have can compute the aggregation of 4 of these subsets. By sorting on BC we can compute the aggregation on BC and B. Finally, we need to compute the aggregation on both AC and C. The cost for computing the cube would be

SORT(ABC) + FTS(ABC) → compute ABC, AB, A, {}
SORT(BCA) + FTS(BCA) → compute BC, B
SORT(CAB) + FTS(CAB) → compute CA, C
SORT(CA) → obtain AC

Compare this with respect to the brute-force solution SORT(X) + FTS(X) for all eight subsets of ABC.

5. Suppose that a cube Sales(A, B, C, D, Amount) has to be fully materialized. The cube contains 64 tuples. Sorting takes the typical $n \log(n)$ time. Every GROUP BY with k attributes has 2^k tuples.

(a) Compute the cube using the PipeSort algorithm.

Solution: A possible evaluation plan to fully materialize the cube is given in Fig. 1.

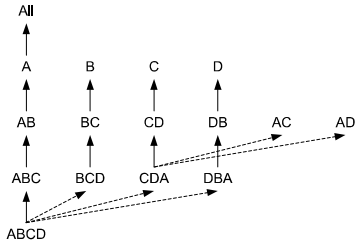


Figure 1: Evaluation plan to fully materialize the cube Sales(A, B, C, D, Amount)

(b) Compute the gain of applying the PipeSort compared to the cost of computing all the views from scratch.

Solution: There are 16 views. Since the cube has 64 tuples, suppose that the views with 3 attributes (such as BCD) have 32 tuples and the views with 2 attributes (such as AC) have 16 tuples. Suppose also that scanning a table of n tuples costs n time units, and that sorting a table of n tuples costs $n \log(n)$ time units.

The cost of applying the PipeSort algorithm is as follows (see Fig. 1). The first scan of 64 tuples costs 64 units. Then, the following two operations are performed three times: a sort of the table for computing a view with three attributes, and a pipeline for computing all views derived for that. The cost of these operations is $3 \times (32 \log(32) + 32)$. Finally, the following two operations are performed two times: a sort of the table for computing a view with two attributes, and a pipeline for computing all views derived for that. The cost of these operations is $2 \times (16 \log(16) + 16)$. Thus, the total cost of the PipeSort algorithm is

$$64 + 3 \times (32 \log(32) + 32) + 2 \times (16 \log(16) + 16) = 64 + 576 + 152 = 792$$

On the other hand, the cost of computing each of the 16 views independently would imply a sort operation and scanning of the table. Thus, the total cost would be

$$4 \times (32 \log(32) + 32) + 6 \times (16 \log(16) + 16) + 4 \times (8 \log(8) + 8) + 8 = 768 + 456 + 128 + 8 = 1360$$

Therefore, in this example, the cost of the view computation with the PipeSort algorithm will be 58% of the cost of computing the views independently.

(c) What is the total benefit of materializing those two views?

Solution: The total benefit is 635 400 + 115 920 = 751 320.

3. Imagine a National Bureau of Statistics holding demographic data of the inhabitants of its country. For each person the following attributes are recorded: *City*, *Age*, *Ethnicity*, and *Income*. The bureau wants to make aggregations of this data available for other governmental agencies as well. The agencies can query the data by selecting any subset of the first three attributes. For any combination of the selected attributes the average income of people with this particular combination will be returned. For example, if an agency selects *City* and *Age*, it will get for every city and age, the average income of the people of that age in that city.

   For reasons of simplicity we assume that for every city, age, and ethnicity there is at least one person that has this combination. There are 400 cities, 100 age values, and 10 ethnicity types in the database. It is decided that the data will be stored in a database system, but for performance reasons some query answers will be materialized in advance. Every time new data arrives these materialized views are updated.

   (a) Suppose 2 views can be materialized. Which views should be materialized? Explain the reasoning behind your answer.

   (b) After a few months the bureau realizes that the system's performance is suboptimal. As a first step towards solving this problem, an analysis is made of what is the frequency of the different queries. It turns out that not all 8 combinations of the attributes *City*, *Age*, and *Ethnicity* are equally likely to be requested. The following table summarizes the results of the analysis (every row represents how often one selection of attributes is being requested. The symbol $\surd$ in the column for attribute A in a row indicates that A is in the selection represented by that row):

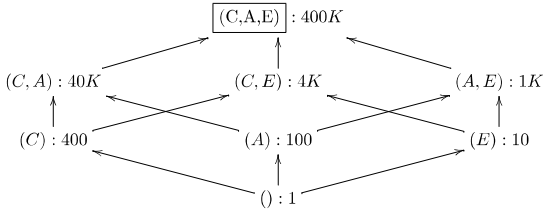
City	*Age*	*Ethnicity*	*Frequency*
$\surd$	$\surd$	$\surd$	5%
$\surd$	$\surd$		20%
$\surd$		$\surd$	30%
$\surd$			30%
	$\surd$	$\surd$	5%
	$\surd$		5%
		$\surd$	5%
			0%

   Hence, for example, it turns out that the average income over all people (no attributes selected) is never requested and that, e.g., the average income per city and the average income per city and ethnicity type are disproportionately frequent (together they form 60% of the requests!).

   Should the views to materialize be changed based on this information? If so, which views should be materialized instead? Explain your answer.

   (c) What additional benefit can be expected if it is decided to materialize also a third view? Use the settings described in (b).

Solution: The partial order between the queries looks as follows:



(a) Clearly materializing (A, E) gives the highest benefit $(4 \times 399K)$. Once (A, E) is materialized, the largest benefit is given by materializing (C, E); this gives an additional benefit of $2 \times 396K$.

As now D gives the highest benefit, this view is selected in the third step. Hence, the three selected views are: C, D, E.

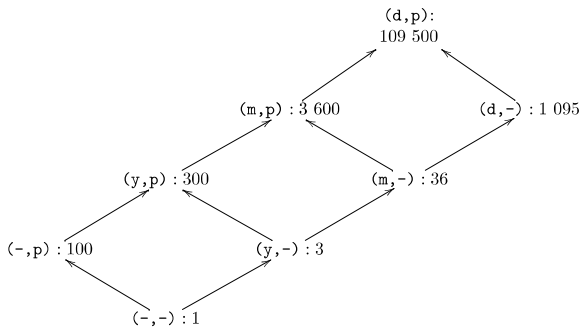
(b) What benefit gives the additional materialization of these three views under the cost model introduced by these authors?

Solution: The total benefit is: 11 960+4 500+2 500 = 18 960

2. Consider a data cube with dimensional attributes `Product` and `Date` and measure `Total Sales`. The following hierarchy is present on the dimension `Date` (No hierarchy on `Product`):

(a) Suppose that the cube is dense; i.e., for every product p and date d there is a tuple (p, d, t) in the base relation with $t > 0$. Furthermore, there are 100 products and 3 years of data (1095 days; no leap year, 157 weeks) in the database. Determine the sizes of the different views.

Solution: Below the partial order between the query is drawn. The number behind the colon indicates the size of the particular view.



(b) Apply the greedy algorithm to select 2 views to materialize.

Solution: Initially we start with the set of materialized views $S = \{(d, p)\}$. This top-level view is always in the set of materialized views as none of the other views can be used to generate it. The benefits of the different views are:

(m,p)	$\mathbf{6 \times (109\,500 - 3\,600) = 635\,400}$
(d,-)	$4 \times (109\,500 - 1\,095) = 433\,620$
(y,p)	$4 \times (109\,500 - 300) = 436\,800$
(m,-)	$3 \times (109\,500 - 36) = 328\,392$
(-,p)	$2 \times (109\,500 - 100) = 218\,800$
(y,-)	$2 \times (109\,500 - 3) = 218\,994$
(-,-)	$1 \times (109\,500 - 1) = 109\,499$

As (m, p) gives the highest benefit, this view is selected.

For selecting the second view, we calculate the benefits w.r.t. the set of already selected views $\{(d, p), (m, p)\}$:

(d,-)	$\mathbf{1 \times (109\,500 - 1\,095) + 3 \times (3\,600 - 1\,095) = 115\,920}$
(y,p)	$4 \times (3\,600 - 300) = 13\,200$
(m,-)	$3 \times (3\,600 - 36) = 10\,692$
(-,p)	$2 \times (3\,600 - 100) = 7\,000$
(y,-)	$2 \times (3\,600 - 3) = 7\,194$
(-,-)	$1 \times (3\,600 - 1) = 3\,599$

As $(d, -)$ gives the highest benefit, this view is selected.

The two selected views next to the obligatory top view (d, p) are hence: (m, p) and $(d, -)$.

Solutions Data Warehousing Exam January 2013

1. (3p) Carefully read the description stating the requirements for a data warehouse for a bank:

Loan Datawarehouse

A bank wants to build a data warehouse for storing and analyzing data about all loans issued by them.

Every loan has one or more borrowers, a starting date, a type (e.g., fixed rate or one of different types of variable rate), the branch of the bank where the loan was issued, the interest rate at the start of the loan, and the amount. Throughout the duration of the loan the interest rate may change. For every loan the purpose of the loan is recorded; e.g., to buy a car, a house, a personal loan, ... When a borrower applies for the loan, different discounts on the interest rate may be awarded; e.g., fidelity discount, discount because the borrower also bought some additional insurances, VIP discount, etc. For one loan, multiple discounts may apply. The set of discount types is fixed over time, although the magnitude of the discount may change over time due to strategic decisions. The amount of discount is independent of the branch. Every discount that has been awarded needs to be stored. When the loan ends, this is stored as well, together with an indication if the loan was fully repaid or the borrower defaulted.

For the borrowers, their date of birth, family status, monthly income, number of children and address is stored.

The following questions are prototypical for the type of query analysts want to answer based on the data warehouse:

- Give the average interest rate before discount at the start of the loan, per loan type and branch.
- For all branches, give the minimum, maximum and average interest rate per loan type and purpose.
- Give the number of loans per branch and per amount category. The amount category depends on predefined thresholds; amounts are divided into the following classes: `very high`, `high`, `medium`, `low`, and `very low`.
- Give the percentage of defaulted loans per year and per city of the branch where the loan was issued.

(a) Make a dimensional model for the data warehouse. Indicate which cube(s) are needed, what are the dimensions, measures, hierarchies, etc.

(b) Describe the tables for storing the data in a relational database; that is, in a ROLAP solution; make sure that your tables can accommodate changes in the data as much as possible.

Solution: *There was not a single correct answer; different choices may lead to different models. The level of detail given in this explanation was not necessarily expected in your answer.*

(b) When we take the frequencies of the queries into account, the benefits in the first step become:

(C, A)	$55\% \times 360K$
$\boxed{(C, E)}$	$\boxed{65\% \times 396K}$
(A, E)	$15\% \times 399K$
(C)	$30\% \times 399\,600$
(A)	$5\% \times 399\,900$
(E)	$5\% \times 399\,990$
$()$	$0\% \times 399\,999$

Clearly, this time (C, E) is selected as the best view to materialize. In the second step, the benefits are:

$\boxed{(C, A)}$	$\boxed{25\% \times 360K}$
(A, E)	$10\% \times 399K + 5\% \times 3K$
(C)	$30\% \times 3\,600$
(A)	$5\% \times 399\,900$
(E)	$5\% \times 3\,990$
$()$	$0\% \times 3\,999$

Hence, (C, A) is selected as the second view to materialize.

(c) For materializing a third view, the benefits are:

$\boxed{(A, E)}$	$\boxed{5\% \times 399K + 5\% \times 39K + 5\% \times 3K}$
(C)	$30\% \times 3\,600$
(A)	$5\% \times 39\,900$
(E)	$5\% \times 3\,990$
$()$	$0\% \times 3\,999$

Hence the additional benefit of materializing a third view would be: $5\% \times 399K + 5\% \times 39K + 5\% \times 3K$.

however that there could be other reasons to go for a snowflake (e.g., enforcing consistency).

The relations should be such that they allow storing the dimensional model, as well as to accomodate changes to the dimensions; e.g., customers may move, discounts can change, etc. It is important to add surrogate keys for the changing dimensions! The OLTP-key will not be unique in the data warehouse because a change in the OLTP database affecting one of the dimensions will result in a new tuple in the dimension table with the same OLTP-key.

The resulting table for the borrower dimension is as follows:

Borrower
Borrower_ID
Borrower_OLTPKey
Address
Income
Gender
No_Children
...

We keep the OLTP key if borrower because otherwise it becomes impossible to see which tuples correspond to the same borrower. To record the exact times of the changes, we could timestamp the tuples. In that case attributes to capture the validity period and maybe a flag indication if the tuple represents the most recent version should be added as well.

In case of frequent changes to some of the attributes of borrower (e.g., Income), we may opt to split-off a mini-dimension containing the profile of a borrower.

As a loan can have more than one borrower, we could decide to define a table borrower-group, grouping the borrowers that jointly have a loan. This solution particularly makes sense if the groups themselves have meaning. In this case it is likely that the groups have meaning; e.g., a married couple, or business associates.

This results in the following table:

Borrower_Group
Borrower_Group_ID
Borrower_ID

An alternative is to make a bridge-table between the fact table and the borrower table. In that case it could be a good idea to add a dedicated key (e.g., loan number) to the fact table to avoid that the bridge table has to contain the complete composite key of the fact table. Make sure, however, that in that case the cluster index for the fact table should not be on this key.

Both date dimensions are stored into the same table:

Date
Date_ID
Day
Week
Month
Quarter
...

As the discount types are fixed, we opt to combine all of them together into a junk-

A first important observation is what wil be the subjects around which we will be building our data warehouse. In this case the most natural choice is "loan." Another option could have been "loan event"; e.g., change in interest rate. This other option, however, has a couple of disadvantages: given the prototypical queries, loan is a more natural choice. Also, it is hard to define meaningful ways to aggregate measures such as amount over these events.

Hence, every fact will correspond to one particular loan. In principle a data warehouse is read-only; once a fact is entered, it remains the same (unless an error in the database is detected which will be propagated through a type-1 update). Notice that this also holds for type-2 and type-3 updates in the dimensions; these are such that the customer address associated with facts that are already in the database does not change. Following this line of reasoning, when the loan is entered, its data will not change anymore. Therefore, we assume that a new fact will be entered every time a loan ends. Data for ongoing loans will be entered in different tables; a fact here could, e.g., be the downpayements for the loan, or the sale of a loan.

Dimensional model
The dimensions and hierarchies for loan are as follows:

- The set of borrowers; hierarchy: set of cities of borrowers; income class of the borrowers, ...
- Start date and end date of the loan; hierarchy: week, month, quarter, semester, year, ...
- Status of the loan (repaid, defaulted);
- Discounts that have been applied to the loan;
- The loan amount category;
- The loan type;
- The branch; hierarchy: city, region, country, continent where the branch is located.
- The purpose of the loan: maybe the purposes can be grouped into categories. In that case, the category would be a level in the hierarchy.

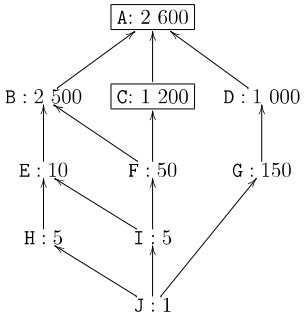
The measures for the loan are:

- the loan amount;
- a number of aggregations of the loan interest, including the starting rate before and after discount. On top of that other useful aggregations could be stored such as the maximal, minimal and average interest before/after discount;
- Another useful measures could be the total discount.

Adding multiple measures to the fact table could make the table grow very big. In this specific situation, however, it is unlikely that it will lead to a significant footprint; after all the number of loans will not increase with a million per day, in contrast to, e.g., the registration of all customer money transactions.

Relational Model.
We could opt for a star schema or a snowflake schema. As space is unlikely to become an issue for this particular case, we have chosen for a star schema. Notice

2. (3p) Consider the following lattice of views along with a representation of the number of rows in each view where A is the base cuboid:



view	frequency
A	0%
B	5%
C	10%
D	0%
E	5%
F	20%
G	10%
H	10%
I	40%
J	0%

The table on the right expresses how often the different views are requested.

(a) Suppose that the top-view A and the view C have already been materialized. Select two additional views from the views B, D, E, F, G, H, I, J to materialize. Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996)

(b) What benefit gives the additional materialization of these two views?

Solution: In the given situation, the benifits of materializing the different views that have not been materialized yet, are as follows:

$$
\begin{array}{lcll}
B & : & (5\% + 5\% + 10\%) \times (2600 - 2500) & = & 20 \\
D & : & 10\% \times (2600 - 1000) & = & 160 \\
\boxed{\mathbf{E}} & \boxed{:} & \boxed{\mathbf{(5\% + 10\%) \times (2600 - 10) + 40\% \times (1200 - 10)}} & \boxed{=} & \boxed{\mathbf{864.5}} \\
F & : & (20\% + 40\%) \times (1200 - 50) & = & 690 \\
G & : & 10\% \times (2600 - 150) & = & 245 \\
H & : & 10\% \times (2600 - 5) & = & 259.5 \\
I & : & 40\% \times (1200 - 5) & = & 478 \\
J & : & 0\% \times (1200 - 1) & = & 0
\end{array}
$$

So, the first extra view to materialize is E. After materializing E, the new benifits of the remaining views become:

$$
\begin{array}{lcll}
B & : & 5\% \times (2600 - 2500) & = & 5 \\
D & : & 10\% \times (2600 - 1000) & = & 160 \\
F & : & (20\%) \times (1200 - 50) & = & 230 \\
\boxed{\mathbf{G}} & \boxed{:} & \boxed{\mathbf{10\% \times (2600 - 150)}} & \boxed{=} & \boxed{\mathbf{245}} \\
H & : & 10\% \times (10 - 5) & = & 0.5 \\
I & : & 40\% \times (10 - 5) & = & 2 \\
J & : & 0\% \times (10 - 1) & = & 0
\end{array}
$$

So, the additional benefit of materializing E and G is: 864.5 + 245 = 1109.5.

dimension:

Discount_Group
Discount_Group_ID
D1_bonification
D2_bonification
...

When the rates change, new groups are added.

We consider Status to be a degenerate dimension; i.e., we do not add a dimension table as the fact table already contains the one and only attribute in this dimension. Similarly for Amout_Cat.

Furthermore, for Branch, Purpose and Type also tables are added.

The fact table is as follows:

Loan
Borrower_Group_ID
Start_Date
End_Date
Discount_Group_ID
Status
Amount_Cat
Branch_ID
Purpose_ID
Type_ID
Amount
Start_Rate_BD
AVG_intrest_BD
MIN_intrest_BD
MAX_intrest_BD
Start_Rate_Discount
...

The explanation given here is very extensive to more clearly motivate the models. This level of detail and complexity was not expected on the exam. Many other good solutions were given.

Common mistakes on the exam: no surrogate keys—would create problems when updating; sometimes a surrogate key was added to the fact table instead of to the dimension tables
relational schema is invalid; e.g., keys to fact table added into the dimensions instead of the other way around;
No keys were given
Changes cannot be appropriately captured;
Measures that become dimensions and vice versa;
For every example query a cube was defined; this is clearly violating the assumption that the schema should be such that it allows for efficiently answering ad-hoc analysis queries—dealing with ad-hoc queries implies we cannot make a differnet cube for every query;
Inability to deal with multiple discounts or multiple borrowers

The following indices hardly make sense:

- On Source_Number, Dest_Number in Call or on Number in Owner or on ID, Name in Person because there are too many different attribute values. Other types of indices such as the BTree will be much more useful for these cases.

The following bitmap indices do make sense, but cannot help in optimizing the query:

- On Call_duration in table Call under the condition that Call_duration is discretized or does not contain too many different values; e.g., if duration is recorded in minutes.
- On Age in table Person. For the numeric attriubtes, however, other indexing techniques that allow for range queries may be more appropriate.

The following bitmap-join indices are helpful for the query:

- On Person.City into table Owner; join on Owner.Owner_ID=Person.ID
- On Person.Gender into table Owner; join on Owner.Owner_ID=Person.ID
- On Person.City into table Call; join on Call.Source_Number=Owner.Number and Owner.Owner_ID=Person.ID;
- On Person.Gender into table Call; join on Call.Dest_Number=Owner.Number and Owner.Owner_ID=Person.ID.

Bitmap-join indices on ID and Name are not useful since the domains are too large; Bitmap-join index on Age cannot be used in the query; on Time_of_day is not preferable since the relation from call to owner and (indirectly) to person is many-one—most bitmaps will only contain 1's. Bitmap-join index on Gender is not selective enough; only useful in combination with other bitmap-join indices.

The query is helped by, e.g., the bitmap-join index on Person.City into table Call: directly find the Calls for numbers owned by persons from Brussels, skipping two joins. Using the index to find the tuples corresponding to callers not in Antwerp is likely to be far less useful because this selection criterium is insufficiently selective.

Common errors: bitmap index is given instead of bitmap-join index;
bitmap join index maps to rows of the joined table (must be: index maps values v in an attribute A of one table T to tuples in another table S that join with a tuple t of the first table having $t.A = v$)
bitmap index is used to combine the condition $City = Brussels$ and $City \neq Antwerp$, which does not makes sense since it concerns two different tuples representing different persons.

3. (2p) Consider the following example database and query over the database:

Call

Source_number	Dest_number	Time_of_day	Call_duration
0497456345	0936596743	"1pm-2pm"	30
0497456345	0936596743	"2pm-3pm"	10
0456972890	0936596743	"3am-4am"	120
0456972890	0456972890	"4p-5pm"	2
...	...	...	...

Owner

Number	Owner_ID
0497456345	001
0765382482	001
0456972890	002
0987653197	004
0936596743	005
...	...

Person

ID	Name	Age	Gender	City
001	Pete	32	m	Hasselt
002	Mary	36	f	Antwerp
003	Jean	57	f	Mechelen
004	Jeff	78	m	Brussels
...	...	...	...	...

```
SELECT P1.City,COUNT(*),SUM(C.duration)
FROM Owner S, Call C, Owner D, Person P1, Person P2
WHERE C.Source_number = S.Number
      AND C.Dest_Number = D.Number
      AND (C.Time_of_day = "1pm-2pm" OR C.Time_of_Day = "2pm-3pm")
      AND P1.ID=S.Owner_ID
      AND P2.ID=D.Owner_ID
      AND P2.City="Brussels"
      AND P1.City<>"Antwerp"
      AND P1.Gender="f"
GROUP BY P1.City
```

(a) Give two examples: one of a bitmap and one of a bitmap-join index that would speed-up the execution of this query.

(b) Explain for one of them how it speeds up the computation of the query.

Solution: The following bitmap indices are helpful for the query:

- On Time_of_day in table Call
- On City in table Person
- On Gender in table Person, although it is not selective enough. This bitmap will only be useful if ti can be used in combination with other bitmap indices.

BONUS (+1) Assume a large graph is stored on disk as a binary relation $R(A, B)$. Describe an efficient method to approximate how often edges in the graph are symmetric; an edge (u, v) is called symmetric if the edge (v, u) exists as well. In other words, we want to compute: $|\{(u, v) \in R \mid (v, u) \in R\}|$.

Solution: The solution relies on the approximation of the Jaccard-index (and hence indirectly for the size of the intersection of two sets) based upon hashing that we have seen in class as one of the main building blocks of the semi-streaming algorithm for counting the number of triangles in a disk-resident graph.

In one scan over the database, construct two "signatures" per node v; one for its forward neighbors $F(v)$, and one for its backward neighbors $B(v)$. This can be stored in memory. [If the signatures are too big to fit into the memory we can process the hash functions one by one, similarly as for counting the triangles. We omit this complexity here.]

Then go over all nodes v, and compute $\sum_{v \in V} |F(v) \cap B(v)|$. The size of the intersection can be estimated directly from the signatures.

Common errors: solutions based on first counting triangles; since there is no relation to the number of symmetric edges and the number of triangles this approach is automatically doomed.
Using sketching methods to count the number of edges (u, v) and then check for every edge (u, v) if (v, u) has a non-zero frequency. This won't work as the streaming methods for counting frequencies are approximate and very inaccurate for low frequencies. The frequencies that need to be estimated in this approach are 0 or 1.

4. (1p) Compute the edit distance between the following two strings: "Belgium" and "Bulgeria".

   Solution: Compute the edit distance using dynamic programming:

		B	e	l	g	i	u	m
	0	1	2	3	4	5	6	7
B	1	0	1	2	3	4	5	6
u	2	1	1	2	3	4	4	5
l	3	2	2	1	2	3	4	5
g	4	3	3	2	1	2	3	4
e	5	4	3	3	2	2	3	4
r	6	5	4	4	3	3	3	4
i	7	6	5	5	4	3	4	4
a	8	7	6	6	5	4	4	5

   Distance equals 5.

5. (1p) Explain the concept of "Data temperature" and "warm data" that was introduced in the Teradata presentation (slide 33 and following), and how it is exploited by Teradata to improve performance.

   Solution: data temerature refers to how often data is accessed. Hot data is accessed most, warm data less, cold data the least. The warmer the data, the faster the disk it will be stored on. So, the most accessed data will be stored on the fastest (part of the) disk. See also `http://d289lrf5tw1zls.cloudfront.net/media/whitepapers/The_Data_Temperature_Spectrum.pdf` for detailed information.

Solution Exam Data Warehouses January 2014

1. (4p) Carefully read the description stating the requirements for a data warehouse for a record label:
   A record label wants to keep track of all contracts they have with bands, records they are producing and the sales of these records. Currently they are only keeping data of their ongoing contracts; no historical information is kept. Therefore it is decided to construct a data warehouse for collecting and storing historical information. With the data warehouse the company wants to analyze its sales. Based on conversations with the managers of the company, you were able to compile the following description of the available data.

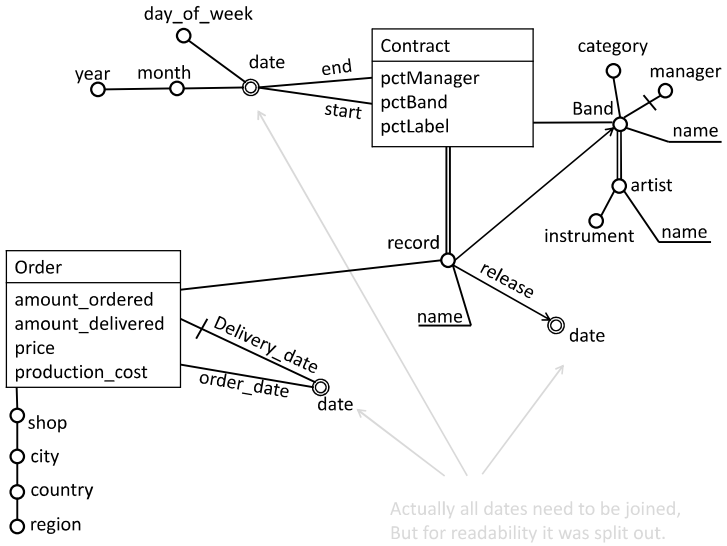
   The record label has several bands under contract. Each band has a name, a main category of music it plays, and one or more artists. A band may have a manager, but does not have to. Bands without a manager must have one of the artists as their main representative. An artist has one main instrument, which can also be his/her voice. Over time, bands can split and attract or replace group members. Bands make records which are physically produced and distributed by the record label. Records, including those that are not yet finished, have a title and are identified by a special international code. For every record the price to produce it and its release date are stored. The record label has contracts with the bands. For every contract start and end data are registered as well as for which records it holds. what percentage of sales goes to the manager of the group (if present). how much to the group, and how much to the record label. A contract can apply to multiple records, but every record falls under exactly one contract. Shops can order records at the record label. The date, amount, and agreed price of these orders are recorded, as well as the number of records that were sent to the shop in the end (sometimes demand outweighs supply) and the date on which the order was fulfilled. For every shop, the record label has information about the spatial location, including: city, country, and region of the shop.

   Some examples of the types of analyses the record label wants to perform based on the data warehouse are the following:

   - *Which record/group/artist has given the record label the highest/lowest gain/return on investment per week/month/year.*
   - *What seller/city/country has the highest number of demands that could not be met?*
   - *Which record/group/artist has the highest number of demands?*

   (a) Draw a dimensional fact model (DFM) for the data warehouse. Notice that there is not a single correct solution and that multiple facts may be needed. For every fact in your model explain succinctly what it represents. Not all information in the description is necessarily relevant for the dimensional fact model. Your model should represent the description as faithfully as possible.

   (b) Translate the DFM you constructed in (a) to an appropriate relational model. Clearly indicate primary and foreign keys in your tables. *In order to gain full points for (b) the model given in your answer to (a) must be reasonable. For instance: not answering (a) and giving an empty relational schema as a (correct) answer for (b) will obviously not gain you any points.*

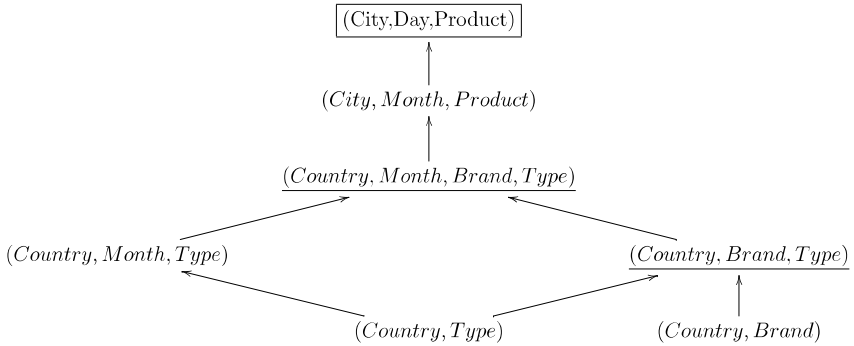


Common mistakes: dimensional attributes that are incorrectly placed into a hierarchy (for instance, roll-up from band to record); many-to-many relations modeled as one-to-many; models with only one fact usually did not work: contracts are not a property of sales, not vice versa. These are two different subjects in the data warehouse, hence requiring two facts. A third fact, for record, or the production of a record could be considered.

A possible relational schema:

- dimDate(DID, day, day_of_week, month, year)
- dimBand(BID, bandName, manager)
- dimArtist(AID, instrument, artistName)
- bridgeBandArtist(BID, AID)
- dimRecord(SpIntCode, releaseDID, recordName, BID)
- recordGroup(RGID, SpIntCode)
- factContract(startDID, endDID, RGID, BID, pctMan, pctBand, pctLabel)
- dimShop(SID, shopName, city, country, region)
- factOrder(deliveryDID, orderDID, SID, SpIntCode, amount_ordered, amount_delivered, price, production_cost)

Solution: As you will quickly notice if you try, constructing the complete lattice is very cumbersome and time-consuming; it is too large to completely draw it and to compute the benefit of every single view in it. Fortunately, we do not need the complete lattice. Since that there are only 4 views that are ever asked, we can restrict the views we need to consider. For instance, we do not need to consider the view (Country,Product); indeed: if we would materialize (Country, Product), of the 4 views only (Country,Brand) and (Country,Type) benefit. But, these 2 views would benefit even more if we materialize the smaller (Country,Brand,Type). Given the relation between the 4 views, it suffices to consider the following additional views: (Country,Brand,Type), and (Country,Month,Brand,Type). (*Convince yourself by trying out some other views; always one of the following six views will be even better.*) This gives us the following roll-up lattice:



Of these 6 views we now compute the benefit of materializing them:

View	Benefit
(City,Month,Product)	$100\% \times (500M - 18M)$
(Country,Month,Brand,Type)	$85\% \times (500M - 720K)$
(Country,Month,Type)	$45\% \times (500M - 7.2K)$
(Country,Brand,Type)	$60\% \times (500M - 20K)$
(Country,Type)	$20\% \times (500M - 200)$
(Country,Brand)	$40\% \times (500M - 1K)$

(City,Month,Product) gives the highest benefit, so this view is selected as the first to materialize. Then the benefits are computed again:

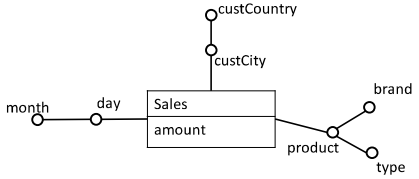
View	Benefit
(Country,Month,Brand,Type)	$85\% \times (18M - 720K)$
(Country,Month,Type)	$45\% \times (18M - 7.2K)$
(Country,Brand,Type)	$60\% \times (18M - 20K)$
(Country,Type)	$20\% \times (18M - 200)$
(Country,Brand)	$40\% \times (18M - 1K)$

This time (Country,Month,Brand,Type) gives the highest benefit and is selected.

The total benefit is: $(500M - 18M) + 85\% \times (18M - 720K)$. The relative benefit is hence:

$$\frac{(500M - 18M) + 85\% \times (18M - 720K)}{500M} \approx 0.99\%$$

2. (2.5p) Consider a data cube *Sales* with three dimensions *Customer*, *Date*, and *Product*. Dimension *Customer* has a hierarchy with levels *City* and *Country*. *City* is the lowest level of granularity in this dimension. Dimension *Date* has levels *Day* and *Month*. Dimension *Product* is stored at *product*-level and can be rolled up to level *Type* and level *Brand*. There is only one measure *Amount*. The DFM for the cube is hence as follows:



The users of the reports that are based on this data cube are complaining about degrading performance. So, you analyze the usage statistics in order to find out which queries have been asked how many times. All queries are of the form:

```
SELECT A1, ...,Ak, SUM(Amount)
FROM (Sales joined with the dimension tables)
GROUP BY A1, ..., Ak
```

`A1`, ..., `Ak` are dimensional attributes of the cube. We will represent every query by its set of grouping attributes. For instance: (Day, Brand) represents the query

```
SELECT Day, Brand, SUM(Amount)
FROM (Sales joined with the dimension tables)
GROUP BY Day, Brand
```

The cube is dense. That is: for every combination of a city c, date d, and product p, there is a tuple (c, d, p, a) in the base table with a non-zero amount a. There are 10 000 products of 100 brands and 20 product types, 50 cities from 10 countries, there are 1000 days and 36 months. Hence, in total there are $50 \times 1\,000 \times 10\,000 = 500M$ tuples in the base table. Furthermore, for every brand there is at least one product of each type. The following table summarizes the results.

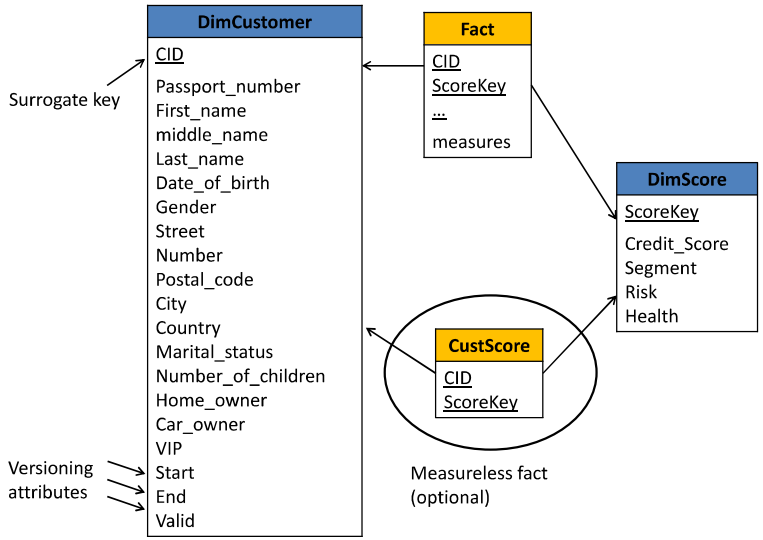
Query	Frequency of query	Output size
(City,Month,Product)	15%	18M
(Country,Brand)	40%	1K
(Country,Type)	20%	200
(Country,Month,Type)	25%	7.2K

No other queries were executed on the data warehouse.

(a) Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996) *(this is the greedy algorithm we saw in class)* to select two views to materialize.

(b) What benefit (in %) do you expect from materialization these two views?

fact-table with dimensions customer and the newly created mini-dimension. A fact in this table would then represent a change in the customers credit score, market segment, risk or health categorization. This complete solution is depicted below:



(b) Describe the process of updating this (these) dimension table(s) for the following updates (the attributes for which no value is specified are not important for this exercise). Clearly indicate which tuples are added in what table(s). Every update is assumed to take place during different data warehouse updates. That is: between every two changes in the source databases there is a data warehouse update.

i. A new customer with passport number 1234 and name Jan Janssens is added to the database. He is single, has no children and has a perfect credit score A+.
Solution: A new tuple is added to DimCustomer. A new surrogate key CID is generated for this tuple. Valid is set to true for this tuple; start is the current time, and end is null. One fact is added to the measureless fact table CustScore, connecting the CID of Jan Janssens with the right combination of scores.

ii. The customer with passport number 1234 gets married.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated marital status. A new fact is added in CustScore to connect this new version to the correct combination of scores.

iii. Customer 1234 buys a house. Obviously his address changes.

3. (2.5p) An insurance company has a data warehouse holding all contracts of its customers. One of the dimensions is *dimCustomer* which is storing information about customers. This dimension holds the following properties for a customer:

- passport number. The passport number uniquely identifies the customer.
- first name, middle name, last name
- date of birth and gender
- * street, number, postal code, city, country
- * marital status (single, married, divorced, widowed)
- * number of children
- * whether or not he or she is a home owner
- * whether or not he or she owns a car
- * whether or not he or she is a VIP client
- * credit score (A+, A, B, C, or D)
- * market segment (a number from 1 to 8)
- * risk categorization as a car driver (number 1 to 10; starts at 5, decreases 1 per year, increases if person has an accident)
- * health categorization (low risk, medium, or high risk)

The attributes with a star indicate attributes for which the history has to be maintained. The attributes without * are in principle not changing, except maybe for errors that may have to be corrected. You may assume that the address of a person, his or her marital status and number of children, whether he or she is a homeowner/car-owner, and whether he or she is a VIP client do not change very frequently. The credit score, market segment, risk and health categorizations, however, change frequently.

(a) Give the relational table(s) with its (their) attributes for storing this dimension in a ROLAP solution.

Solution: To keep track of the type-II changes, we need to introduce versioning of the tuples. Therefore we need at least a surrogate key. Furthermore, later when updating the database, we will need to be able to identify the current version of the tuple. As such we also need either an attribute "valid" or attributes "start" and "end" that indicate the valid time of the tuple.

Furthermore, since some of the attributes are rapidly changing, it is better to split the dimension into two separate dimensions; one of them a so-called *mini-dimension*. The mini-dimension consists of the attributes credit score, market segment, risk, and health categorizations. These are placed in a separate dimensional table with its own surrogate key. The mini-dimension should not be versioned, and the key of the mini-dimension needs to be added in the fact table, not in the customer dimension table. Given that the mini-dimension contains only few attributes, each with only few different values, it is a good idea to completely materialize it right from the start. It does not give any benefit, on the contrary even, to further split up the customer dimension. If order to maintain the changes in the customer dimension, even if the customer does not generate new facts, one could add a measureless

4. (1p) Consider the following relational model (this relational model was one of the solutions of the exercises in week 1):

   State(sName, region)
   District(dID, dName, sName)
   Congressman(cName, dID, sDate, party)
   Bill(bName, bDate, passed)
   Sponsors(cName, bName, bDate)
   Voted(cName, bName, bDate, vote)
   There are the following foreign key dependencies:

   - District(sName) references State
   - Congressman(dID) references District
   - Sponsors(cName) references Congressman and Sponsors(bName,bDate) references Bill
   - Voted(cName) references Congressman and Voted(bName,bDate) references Bill

   For this database, give an example of a bitmap-join index, and a query that benefits from this index.

Solution: Examples of a useful bitmap-join index are: for sName into table Sponsors (sName into Voted, for party into sponsors if in combination with other indices, for party into voted, if in combination with other indices). Example query for which this index is useful:

```
SELECT count(distinct bName, bDate)
FROM Sponsors S, Congressman C, District D
WHERE S.cName=C.cName AND C.dID=D.dId AND D.sName="Florida";
```

For this query we don't even have to do any of the joins, the result can directly be derived from the index.

(Common errors included: giving a bitmap index instead of a bitmap join index, or selecting an index that was insufficiently selective—unless used in combination with other bitmap/bitmap-join indices; stating a query that does not have a selection on the indexed attribute.)

Solution: Similar as previous.

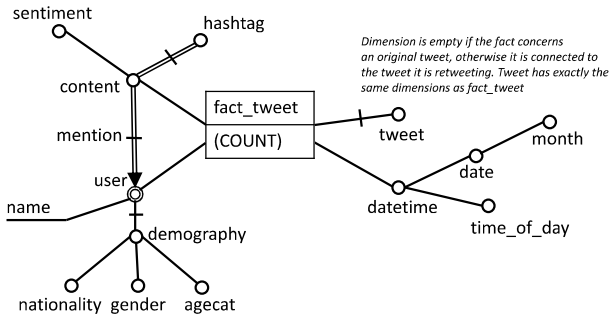
iv. The name of customer 1234 is corrected to Jan Jansens (one s is removed from the last name).
Solution: This is a type 1 change. All versions of the tuple for customer 1234 are updated.

v. Customer 1234 becomes the father of twins. His credit score drops to B.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated parenthood status. A new fact is added in CustScore to connect this new version to the updated combination of scores.

vi. Customer 1234's credit score raises again to A. He buys a car.
Solution: The current version of the tuple for 1234 (the tuple with Valid=true and end is null) gets updated: end is set to the current time, and Valid to false. A new version for 1234 is inserted with his updated car owner status. A new fact is added in CustScore to connect this new version to the updated combination of scores.

Solution:

(a) The main difficulty in this assignment is how to deal with retweets; essentially a retweet is a tweet, and tweet is the logical choice for the fact. This leaves us with a number of options; either we completely copy the model part representing a tweet, once for the fact and once for the dimension. On the other hand we could also consider tweet as the single dimension, and make a "tweetevent" fact. But, in the translation this would result in a fact table with only one attribute referring to the identifier of the tweet. Also some intermediate solutions are possible, where a dimension tweet is introduced which contains relevant tweet information that is also of interest for the retweets, and other dimensions are directly connected to the fact. In the solution below we have opted for considering a retweet as a separate dimension which is structurally equivalent to the tweet fact. In the second part of the assignment we take this into account by storing them together in one table.



Notice that with this schema we can answer all proposed queries. For instance for the last one (number of retweets per month of tweets by @realDonaldTrump): count the number of tweets by month, that are retweets of a tweet by user @realDonaldTrump.

(b) For tweets and retweets the same table factTweet will be used. Therefore we have chosen to introduce a surrogate key into the fact table instead of using the combination of dimension keys as primary key. Depending on the solution chosen, however, other approaches may be correct as well. Primary keys are underlined, and foreign keys are identified by identical names; only if there is a foreign key dependency attribute names are reused in our solution. There is one additional foreign key dependency: ReTweetID in table factTweet refers to primary key TweetID in factTweet.

- dimDate(DID,day,month,year)
- dimTime(TID,hour,minute,second)
- hashtagGroup(HGID,hashtag)
- mentionGroup(MGID,UID)
- dimUser(UID,name,DemID)
- dimDemography(DemID,nationality,gender,agecat)
- dimContent(TweetID,text,sentiment,HGID,MGID)
- factTweet(TweetID,DID,TID,UID,ReTweetID)

INFOH419 Data Warehousing — January 2017

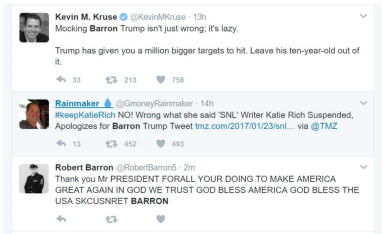
Toon Calders - Hatem Haddad

The exam is **open book**, so all books and notes can be used. "Open book" implies that you can refer to a specific slide, book page, or exercise. Verbatim copying lecturing material will not be rewarded. Stay focused on the question and avoid excessively long answers; succinct, to the point answers will be rewarded.
The questions in the exam should all be self-explanatory. In case of doubt, make a short note and solve the question to the best of your knowledge.
The maximal time to complete this exam is 3h and will be strictly observed.

---

1. (6 points) Consider the following requirement analysis: *We would like to store tweets of users in order to do market analysis. For that purpose, we capture the twitter stream, and store the information of interesting tweets in a data warehouse. For each tweet we store: time and date; the message itself; hashtags (for instance #DataWarehousing); mentions of users (for instance @tcalders); whether they are re-tweets and if so, of which tweet; and the user that sent the tweet. Furthermore, we perform natural language processing on the tweets resulting in a characterization of the sentiment of the tweet (euphoric, happy, neutral, sad, angry, and outrageous are the possible values.) Of all users we have the full name as registered by twitter. For some users, but not all, we have the demographic characteristics age category, gender, and nationality.*

   For example, for the following tweets on the left, the data is displayed on the right.



Tweet 1: user KevinMKruse full name Kevin M. Kruse
message sent 24/1/2017 at 2:43
sentiment: sad
no mentions, no hashtags

Tweet 2: user GmoneyRainmaker full name Rainmaker
message sent 24/1/2017 at 1:55
sentiment: angry
mentions @TMZ, hashtags #keepKatieRich

Tweet 3: user RobertBarron5 full name Robert Barron
message sent 20/11/2016 at 13:55
sentiment: euphoric
no mentions, no hashtags

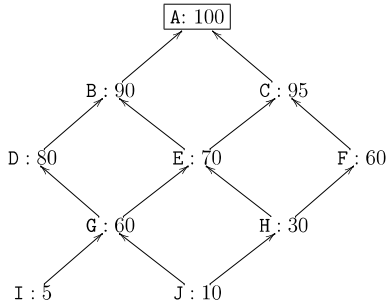
   Each tweet has exactly one user and one sentiment associated to it, but may have 0 or more mentions, and 0 or more hashtags. A tweet can be a retweet of exactly one other tweet, or be an original tweet.

   *The datawarehouse storing the tweets will be used afterwards to answer questions **like** (non-exhaustive list!): for the month December 2016, give for each sentiment category the number of tweets which contained #trump; give per month the number of tweets which contain hashtag #barron; give per month how many times the tweets of user @realDonaldTrump are retweeted on average.*

   (a) Draw a dimensional fact model (DFM) for the data warehouse that will hold this information. Notice that there is not a single correct solution and that multiple facts may be needed. For every fact in your model explain succinctly what it represents. Not all information in the description is necessarily relevant for the dimensional fact model. Your model should represent the description as faithfully as possible.

   (b) Translate the DFM you constructed in (a) to an appropriate relational model. Clearly indicate primary and foreign keys in your tables.

2. (3 points) An important technique to speed up analytical queries is by pre-computing and materializing aggregations. Consider the following lattice of views that can be requested by the user, along with the size of each view.

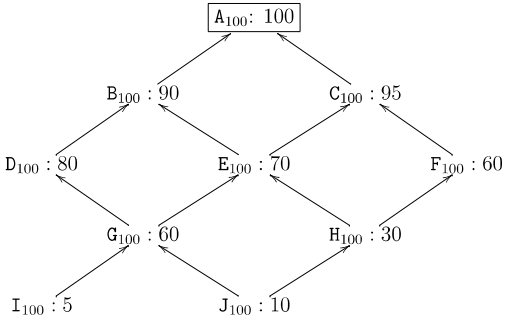


A is the view representing the base relation. The edges indicate the relation "can be computed from."

(a) Suppose that only the top-view A has been materialized. Select 4 additional views from the views B, C, D, E, F, G, H, I, and J to materialize. Apply the greedy method described by *Harinarayan, Rajaraman, and Ullman* in their seminal paper "Implementing Data Cubes Efficiently" (SIGMOD 1996) in order to optimize the overall query time.

(b) What is the gain under the cost model of *Harinarayan et al.* that you obtain by materializing these 4 views?

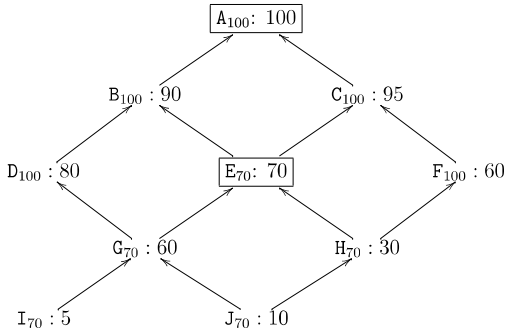
Solution: (a) As long as only A is materialized, the costs for computing the views is as follows (cost in subscript):



The benefits of materializing the different views are as follows:

View	benefit
B	$7 \times 10 = 70$
C	$7 \times 5 = 35$
D	$4 \times 20 = 80$
E	$\mathbf{5 \times 30 = 150}$
F	$3 \times 40 = 120$
G	$3 \times 40 = 120$
H	$2 \times 70 = 140$
I	$1 \times 95 = 95$
J	$1 \times 90 = 90$

Hence we first materialize E, resulting in a benefit of 150. The costs of the queries (in subscript) under the materialized views (in rectangles) now become:



The benefits of materializing an additional view are as follows:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$1 \times 40 + 2 \times 10 = 60$
G	$3 \times 10 = 30$
H	$\mathbf{2 \times 40} = 80$
I	$1 \times 65 = 65$
J	$1 \times 60 = 60$

Hence we materialize H, resulting in an additional benefit of 80. The costs of the queries

3. (2 points) Are the following statements true or false? Next to your answer, provide a justification in at most 140 characters.

   (a) A junk dimension combines several low-cardinality flags and attributes into a single dimension table rather than modeling them as separate dimensions. The main benefit of a junk dimension is that in this way the size of the fact table is reduced.
   Solution: True; for the combined junk dimensions now only one surrogate key needs to be stored in the fact table. *One could argue that this is not the main benefit, but that the main benefit comes from not having to perform the joins to get the values of the junk dimensions. In very specific situation this may be true; if the junk dimensions have only few values and are not degenerate (hence more than just flags; for instance also an explanation field and maybe a small hierarchy), and many queries specify many or all junk dimensions for slicing and rolling up, it could be the case that avoiding the joins gives some benefit. Notice, however, that the junk dimensions are small and surely fit in memory which would render the joins very cheap. The gain we get by having to use less bytes per fact will however vastly outweigh the benefit of having lesser joins in almost all cases.*

   (b) The lower the cardinality of an attribute, the more useful a bitmap index becomes.
   Solution: False; too low cardinality leads to poor query selectivity resulting in the index not being used.

   (c) For a rapidly-changing dimension where all changes are handled as type-1 changes, it does not make sense to introduce a mini-dimension.
   Solution: True; you would only pay the price of having to do additional joins without having the benefit of space and redundancy reduction.

   (d) View materialization as a query optimization technique for OLAP only makes sense if the measures are distributive.
   Solution: False; also for algebraic measures it makes sense and if the exact view is requested obviously materialization speeds up computations.

4. (2 points) Explain what is run-length encoding and why it is particularly effective in bitmap indices.
   Solution: Especially when the cardinality of an attribute grows, the bitmaps in the bitmap index will contain large streaks of 0's. Under the run-length encoding these streaks will be compressed by storing the length of the streak instead of the sequence of 0's. For instance, in case we chose for a run-length encoding of streaks of 0's only, the following bitmap:

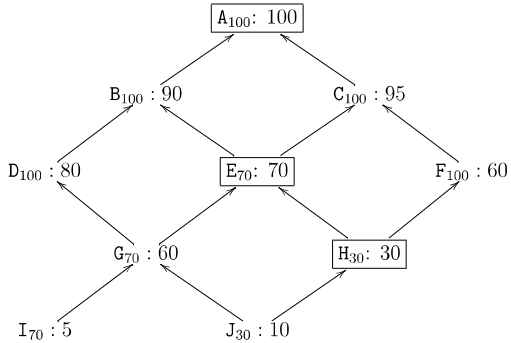
   $$000000000001100000001000001000000000000$$

   could alternative be stored as:

   $$11; 0; 7; 5; 12\ .$$

   This leads to huge space savings resulting in more bitmaps fitting into memory and/or in one disk block, significantly speeding up query processing. Important here is also to remark that the bitwise operations we need to perform on the bitmaps to resolve queries can be carried out directly on the run-length encoded bitmaps.

(in subscript) under the materialized views (in rectangles) now become:



The benefits of materializing a third view becomes:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$1 \times 40 = 40$
G	$2 \times 10 = 20$
I	$\mathbf{1 \times 65 = 65}$
J	$1 \times 20 = 20$

Hence we materialize I, resulting in an additional benefit of 65. Only the cost of I itself decreases to 5.

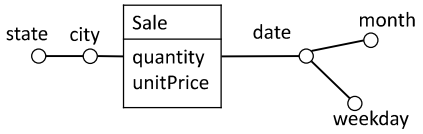
The benefits of materializing a fourth view are now:

View	benefit
B	$2 \times 10 = 20$
C	$2 \times 5 = 10$
D	$1 \times 20 = 20$
F	$\mathbf{1 \times 40 = 40}$
G	$1 \times 10 = 10$
J	$1 \times 20 = 20$

The last materialized view is hence F resulting in an additional benefit of 40.

(b) The total benefit of materializing these 4 views is: 150+80+65+40=335. This is an improvement of 335/1000 = 33,5% over the original cost.

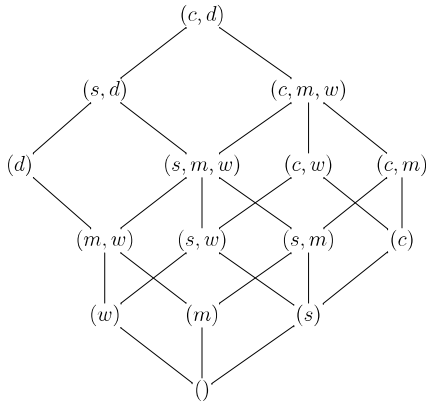
5. (4 points) Consider the following simple DFM



(a) Draw the roll-up lattice for the data cube modelled by this DFM.

(b) We want to completely materialize this data cube. Explain how this computation of the complete data cube could be optimized.

Solution: (a)

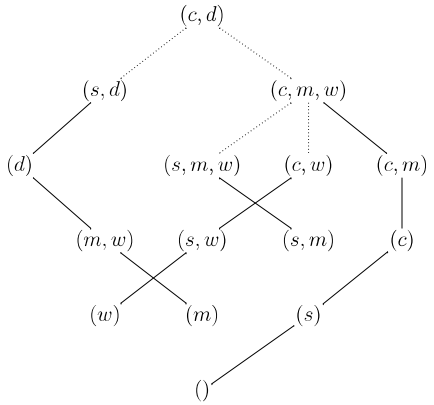


(b) The original table from which we need to compute all aggregations must contain all dimensional attributes which we need. That is, we have attributes (c, s, d, m, w). Given the attributes it is not unreasonable to assume that all views fit together into memory and hence we can use the hash-based aggregation algorithm. We maintain 15 hash-tables in memory, one for each cuboid, and compute the fully materialized data cube in one scan over the original relation.

Alternatively, if the cuboids do not fit into memory, we could use the pipe-sorting algorithm. This algorithm is based on sort-based aggregation and organizes the attributes in such order that the output of one aggregation is immediately ordered correctly for computing a next cuboid. The execution can subsequently be pipelined (see slides on pipe-sort). The roll-up lattice in (a) can be decomposed in the following pipes:

- (s,c,m,w,d) → (s,c,m,w) → (s,c,m) → (s,c) → (s) → ()
- (m,w,d,s) → (m,w,d) → (m,w) → (m)
- (w,s,c) → (w,s) → (w)
- (s,m,w) → (s,m)

Hence, with only 4 sorting operations and 4 full table scans we can compute all 15 cuboids. The pipes can be visualized as follows (we maintained the original order of the attributes; dotted line means full table scan and sorting needed):



Université libre de Bruxelles
Faculté des Sciences Appliquées

January 28, 2022

INFO-H-419 – Data Warehouses

First session examination

Question 1: Data Warehouse Design (8 points)

Consider the data warehouse of a university that contains information about teaching and research activities. On the one hand, the information about teaching activities is related to dimensions department, professor, course, and time, the latter at a granularity of academic semester. Measures for teaching activities are number of hours and number of credits. On the other hand, the information about research activities is related to dimensions professor, funding agency, project, and time, the latter twice for the start date and the end date, both at a granularity of day. In this case, professors are related to the department to which they are affiliated. Measures for research activities are the number of person months and amount.

1. Design a conceptual schema for the data warehouse. Propose dimension attributes and dimension hierarchies.
2. Translate the conceptual schema into a relational schema. Clearly indicate primary and foreign keys in your tables.
3. For the relational schema obtained in the previous question, write in SQL the following queries.
   (a) By department, total number of teaching hours during the academic year 2018–2019.
   (b) By department, total amount of research projects during the calendar year 2018.
   (c) By department, total number of professors involved in research projects during the calendar year 2018.
   (d) By professor, total number of courses delivered during the academic year 2018–2019.
   (e) By department and funding agency, total number of projects started in 2018.

Answer A MultiDim schema for this application is given in Fig. 1. The translation of this schema into a relational schema is given in Fig. 2. The SQL queries are given next.

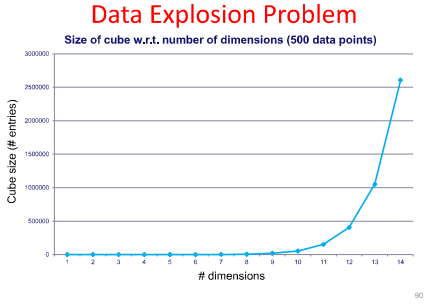
1. Total number of kilometers made by Alstom trains during 2018 departing from French or Belgian stations.

```
SELECT SUM(Distance)
FROM   Segments F, Time T, Train TR, Model M,
       Constructor C, Station S, City CI, State ST, Country CO
WHERE F.FromTimeKey = T.TimeKey AND T.Year = '2018' AND
       F.TrainKey = TR.TrainKey AND
       TR.ModelKey = M.ModelKey AND
       M.ConstructorKey = C.ConstructorKey AND
       C.ConstructorName = 'Alstom' AND
       F.FromStationKey = S.StationKey AND
       S.CityKey = CI.CityKey AND
       CI.StateKey = ST.StateKey AND
       ST.CountryKey = CO.CountryKey AND
       ( CO.CountryName = 'France' OR
       CO.CountryName = 'Belgium' )
```

2. Total duration of international trips during 2018, that is, trips departing from a station located in a country and arriving at a station located in another country.

```
SELECT SUM(Duration)
FROM   Segments F, Time T, Station A1, City C1, State S1,
```


6. (3 points) When we studied the database explosion problem, the following graph was shown:



This graph illustrates how quickly the size of the fully materialized data cube grows in function of the number of dimensions. Explain the reasons behind the exponential growth in function of the number of dimensions and why it only occurs when the cube is sparse.
Solution: When we fully materialize a data cube, we have to compute all possible aggregations as well. Suppose that we have attributes $A_1,\ldots,A_n$ with active domain size $|dom(A_i)| = d_i$ for all $i = 1 \ldots n$. Then there exist $\Pi_{i=1}^{n} d_i$ possible non-aggregated values in the cube versus $\Pi_{i=1}^{n}(d_i + 1)$ total values (original values plus aggregations). The ratio between these two values is :

$$\frac{\Pi_{i=1}^{n}(d_i+1)}{\Pi_{i=1}^{n} d_i} = \Pi_{i=1}^{n} \frac{(d_i+1)}{d_i} .$$

Even though this ratio is exponential in the number of dimensions, its growth rate is very modest (For instance: for domain sizes of 1000 or more the additional size is 2% for 20 dimensions). Hence, in cubes where almost all combinations of the dimensions are present (a dense cube), the growth will be relatively modest.

The story changes completely, however, when cubes are sparse. Even when only a limited number of possible dimension combinations exists, already a significant amount of aggregations may exist. Moreover, the probability that a certain aggregated value is present, increases exponentially with the number of dimensions that are aggregated over. For a mathematical elaboration see the additional material on the explosion problem that is available in the course notes. The graph is depicting this blow-up; in a situation with only few possible dimension combinations present (500 points out of 2^n with n the number of dimensions, the size of the fully materialized cube is plotted for an increasing number of dimensions. Here indeed we see an enormous exponential growth in the size of the data cube, which is called the database explosion problem. The database explosion problem can hence be defined as *the phenomenon of the fast exponential growth of the fully materialized cube size in function of the number of dimensions which manifests itself in sparse data cubes.*

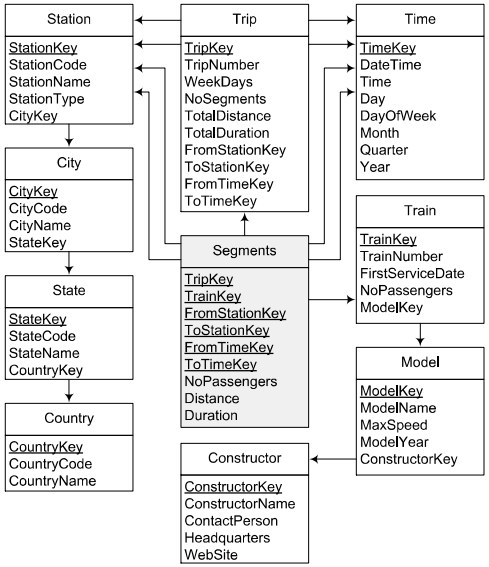


Figure 2: A snowflake schema for the data warehouse

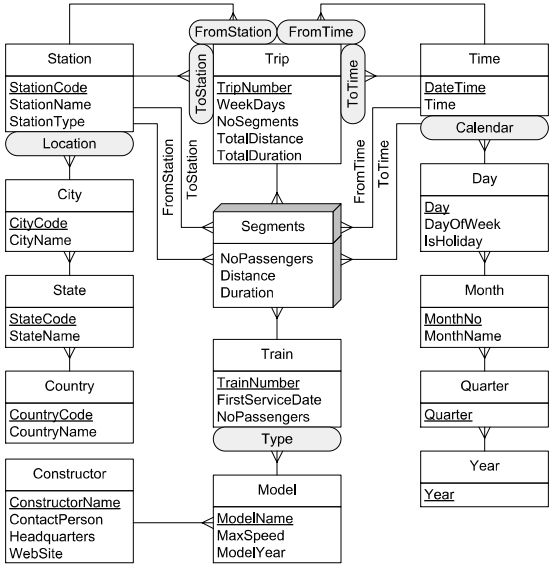


Figure 1: MultiDim schema for the train data warehouse

```
        CO1.CountryName = 'Belgium' AND
        F.ToStationKey = A2.StationKey AND
        A2.CityKey = C2.CityKey AND
        C2.StateKey = S2.StateKey AND
        S2.CountryKey = CO2.CountryKey AND
        CO2.CountryName = 'Belgium'
```

5. For each trip, average number of passengers per segment, that means, take all the segments of each trip, and average the number of passengers.

```
SELECT    T.TripNumber, AVG(NoPassengers)
FROM      Segments F, Trip T
WHERE     F.FromTimeKey = T.TripKey
GROUP BY T.TripNumber
```

Question 2: View Materialization (4 points)

Consider the graph in Fig. 3, where each node represents a view and the numbers are the costs of materializing the view. Assume that the bottom of the lattice is materialized, and that the probability of the different views is as follows: B 5%, AC 10%, BC 15%, CD 20%, ABC 25%, ACD 25%, and 0% for the other views. Determine using the View Selection Algorithm the five views to be materialized first.

Answer The results of five iterations of the algorithm are shown in Fig. 4. Notice that in the fourth iteration there is a tie between views BCD and CD. We have chosen arbitrarily to materialize the latter.

Question 3: Indexing (4 points)

Consider the tables Sales, Employee, and Department given below.


```
        Station A2, City C2, State S2
WHERE   F.FromTimeKey = T.TimeKey AND T.Year = '2018' AND
        F.FromStationKey = A1.StationKey AND
        A1.CityKey = C1.CityKey AND
        C1.StateKey = S1.StateKey AND
        F.ToStationKey = A2.StationKey AND
        A2.CityKey = C2.CityKey AND
        C2.StateKey = S2.StateKey AND
        S1.CountryKey <> S2.CountryKey
```

3. Total number of trains that departed from or arrived at Paris during July 2018.

```
SELECT COUNT(*)
FROM   Segments F, Time T, Trip TR, Station S, City C
WHERE  F.FromTimeKey = T.TimeKey AND
       T.Month = 'July' AND T.Year = '2018' AND
       ( F.FromStationKey = S.StationKey OR
       F.ToStationKey = S.StationKey ) AND
       S.CityKey = C.CityKey AND
       C.CityName = 'Paris'
```

4. Average duration of train segments in Belgium in 2018.

```
SELECT SUM(Duration)
FROM   Segments F, Time T, Station A1, City C1, State S1,
       Country CO1, Station A2, City C2, State S2, Country CO2
WHERE  F.FromTimeKey = T.TimeKey AND T.Year = '2018' AND
       F.FromStationKey = A1.StationKey AND
       A1.CityKey = C1.CityKey AND
       C1.StateKey = S1.StateKey AND
       S1.CountryKey = CO1.CountryKey AND
```

Department Key	Department Name	Location
d1	Management	Brussels
d2	Production	Paris
d3	Marketing	London
d4	Human Resources	Brussels
d5	Research	Paris

Propose an indexing scheme for the tables, including any kind of index you consider it necessary. Discuss possible alternatives according to several query scenarios. Discuss the advantages and disadvantages of creating the indexes.

Question 4: Aggregate Computation (4 points)

1. Draw the data cube lattice of a three-dimensional cube with dimensions A, B, and C.
2. Extend the lattice to take into account the hierarchies $A \rightarrow A_1 \rightarrow All$ and $B \rightarrow B_1 \rightarrow B_2 \rightarrow All$. Since the lattice is complex to draw, represent it by giving the list of nodes and the list of edges.

Answer

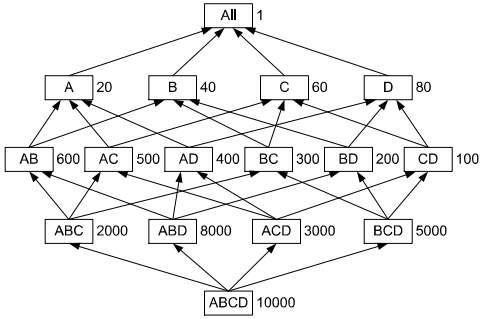


Figure 3: A data cube lattice

View	Iteration				
	1	2	3	4	5
ABC	**64000**				
ABD	14000	8000	4000	2000	2000
ACD	49000	**28000**			
BCD	35000	20000	10000	5000	**5000**
AB	37600	5600	5600	2800	2800
AC	38000	6000	6000	4500	3000
AD	38400	22400	8400	4200	4200
BC	38800	6800	6800	3400	1700
BD	39200	23200	**16200**		
CD	39600	23600	9600	**5000**	
A	19960	3960	3960	2160	2060
B	19920	3920	3920	320	220
C	19880	3880	3880	2080	80
D	19840	19840	5840	240	44
All	9999	1999	1999	199	99

Figure 4: Result of the selection of the five views to be materialized

RowID Sales	Product Key	Customer Key	Employee Key	Date Key	Sales Amount
1	p1	c1	e1	d1	100
2	p1	c2	e3	d1	100
3	p2	c2	e4	d2	100
4	p2	c3	e5	d2	100
5	p3	c3	e1	d3	100
6	p4	c4	e2	d4	100
7	p5	c4	e2	d5	100

Employee Key	Employee Name	Title	Address	City	Department Key
e1	Peter Brown	Dr.	...	Brussels	d1
e2	James Martin	Mr.	...	Wavre	d1
e3	Ronald Ritchie	Mr.	...	Paris	d2
e4	Marco Benetti	Mr.	...	Versailles	d2
e5	Alexis Manoulis	Mr.	...	London	d3
e6	Maria Mortsel	Mrs.	...	Reading	d3
e7	Laura Spinotti	Mrs.	...	Brussels	d4
e8	John River	Mr.	...	Waterloo	d4
e9	Bert Jasper	Mr.	...	Paris	d5
e10	Claudia Brugman	Mrs.	...	Saint-Denis	d5

For SQL Server, the books devoted to Analysis Services [108], Integration Services [52], and Reporting Services [135] cover extensively these components. The tabular model in Microsoft Analysis Services is studied in [204], while DAX is covered in [205].

3.8 Review Questions

3.1 What is the meaning of the acronyms OLAP and OLTP?
3.2 Using an example of an application domain that you are familiar with, describe the various components of the multidimensional model, that is, facts, measures, dimensions, and hierarchies.
3.3 Why are hierarchies important in data warehouses? Give examples of various hierarchies.
3.4 Discuss the role of measure aggregation in a data warehouse. How can measures be characterized?
3.5 Give an example of a problem that may occur when summarizability is not verified in a data warehouse.
3.6 Describe the various OLAP operations using the example you defined in Question 3.2.
3.7 What is an operational database system? What is a data warehouse system? Explain several aspects that differentiate these systems.
3.8 Give some essential characteristics of a data warehouse. How do a data warehouse and a data mart differ? Describe two approaches for building a data warehouse and its associated data marts.
3.9 Describe the various components of a typical data warehouse architecture. Identify variants of this architecture and specify in what situations they are used.
3.10 Briefly describe the components of Microsoft SQL Server.

3.9 Exercises

Exercise 3.1. A data warehouse of a telephone provider consists of five dimensions, namely, caller customer, callee customer, date, call type, and call program, and three measures, namely, number of calls, duration, and amount.

Define the OLAP operations to be performed in order to answer the following queries. Propose dimension hierarchies when needed.

a. Total amount collected by each call program in 2012.
b. Total duration of calls made by customers from Brussels in 2012.
c. Total number of weekend calls made by customers from Brussels to customers in Antwerp in 2012.

d. Total duration of international calls started by customers in Belgium in 2012.
e. Total amount collected from customers in Brussels who are enrolled in the corporate program in 2012.

Exercise 3.2. A data warehouse of a train company contains information about train segments. It consists of six dimensions, namely, departure station, arrival station, trip, train, arrival time, and departure time, and three measures, namely, number of passengers, duration, and number of kilometers.

Define the OLAP operations to be performed in order to answer the following queries. Propose dimension hierarchies when needed.

a. Total number of kilometers made by Alstom trains during 2012 departing from French or Belgian stations.
b. Total duration of international trips during 2012, that is, trips departing from a station located in a country and arriving at a station located in another country.
c. Total number of trips that departed from or arrived at Paris during July 2012.
d. Average duration of train segments in Belgium in 2012.
e. For each trip, average number of passengers per segment, that is, take all the segments of each trip, and average the number of passengers.

Exercise 3.3. Consider the data warehouse of a university that contains information about teaching and research activities. On the one hand, the information about teaching activities is related to dimensions department, professor, course, and time, the latter at a granularity of academic semester. Measures for teaching activities are number of hours and number of credits. On the other hand, the information about research activities is related to dimensions professor, funding agency, project, and time, the latter twice for the start date and the end date, both at a granularity of day. In this case, professors are related to the department to which they are affiliated. Measures for research activities are the number of person months and amount.

Define the OLAP operations to be performed in order to answer the following queries. For this, propose the necessary dimension hierarchies.

a. By department, total number of teaching hours during the academic year 2012–2013.
b. By department, total amount of research projects during the calendar year 2012.
c. By department, total number of professors involved in research projects during the calendar year 2012.
d. By professor, total number of courses delivered during the academic year 2012–2013.
e. By department and funding agency, total number of projects started in 2012.

,

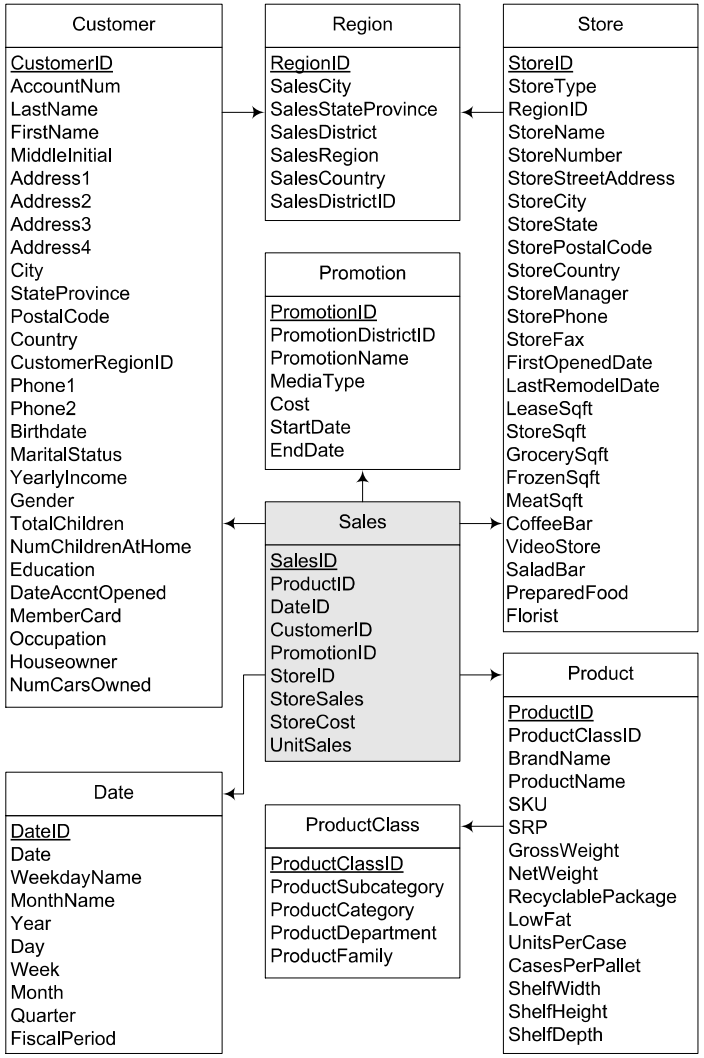


Fig. 5.41 Relational schema of the Foodmart data warehouse

5.12 Discuss how snowflake schemas are represented in Analysis Services Multidimensional and in Analysis Services Tabular.

5.13 Exercises

Exercise 5.1. Consider the data warehouse of the telephone provider given in Ex. 3.1. Draw a star schema diagram for the data warehouse.

Exercise 5.2. For the star schema obtained in the previous exercise, write in SQL the queries given in Ex. 3.1.

Exercise 5.3. Consider the data warehouse of the train application given in Ex. 3.2. Draw a snowflake schema diagram for the data warehouse with hierarchies for the train and station dimensions.

Exercise 5.4. For the snowflake schema obtained in the previous exercise, write in SQL the queries given in Ex. 3.2.

Exercise 5.5. Consider the university data warehouse described in Ex. 3.3. Draw a constellation schema for the data warehouse taking into account the different granularities of the time dimension.

Exercise 5.6. For the constellation schema obtained in the previous exercise, write in SQL the queries given in Ex. 3.3.

Exercise 5.7. Translate the MultiDim schema obtained for the French horse race application in Ex. 4.5 into the relational model.

Exercise 5.8. Translate the MultiDim schema obtained for the Formula One application in Ex. 4.7 into the relational model.

Exercise 5.9. Implement in Analysis Services a multidimensional model for the Foodmart data warehouse given in Fig. 5.41.

Exercise 5.10. Implement in Analysis Services a tabular model for the Foodmart data warehouse given in Fig. 5.41.

Exercise 5.11. The Research and Innovative Technology Administration (RITA)[1] coordinates the US Department of Transportation's (DOT) research programs. It collects several statistics about many kinds of transportation means, including the information about flight segments between airports summarized by month. It is possible to download a set of CSV files in ZIP format, one by year, ranging from 1990 up until now. These files include information about the scheduled and actually departured flights, the number of seats sold, the freight transported, and the distance traveled, among other ones. The mentioned web site describes all fields in detail.

Construct an appropriate data warehouse schema for the above application. Analyze the input data and motivate the choice of your schema.

[1] http://www.transtats.bts.gov/

```
FROM      Calls C, Time T, CallProgram P
WHERE     C.TimeKey = T.TimeKey AND T.Year = 2012 AND
          C.CallProgramKey = P.ProgramKey
GROUP BY ProgramName
```

(b) List the total duration of calls made by customers from Brussels in 2012.

```
SELECT    SUM(TotalDuration)
FROM      Calls C, Time T, Customer U
WHERE     C.TimeKey = T.TimeKey AND T.Year = 2012 AND
          C.CustomerFromKey = U.CustomerKey AND
          C.City = 'Brussels'
```

(c) List the total number of weekend calls made by customers from Brussels to customers in Antwerp in 2012.

```
SELECT    SUM(NumberCalls)
FROM      Calls C, Time T, Customer F, Customer To
WHERE     C.TimeKey = T.TimeKey AND T.Year = 2012 AND
          ( T.DayOfWeek = 'Saturday' OR
          T.DayOfWeek = 'Saturday') AND
          C.CustomerFromKey = F.CustomerKey AND
          C.CustomerToKey = To.CustomerKey AND
          F.City = 'Brussels' AND To.City = 'Antwerp'
```

(d) List the total duration of international calls started by customers in Belgium in 2012.

```
SELECT    SUM(TotalDuration)
FROM      Calls C, Time T, Customer F, Customer To
WHERE     C.TimeKey = T.TimeKey AND T.Year = 2012 AND
          C.CustomerFromKey = F.CustomerKey AND
          C.CustomerToKey = To.CustomerKey AND
          F.Country = 'Belgium' AND To.Country <> 'Belgium'
```

(e) List the total amount collected from customers in Brussels who are enrolled in the corporate program in 2012.

```
SELECT    SUM(Amount)
FROM      Calls C, Time T, Customer U, CallProgram P
WHERE     C.TimeKey = T.TimeKey AND T.Year = 2012 AND
          C.CustomerFromKey = U.CustomerKey AND
          C.CallProgramKey = P.CallProgramKey AND
          C.City = 'Brussels' AND P.ProgramName = 'Corporate'
```

5

Logical Data Warehouse Design

Exercises

5.1 Consider the data warehouse of a telephone provider given in Ex. 3.1. Draw a star schema diagram for the data warehouse.

Answer A star schema is shown in Fig. 5.1.

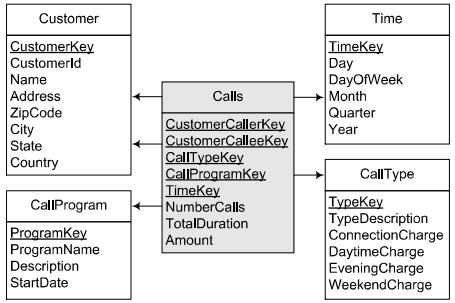


Fig. 5.1. A star schema for the data warehouse of Ex. 3.1

5.2 For the star schema obtained in the previous exercise, write in SQL the following queries.

(a) List the total amount collected by each call program in 2012.

```
SELECT    ProgramName, SUM(Amount)
```

```
        M.ConstructorKey = C.ConstructorKey AND
        C.ConstructorName = 'Alstom' AND
        F.FromStationKey = S.StationKey AND
        S.CityKey = CI.CityKey AND
        CI.StateKey = ST.StateKey AND
        ST.CountryKey = CO.CountryKey AND
        ( CO.CountryName = 'France' OR
        CO.CountryName = 'Belgium' )
```

(b) List the total duration of international trips during 2012, that is, trips departing from a station located in a country and arriving at a station located in another country.

```
SELECT SUM(Duration)
FROM   Segments F, Time T, Station A1, City C1, State S1,
       Station A2, City C2, State S2
WHERE  F.TimeFromKey = T.TimeKey AND T.Year = '2012' AND
       F.FromStationKey = A1.StationKey AND
       A1.CityKey = C1.CityKey AND
       C1.StateKey = S1.StateKey AND
       F.ToStationKey = A2.StationKey AND
       A2.CityKey = C2.CityKey AND
       C2.StateKey = S2.StateKey AND
       S1.CountryKey <> S2.CountryKey
```

(c) List the total number of trains that departed from or arrived at Paris during July 2012.

```
SELECT COUNT(*)
FROM   Segments F, Time T, Trip TR, Station S, City C
WHERE  F.TimeFromKey = T.TimeKey AND
       T.Month = 'July' AND T.Year = '2012' AND
       ( F.FromStationKey = S.StationKey OR
       F.ToStationKey = S.StationKey ) AND
       S.CityKey = C.CityKey AND
       C.CityName = 'Paris'
```

(d) List the average duration of train segments in Belgium in 2012.

```
SELECT SUM(Duration)
FROM   Segments F, Time T, Station A1, City C1, State S1,
       Country CO1, Station A2, City C2, State S2, Country CO2
WHERE  F.TimeFromKey = T.TimeKey AND T.Year = '2012' AND
       F.FromStationKey = A1.StationKey AND
       A1.CityKey = C1.CityKey AND
       C1.StateKey = S1.StateKey AND
       S1.CountryKey = CO1.CountryKey AND
```

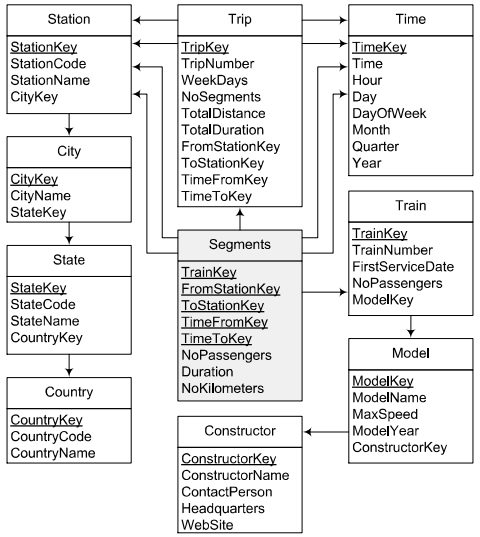



Fig. 5.2. A snowflake schema for the data warehouse of Ex. 3.2

5.3 Consider the data warehouse of the train application given in Ex. 3.2. Draw a snowflake schema diagram for the data warehouse with hierarchies for the train and station dimensions.

Answer A snowflake schema is shown in Fig. 5.2.

5.4 For the snowflake schema obtained in the previous exercise, write in SQL the following queries.

(a) List the total number of kilometers made by Alstom trains during 2012 departing from French or Belgian stations.

```
SELECT SUM(NoKilometers)
FROM   Segments F, Time T, Train TR, Model M,
       Constructor C, Station S, City CI, State ST, Country CO
WHERE  F.TimeFromKey = T.TimeKey AND T.Year = '2012' AND
       F.TrainKey = TR.TrainKey AND
       TR.ModelKey = M.ModelKey AND
```

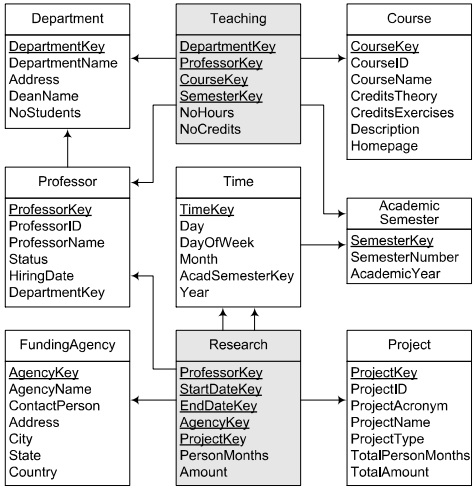


Fig. 5.3. A constellation schema for the data warehouse in Ex. 3.3

```
SELECT      DepartmentName, COUNT(ProfessorID)
FROM        Research R, Professor P, Department D, Time T
WHERE       R.ProfessorKey = P.ProfessorKey AND
            P.DepartmentKey = D.DepartmentKey AND
            R.StartDateKey = T.TimeKey AND Year = '2012'
GROUP BY DepartmentName
```

(d) List by department the total number of courses delivered during the academic year 2012–2013.

```
SELECT      DepartmentName, COUNT(CourseID)
FROM        Teaching T, AcademicSemester S, Department D
WHERE       T.SemesterKey = S.SemesterKey AND
            T.DepartmentKey = D.DepartmentKey
            AcademicYear = '2012-2013'
GROUP BY DepartmentName
```

(e) List by department and funding agency, the total number of projects started in 2012.


```
            CO1.CountryName = 'Belgium' AND
            F.ToStationKey = A2.StationKey AND
            A2.CityKey = C2.CityKey AND
            C2.StateKey = S2.StateKey AND
            S2.CountryKey = CO2.CountryKey AND
            CO2.CountryName = 'Belgium'
```

(e) For each trip, list the average number of passengers per segment, that means, take all the segments of each trip, and average the number of passengers.

```
SELECT      T.TripNumber, AVG(NoPassengers)
FROM        Segments F, Trip T
WHERE       F.TimeFromKey = T.TripKey
GROUP BY T.TripNumber
```

5.5 Consider the university data warehouse described in Ex. 3.3. Draw a constellation schema for the data warehouse taking into account the different granularities of the time dimension.

Answer A constellation schema is shown in Fig. 5.3.

5.6 For the constellation schema obtained in the previous exercise, write in SQL the following queries.

(a) List by department the total number of teaching hours during the academic year 2012–2013.

```
SELECT      DepartmentName, SUM(NoHours)
FROM        Teaching T, AcademicSemester S, Department D
WHERE       T.SemesterKey = S.SemesterKey AND
            T.DepartmentKey = D.DepartmentKey
            AcademicYear = '2012-2013'
GROUP BY DepartmentName
```

(b) List by department the total amount of research projects during the calendar year 2012.

```
SELECT      DepartmentName, SUM(Amount)
FROM        Research R, Professor P, Department D, Time T
WHERE       R.ProfessorKey = P.ProfessorKey AND
            P.DepartmentKey = D.DepartmentKey AND
            R.StartDateKey = T.TimeKey AND Year = '2012'
GROUP BY DepartmentName
```

(c) List by department the total number of professors involved in research projects during the calendar year 2012.

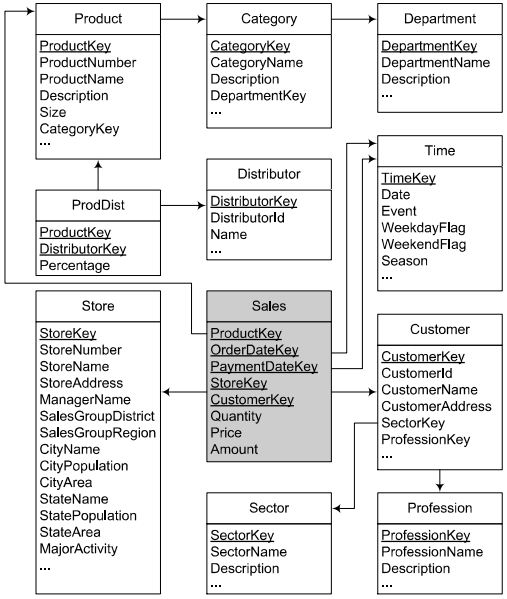


Fig. 5.5. A logical translation of the conceptual schema in Fig. 5.4

5.9 Translate the MultiDim schema obtained for the Formula One application in Ex. 4.7 into the relational model.

Answer A snowflake schema for this application is given in Fig. 5.7.


```
SELECT    DepartmentName, AgencyName, COUNT(ProfessorID)
FROM      Research R, Professor P, Department D,
          FundingAgency F, Time T
WHERE     R.ProfessorKey = P.ProfessorKey AND
          P.DepartmentKey = D.DepartmentKey AND
          R.StartDateKey = T.TimeKey AND Year = '2012' AND
          R.AgencyKey = F.AgencyKey
GROUP BY DepartmentName, AgencyName
```

5.7 Translate into the relational model the MultiDim schema given in Fig. 5.4.

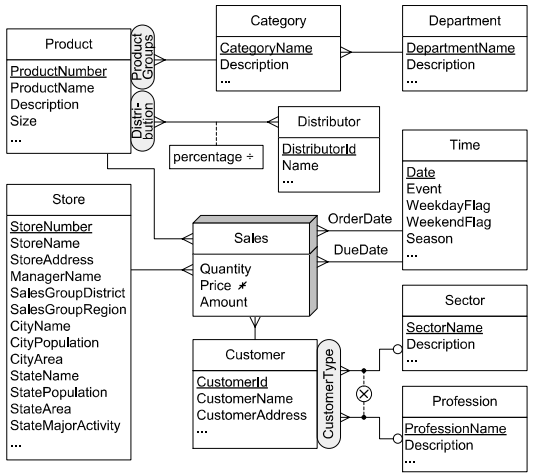


Fig. 5.4. A conceptual schema of a sales data warehouse

Answer The logical schema is given in Fig. 5.5.

5.8 Translate the MultiDim schema obtained for the French horse race application in Ex. 4.5 into the relational model.

Answer A constellation schema for this application is given in Fig. 5.6.

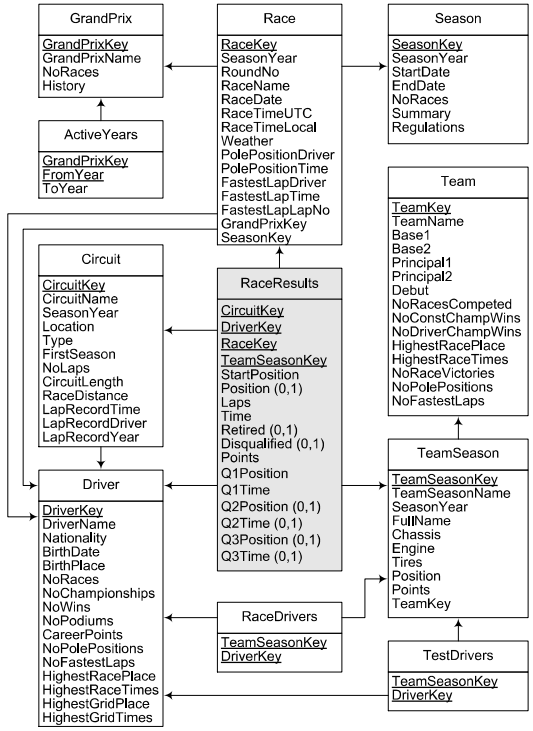


Fig. 5.7. Snowflake schema of the Formula One data warehouse in Ex. 4.7

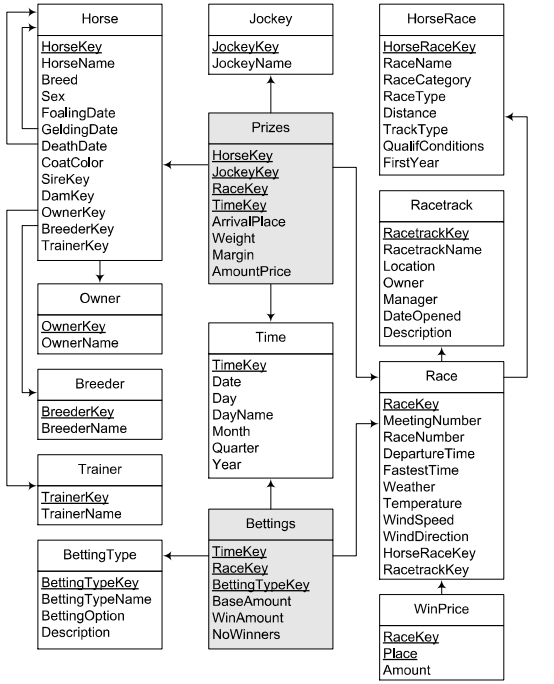


Fig. 5.6. Constellation schema of the French horse racing data warehouse in Ex. 4.5

Summaries	
DepScheduled	Departures scheduled
DepPerformed	Departures performed
Payload	Available payload (pounds)
Seats	Available seats
Passengers	Non-stop segment passengers transported
Freight	Non-stop segment freight transported (pounds)
Mail	Non-stop segment mail transported (pounds)
Distance	Distance between airports (miles)
RampTime	Ramp to ramp time (minutes)
AirTime	Airborne time (minutes)
Carrier	
UniqueCarrier	Unique carrier code. When the same code has been used by multiple carriers, a numeric suffix is used for earlier users, for example, PA, PA(1), PA(2). Use this field for analysis across a range of years.
AirlineID	An identification number assigned by US DOT to identify a unique airline (carrier). A unique airline (carrier) is defined as one holding and reporting under the same DOT certificate regardless of its code, name, or holding company/corporation.
UniqueCarrierName	Unique carrier name. When the same name has been used by multiple carriers, a numeric suffix is used for earlier users, for example, Air Caribbean, Air Caribbean (1).
UniqCarrierEntity	Unique entity for a carrier's operation region.
CarrierRegion	Carrier's operation region. Carriers report data by operation region
Carrier	Code assigned by IATA and commonly used to identify a carrier. As the same code may have been assigned to different carriers over time, the code is not always unique. For analysis, use the unique carrier code.
CarrierName	Carrier name
CarrierGroup	Carrier group code. Used in legacy analysis
CarrierGroupNew	Carrier group new

Table 5.1. Attributes of the tables T_T100I_Segment_All_Carrier_XXXX

5.10 The Research and Innovative Technology Administration (RITA)[1] coordinates the U.S. Department of Transportation's (DOT) research programs. It collects several statistics about many kinds of transportation means. The information is published at the following URL: http://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=261

There is a set of tables T_T100I_Segment_All_Carrier_XXXX, one by year, ranging from 1990 up till now. These tables report statistics about flight segments between airports summarized by month. This information includes the scheduled and actually departured flights, the number of seats sold, the freight transported, and the distance traveled, among other ones. The schema and description of these tables is given in Table 5.1. A set of lookup tables given in Table 5.2 include information about airports, carriers, time, and other ones. The schemas of these lookup tables are composed of just two columns called Code and Description. The mentioned web site describes all tables in detail.

From the information above, construct an appropriate data warehouse schema. Analize the input data and motivate the choice of your schema.

Answer We have imported all the input CSV files in database tables using the import utility provided by SQL Server. Since the fact data is split into tables Fact_1990, Fact_1991, . . . , Fact_2013, we created a view that performs the union of these tables as follows

```
CREATE VIEW Fact AS (
SELECT * FROM Fact_1990 UNION
SELECT * FROM Fact_1991 UNION
...
SELECT * FROM Fact_2013 )
```

In the tables Fact_XXXX the following attributes may be null:

Origin_Country, Dest_Country, Unique_Carrier, Airline_ID
Unique_Carrier_Name, Unique_Carrier_Entity, Carrier_Name,
Carrier_Group_New, Region

Records with either Origin_Country or Dest_Country null are obviously erroneous but this can be easily corrected since in both cases the Origin_Country_Name or Dest_Country_Name is not null and contains the values Berlin or Czechoslovakia. Since all other attributes that may be null refer to carriers, we study next data about carriers.

The attributes pertaining to carriers are the following

Unique_Carrier, Airline_ID, Unique_Carrier_Name, Unique_Carrier_Entity,
Region, Carrier, Carrier_Name, Carrier_Group, Carrier_Group_New

[1] http://www.transtats.bts.gov/

Aircraft	
AircraftGroup	Aircraft group
AircraftType	Aircraft type
AircraftConfig	Aircraft configuration
Time Period	
Year	Year
Quarter	Quarter
Month	Month
Other	
DistanceGroup	Distance intervals, every 500 Miles, for flight segment
Class	Service Class

Table 5.1. Attributes of the tables T_T100I_Segment_All_Carrier_XXXX (cont.)

L_STRCRAFT_CONFIG	L_CITY_MARKET_ID
L_STRCRAFT_GROUP	L_COUNTRY_CODE
L_STRCRAFT_TYPE	L_DISTANCE_GROUP_500
L_STRLINE_ID	L_MONTHS
L_STRPORT	L_QUARTERS
L_STRPORT_ID	L_REGION
L_STRPORT_SEQ_ID	L_SERVICE_CLASS
L_CARRIER_GROUP	L_UNIQUE_CARRIER_ENTITIES
L_CARRIER_GROUP_NEW	L_UNIQUE_CARRIERS
L_CARRIER_HISTORY	L_WORLD_AREA_CODES

Table 5.2. Lookup tables for the table T_T100I_Segment_All_Carrier_XXXX

There are many records for which only the carrier name is known among all these attributes, such as follows

```
NULL NULL NULL NULL NULL BEQ NULL 0 NULL
NULL NULL NULL NULL NULL EG  NULL 0 NULL
NULL NULL NULL NULL NULL SU  NULL 0 NULL
```

For this reason we exluded those records. These are the only ones with Unique_Carrier as null.

Thus, data about carriers can be obtained from the query below.

```
SELECT DISTINCT Unique_Carrier, Airline_ID, Unique_Carrier_Name,
       Unique_Carrier_Entity, Region, Carrier,
       Carrier_Name, Carrier_Group, Carrier_Group_New
FROM   Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```


Origin	
OriginAirportID	Origin airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.
OriginAirportSeqID	Origin airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.
OriginCityMarketID	Origin airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Origin	Origin airport
OriginCityName	Origin city
OriginCountry	Origin airport, country
OriginCountryName	Origin airport, country name
OriginWAC	Origin airport, world area code
Destination	
DestAirportID	Destination airport, Airport ID. An identification number assigned by US DOT to identify a unique airport. Use this field for airport analysis across a range of years because an airport can change its airport code and airport codes can be reused.
DestAirportSeqID	Destination airport, Airport Sequence ID. An identification number assigned by US DOT to identify a unique airport at a given point of time. Airport attributes, such as airport name or coordinates, may change over time.
DestCityMarketID	Destination airport, City Market ID. City Market ID is an identification number assigned by US DOT to identify a city market. Use this field to consolidate airports serving the same city market.
Dest	Destination airport
DestCityName	Destination city
DestCountry	Destination airport, country
DestCountryName	Destination airport, country name
DestWAC	Destination airport, world area code

Table 5.1. Attributes of the tables T_T100I_Segment_All_Carrier_XXXX (cont.)

```
SELECT DISTINCT Unique_Carrier, Unique_Carrier_Name,
        Carrier, Carrier_Name
FROM    Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```

Contrary to what it is said in the documentation, the attribute Unique_Carrier is not unique for carriers since the following query gives less answers that the previous query.

```
SELECT DISTINCT Unique_Carrier
FROM    Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```

What is unique is the combination of Unique_Carrier, Carrier, and Carrier_Name

```
SELECT DISTINCT Unique_Carrier, Carrier, Carrier_Name
FROM    Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```

since the query above has the same number of answers as the query asking for the four attributes before. For example, the following records belong to the answer of the last query.

```
ADB AUQ Antonov Company
ADB ADB Antonov Company
ADB AUQ Antonov Design Bureau
```

We can see that for the same unique carrier there are two values of carrier and two values of carrier names.

The next group of attributes of the fact table pertains to origin and destination airports. The information about all airports can be obtained by the following query

```
SELECT DISTINCT
    Origin_City_Market_ID AS City_Market_ID, Origin AS Airport_Code,
    Origin_City_Name AS City_Name, Origin_Country AS Airport_Code,
    Origin_Country_Name AS Country_Name, Origin_WAC AS WAC
FROM Fact
UNION
SELECT DISTINCT
    Dest_Airport_Seq_ID AS Airport_Seq_ID,
    Dest_City_Market_ID AS City_Market_ID, Dest AS AirportCode,
    Dest_City_Name AS City_Name, DEST_Country AS Country_Code,
    Dest_Country_Name AS Country_Name, Dest_WAC AS WAC
FROM FACT
```


As stated in the documentation, carriers report data by operation region (such as Atlantic, Domestic, International, ...). Attribute Unique_Carrier_Entity is unique for a combination of carrier and region. Therefore, the following query allows to obtain all the information pertaining to carriers without the region information.

```
SELECT DISTINCT Unique_Carrier, Airline_ID, Unique_Carrier_Name,
        Carrier, Carrier_Name, Carrier_Group, Carrier_Group_New
FROM    Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```

There is a functional dependency Unique_Carrier $\rightarrow$ Airline_ID. Indeed, the following query returns an empty answer.

```
SELECT *
FROM    Fact F1
WHERE EXISTS (
        SELECT *
        FROM    Fact F2
        WHERE F1.Unique_Carrier = F2.Unique_Carrier AND
                F1.Airline_ID <> F2.Airline_ID )
```

Further, attribute Airline_ID can be removed since the lookup table L_Airline_ID does not add any other information that is not in the attributes Carrier and Carrier_Name.

Therefore, the information about carriers is obtained as follows

```
SELECT DISTINCT Unique_Carrier, Unique_Carrier_Name,
        Carrier, Carrier_Name, Carrier_Group, Carrier_Group_New
FROM    Fact
WHERE Unique_Carrier IS NOT NULL
ORDER BY Unique_Carrier
```

A carrier may have several carrier groups and carrier groups new as illustrated by the records below which are in the answer of the above query.

```
9E Endeavor Air Inc. 9E Pinnacle Airlines Inc. 1 6
9E Endeavor Air Inc. 9E Pinnacle Airlines Inc. 2 2
```

Therefore, this will induce nonstrict hierarchies between carriers and carrier groups. Finally, we have chosen to keep only the carrier group new since the documentation states that carrier group should only be used for legacy analysis.

Carrier information without carrier groups can be obtained by the following query.

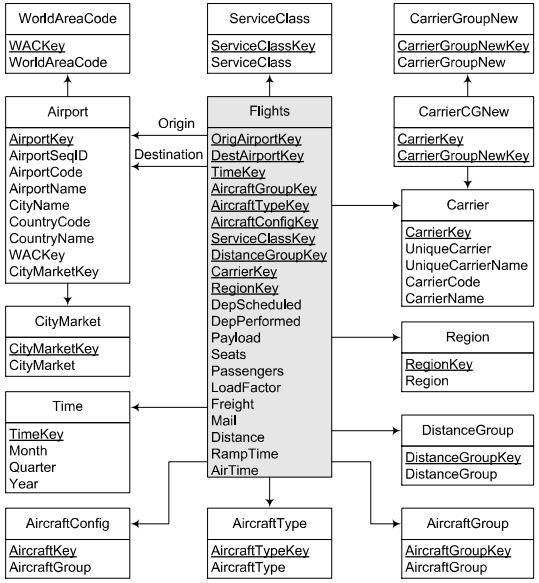


Fig. 5.9. Logical schema of the data warehouse for the air carrier example

The attribute Airport_Seq_ID is a key of the data obtained by the above query, as the number of distinct values of the attribute is the same as the number of answers of the above query.

Finally, the remaining attributes of the fact table pertain to dimensions such as AircraftGroup, AircraftType, AircraftConfig, etc., and the value of such dimensions is given in the lookup tables.

Therefore, the conceptual schema of the data warehouse is given in Fig. 5.8 and the logical schema is given in Fig. 5.9.

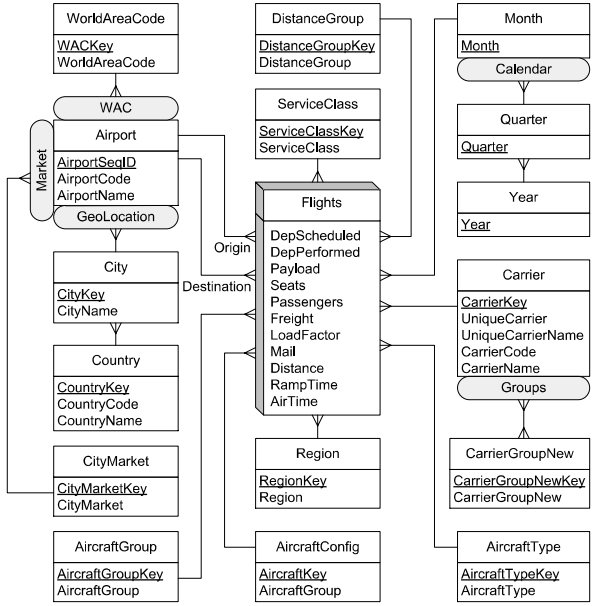


Fig. 5.8. Conceptual schema of the data warehouse for the air carrier example

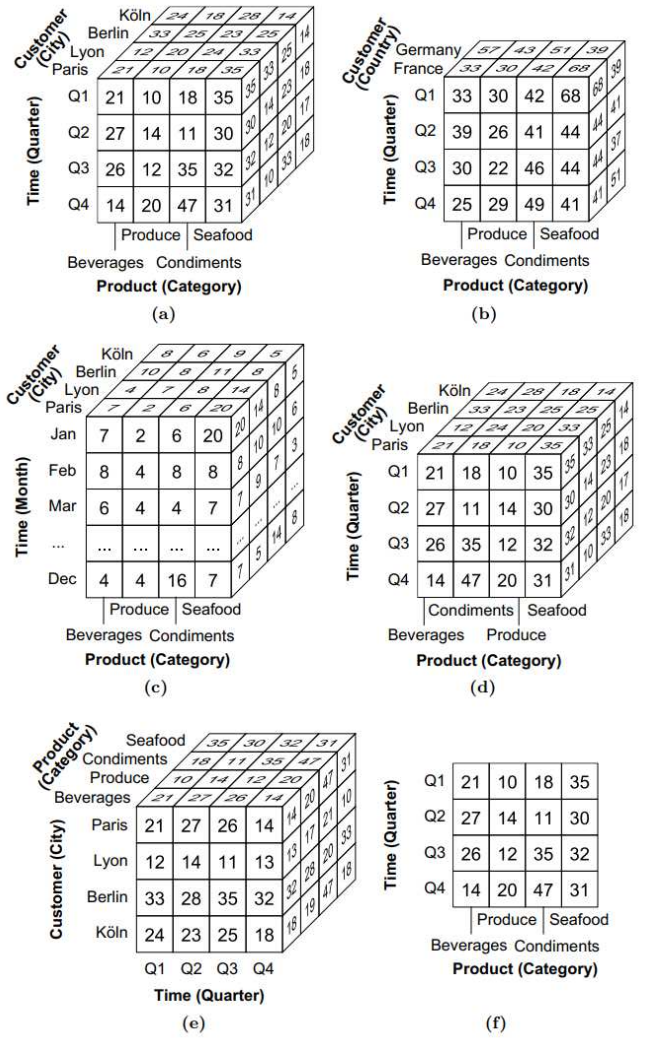


Fig. 3.4 OLAP operations. (**a**) Original cube; (**b**) Roll-up to the Country level; (**c**) Drill-down to the Month level; (**d**) Sort product by name; (**e**) Pivot; (**f**) Slice on City='Paris'

OLAP Operations

Course	INFOH419 - DW
Resource(s)	DWSDI 3.2

DWSDI 3.2 - OLAP Operations

ASC và DESC sắp xếp members trong parents của chúng (nghĩa là tính đến herarchies), còn BASC và BDESC sắp xếp theo tất cả members. Mặc định là ASC.

- VD: `SORT(Sales, Product, ProductName)`. KQ ở hình (d)

Pivot (Rotate)

- Xoay các trục của một cube để cho ra một biểu diễn khác của data. Cú pháp là:

```
PIVOT(CubeName, (Dimension → Axis)*)
```

- VD để xem Time dimension trên x axis: `PIVOT(Sales, Time → X, Customer → Y, Product → Z)`. KQ ở hình (e).

Slice

- Gỡ bỏ một dimension (nghĩa là cho ra một cube $n-1$ dimensions) bằng cách chọn một instance trong một dimension level. Cú pháp là:

```
SLICE(CubeName, Dimension, Level = Value)
```

- VD: Để xem data chỉ ở Paris: `SLICE(Sales, Customer, City = 'Paris')`. KQ ở hình (f)

Roll-up

- Aggregate measues dọc theo một dimension hierachy để có đc measures tại một coarser granularity. Cú pháp là:

```
ROLLUP(CubeName, (Dimension → Level)*, AggFunction(Measure)*)
```

- VD: `ROLLUP(Sales, Customer → Country, SUM(Quantity))`. KQ ở hình (b)
- Thông thường ta muốn roll-up vài dimensions tới các levels cụ thể và gỡ bỏ các dimensions khác bằng roll-up tới All level. Trong một cube có n dimensions, ta có thể làm điều này bằng n roll-up operations liên tiếp. Ta dùng ROLLUP* để rút gọn:

```
ROLLUP*(CubeName, [(Dimension → Level)*], AggFunction(Measure)*)
```

- VD: `ROLLUP*(Sales, Time → Quarter, SUM(Quantity))`: tính total quantity theo quarter. Operation này thực hiện roll-up dọc theo Time dimension tới Quarter level và các dimensions khác tới All level.
- Nếu dimenions ko đc chỉ rõ như: `ROLLUP*(Sales, SUM(Quantity))`
  thì tất cả dimensions đều đc roll-ed-up tới All level, cho ra một single cell là overal sum of Quantity measure.

Drill down

- Làm ngược lại roll-up operation. Cú pháp là:

```
DRILLDOWN(CubeName, (Dimension → Level)*)
```

- VD: Từ original cube ta thực hiện drill-down dọc theo Time dimension tới Month level để xem tháng nào có giá trị cao.

```
DRILLDOWN(Sales, Time → Month)
```

  KQ ở hình (c).

Sort

- Trả về một cube trong đó các members của một dimension đc sorted. Cú pháp là:

```
SORT(CubeName, Dimension, (Expression [ {ASC | DESC | BASC | BDESC} ])*)
```

Dice

- Giữ các cells TM một Boolean condition φ. Cú pháp là:

  ```
  DICE(CubeName, Condition)
  ```

- VD: `DICE(Sales, (Customer.City = 'Paris' OR Customer.City = 'Lyon') AND (Time.Quarter = 'Q1' OR Time.Quarter = 'Q2'))`. KQ ở hình (g)
- VD: `DICE(Sales, Quantity > 15)`. KQ ở hình (h)

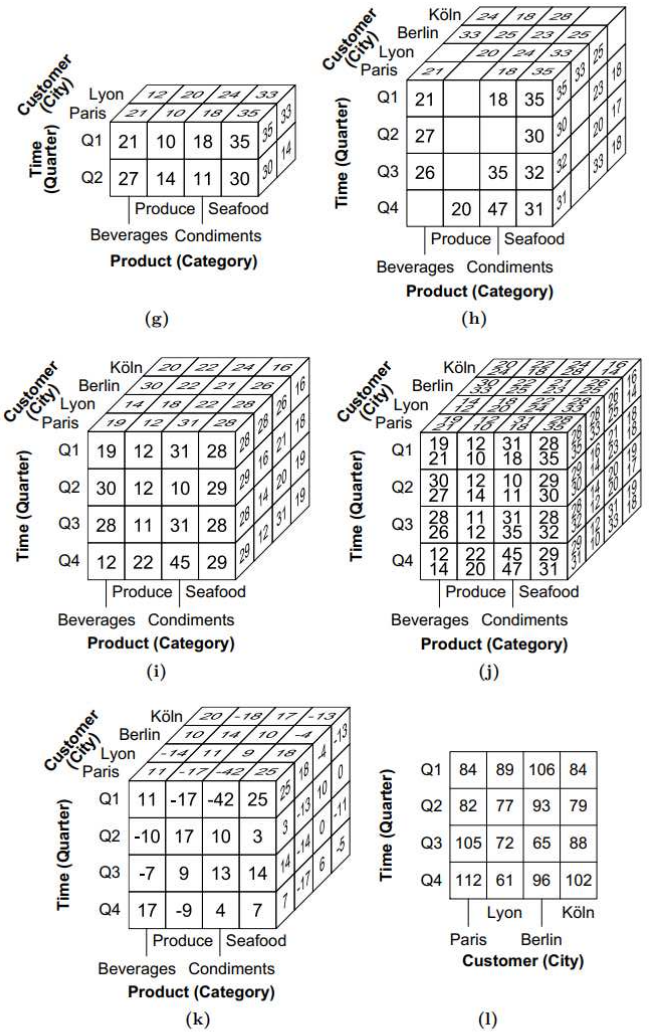
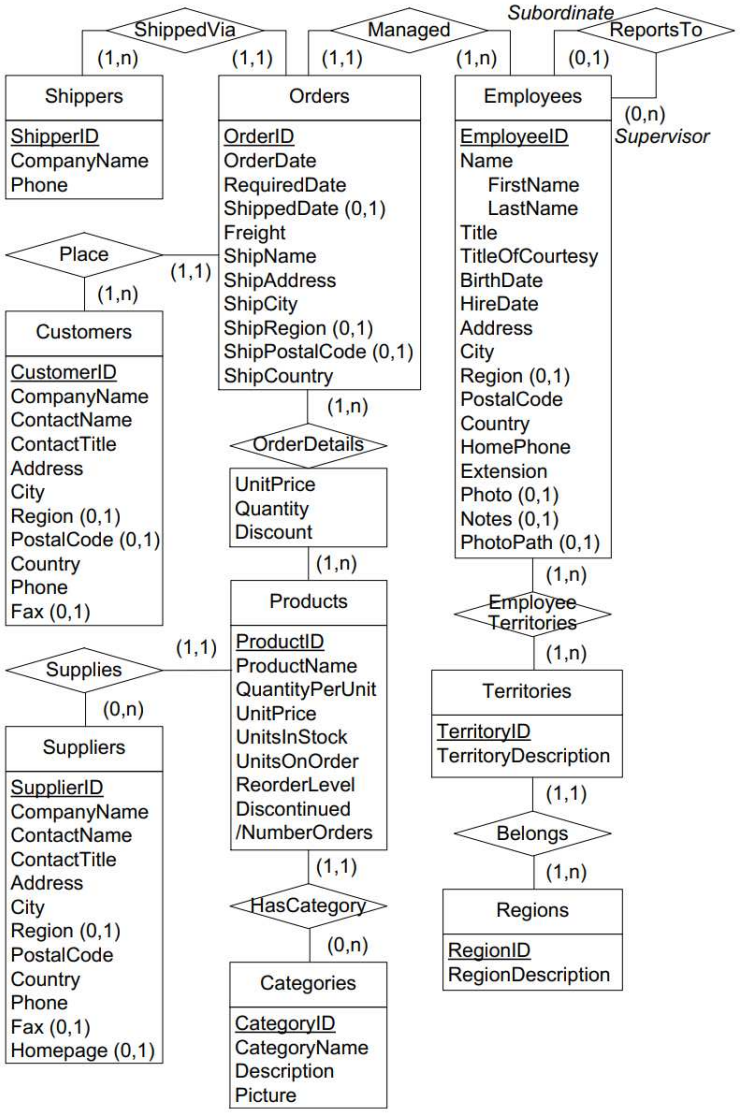


Fig. 3.4 OLAP operations (continued). (**g**) Dice on City='Paris' or 'Lyon' and Quarter='Q1' or 'Q2'; (**h**) Dice on Quantity > 15; (**i**) Cube for 2011; (**j**) Drill-across; (**k**) Percentage change; (**l**) Total sales by quarter and city



Conceptual Design and Modeling

Course	INFOH419 - DW
Resource(s)	DWSDI 2.3, 4.1, 4.2, DWDMPM 5.1, 5.2

DWSDI 2.3 - Conceptual Database Design

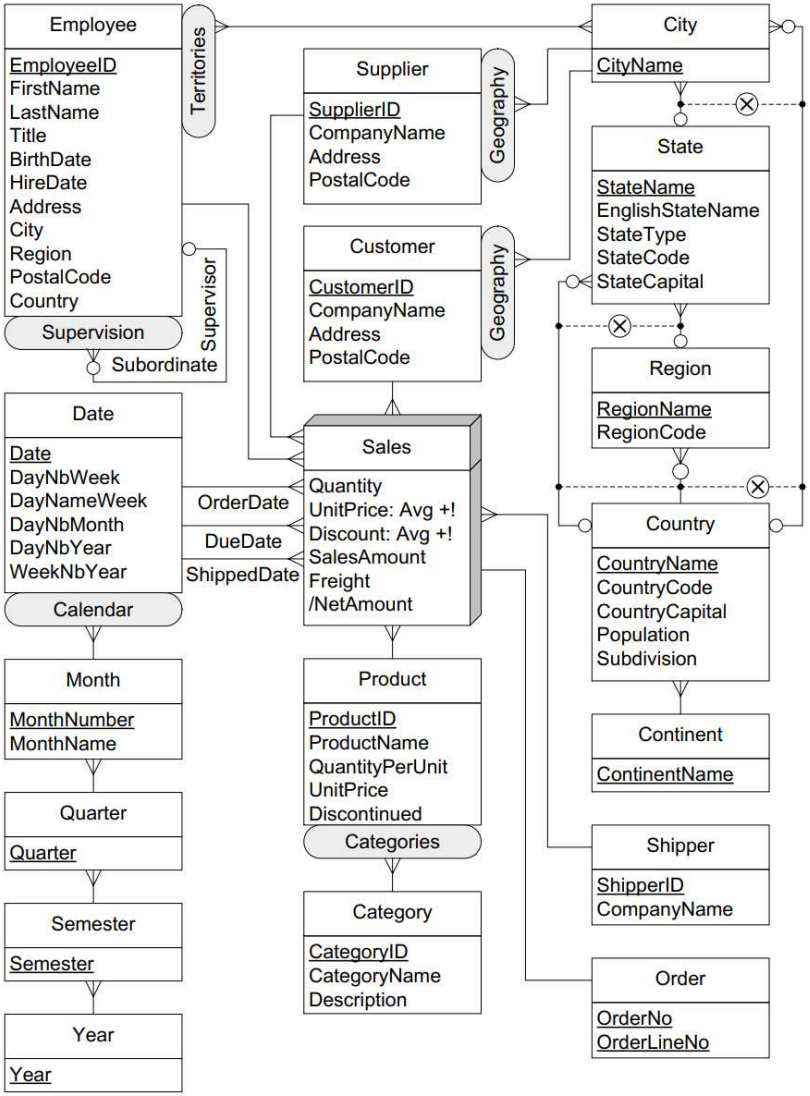
- **Entity-relationship (ER)** model thường đc dùng làm conceptual model. Hình dưới là ER model cho Northwind DB.

- Một hoặc một số attributes có thể xđ duy nhất một object cụ thể, chúng đgl **identifiers**.

DWSDI 4.1 - Conceptual Modeling of Data Warehouses

- Ko có một well-established conceptual model cho multidimensional data. Do đó, data warehouse design thường đc thực hiện trực tiếp tại logical level, dựa trên star và/hoặc snowflake schemas, gây khó hiểu cho người dùng. Vì vậy, conceptual data warehousing modeling cần có một model đứng phía trên logical level. Ở đây ta dùng MultiDim model.
- Một **schema** trong MultiDim model đc tạo nên bởi một tập dimensions và một tập facts.
- Một **dimension** đc tạo nên bởi một level, hoặc một hay nhiều hierarchies. Một hierarchy là một tập các levels.
- Một **level** là tương tự với một entity type trong ER model. Nó mô tả một tập các real-world concepts mà từ application perspective, có các characteristics tương đồng. Các instances của một level đgl **members**. VD, Product và Category là các levels. Một level có một tập các **attributes** mô tả characteristics của các members. Ngoài ra, một level có một hoặc một số **identifiers** xđ duy nhất members của nó. Mỗi identifier có một hoặc một số attributes. VD, Category**I**D là một identifier của Category level. Mỗi attribute của một level có một type, hay domain.
- Một **fact** liên quan đến một số levels. VD, Sales fact liên quan đến Employee, Customer, Supplier, Shipper, Order, Product, Date levels. Cùng một level có thể tham gia nhiều lần vào một fact, có nhiều **roles** khác nhau. VD, Date level tham gia avào Sales fact với các roles OrderDate, DueDate, ShippedDate. Các instances của một fact đgl fact **members. Cardinality** của relationship giữa facts và levels xđ số lần tối thiểu và tối đa các fact members có thể đi với level members. VD, Sales fact đi với Product theo một one-to-many cardinality, nghĩa là một sale chỉ có một product còn một product có thể có nhiều sales.
- Một fact có các attributes đgl measures. Chúng thường là numerical data đc phân tích bằng các perspectives biểu diễn bởi dimensions. Các identifier attributes của các levels kết nối với một fact sẽ xđ **granularity** của measures.

- **Entity types** đc dùng để biểu diễn một tập các real-world objects trong ứng dụng (VD: Employees, Orders, Customers). Một object thuộc về một entity type đgl một **entity** hoặc **instance.** Tập các instances của một entity type đgl **population** của nó.
- **Relationship types** đc dùng để biểu diễn quan hệ giữa các objects (VD: Supplies, ReportsTo, HasCategory). Một liên kết giữa các objects của một relationship type đgl một **relationship** hoặc một **instance**. Tập các liên kết của một relationship type đgl **population** của nó.
- Sự tham gia của một entity type vào một relationship type đgl một **role**. Mỗi role của một relationship type có một cặp **cardinalities** mô tả số lần tối thiểu và tối đa mà entity có thể tham gia vào relationship type này. VD, role giữa Products và Supplies có cardinalities (1, 1), nghĩa là mỗi product chỉ tham gia đúng một lần vào relationship type. Một role đgl **optional** hay **mandantory** tùy vào minimum cardinality của nó là 0 hay 1; và đgl **monovalued** hay **multivalued** tùy vào maximum cardinality của nó là 1 hay n.
- Tất cả relationship types ở đây đều là binary. Tùy vào maximum cardinality của mỗi role, binary relationship types đc phân loại thành **one-to-one**, **one-to-many**, và **many-to-many.** VD, relationship Supplies là one-to-many, vì một product chỉ có nhiều nhất một supplier, trong khi một supplier có thể có nhiều products.
- Cùng một entity có thể tham gia nhiều hơn một lần vào một relationship type, VD trường hợp của ReportsTo. Relationship type này đgl **recursive** và **role names** là cần thiết để phân biệt các roles khác nhau của entity type.
- **Attributes** đc dùng để mô tả các characteristics của entity hoặc relationship types. VD, Address và Homepage là các attributes của Suppliers.
- Attributes cũng có cardinalities, mô tả số giá trị mà attribute có thể có trong mỗi instance. Hầu hết attribute có cardinality (1, 1) và ko đc thể hiện ra.
- Attributes có thể đc tạo nên từ các attributes khác. VD, attribute Name trong entity type Employees đc tạo nên từ FirstName và LastName. Các attributes như vậy đgl **complex attributes**, còn lại đgl **simple attributes.** Một vài attributes có thể là **derived**, VD NumberOrders trong Products, nghĩa là nó có thể có đc từ một công thức lq đến các elements khác của schema, và đc lưu như một attribute.

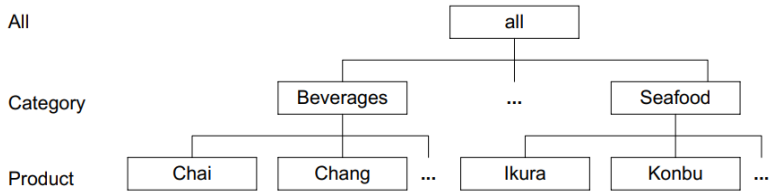


- Các measures đc aggregated dọc theo các dimensions bằng roll-up operations. Mặc định các measures là **additive**, và kí hiệu **semiadditive** (”+!”) và **nonadditive**. Các measures và level attributes cũng có thể là **derived** (”/”), nếu chúng đc tính dựa trên các measures và attributes khác trong schema.
- Một **hierarchy** bao gồm nhiều related levels. Lower level đgl **child** và higher level đgl **parent.** Cardinalities trong parent-child relationships xđ số lần tối thiểu và tối đa mà các members trong một level có thể tham gia với một member ở level khác. VD, child level Product đc related tới parent level Category theo một one-to-many cardinality, nghĩa là một product chỉ có một category và một category có thể có nhiều products.
- Một dimension có thể có nhiều hierarchies, mỗi cái mô tả một particular criterion cho analysis purposes, do đó ta dùng **hierarchy name** để phân biệt chúng. VD, Employee dimension có hai hierarchies là Territories và Supervision.
- Hai hay nhiều parent-child relationships có thể là **exclusive.** VD, states có thể đc aggregated theo regions hoặc countries, và chỉ tham gia vào một relationship đi từ State level.

DWSDI 4.2 - Hierarchies

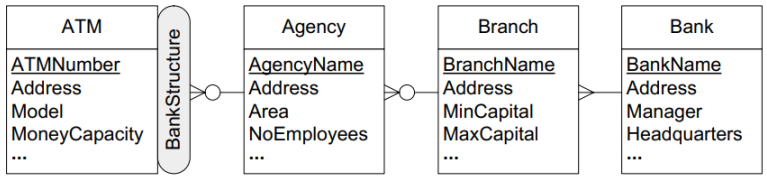
4.2.1 Balanced Hierarchies

- Một **balanced hierarchy** chỉ có một path ở schema level, trong đó tất cả levels là bắt buộc. VD, xét hierarchy Product→Category, tại instance level, các members tạo nên một tree trong đó tất cả branches có cùng length.

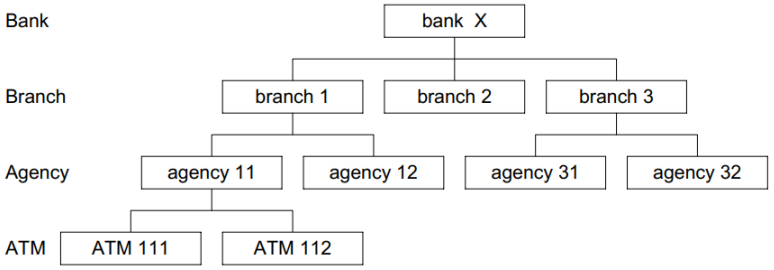


4.2.2 Unbalanced Hierarchies

- Một unbalanced hierarchy chỉ có một path ở schema level, trong đó ít nhất một level là ko bắt buộc. Do đó, tại instance level, có thể có parent members ko có child members.

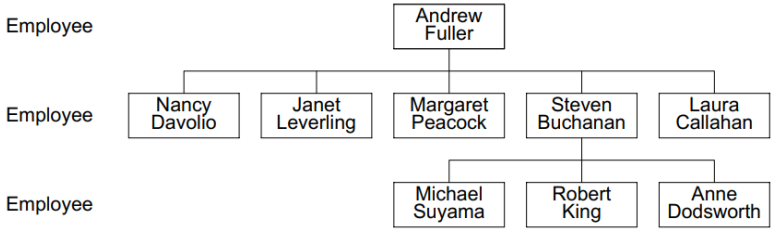


(a)



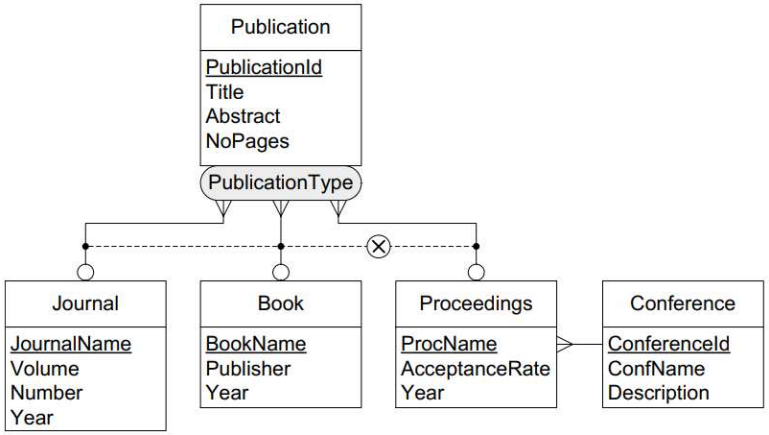
(b)

- Unbalanced hierarchies có một TH đặc biệt là recursive hierarchies, hay parent-child hierarchies. Trong TH này, cùng level đc kết nối bởi hai roles của một parent-child relationship. VD, Subordinate và Supervisor roles của parent-child relationship đc kết nối tới Employee level.

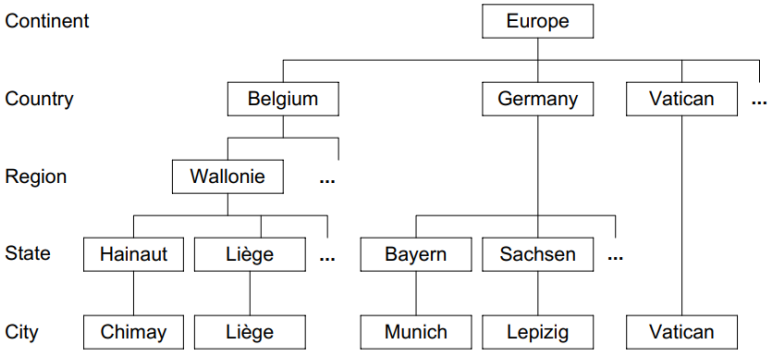


4.2.3 Generalized Hierarchies

- **Generalized hierarchies** biểu diễn case khi các members của một level có các types khác nhau. VD customers có thể là companies hoặc persons. Ngoài ra, các



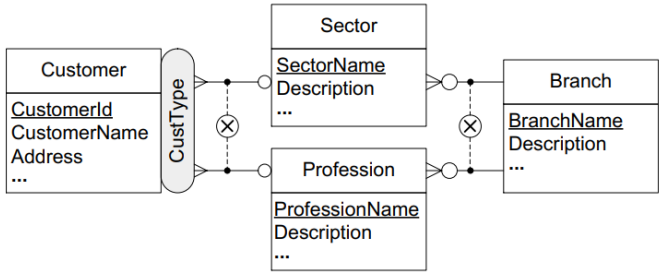
- Genralized hierarchies có 1 TH đặc biệt là **ragged hierarchies.** VD là Geography hierarchy.



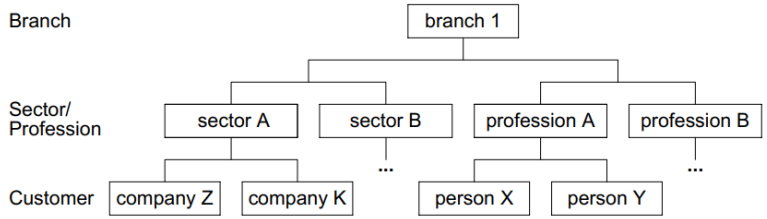
4.2.4 Alternative Hierarchies

- Alternative hierarchies biểu diễn case trong đó tại schema level một số nonexclusive hierarchies chia sẻ ít nhất là leaf level. VD, Date dimension gồm hai hierarchies là hai các groupings of months khác nhau thành calendar years và fiscal years.

measures với customers phải đc aggregated khác nhau theo customer type, với companies thì là Customer→Sector→Branch, với persons thì là Customer→Profession→Branch.

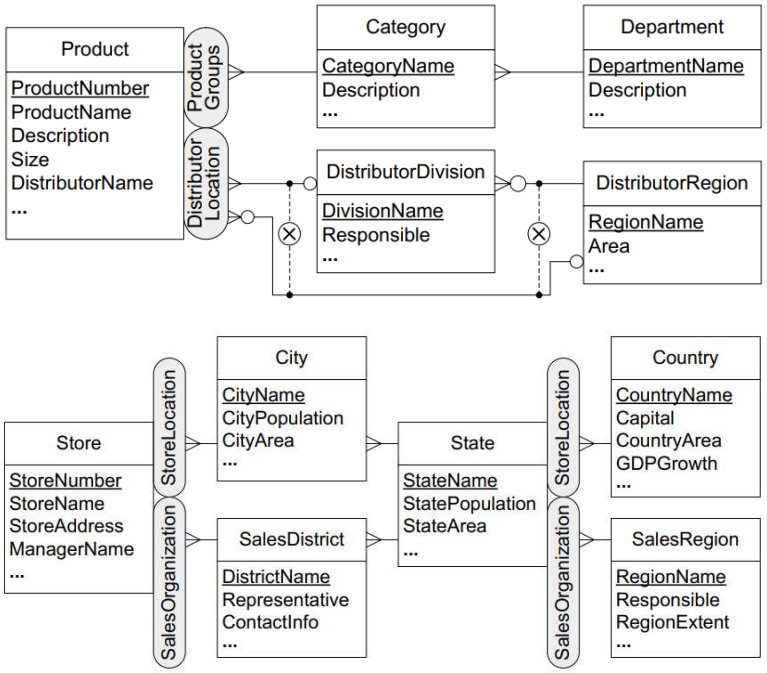


(a)

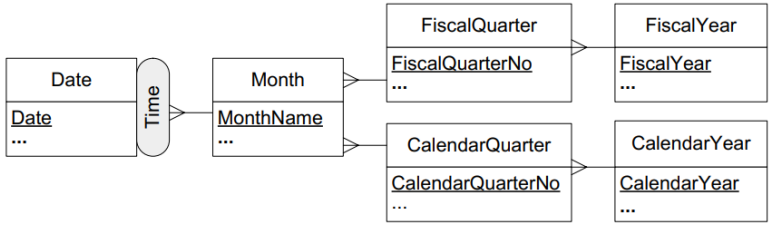


(b)

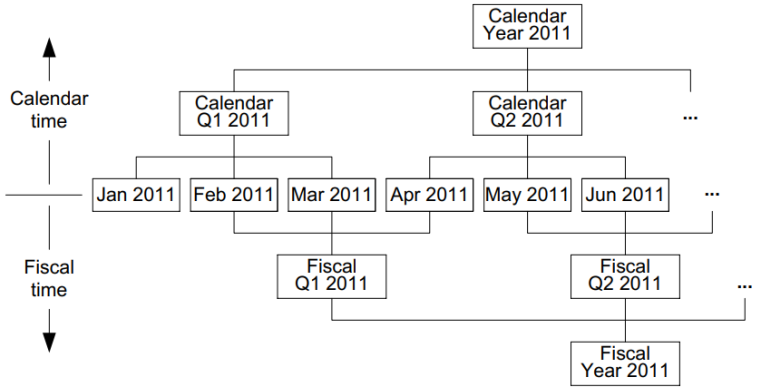
- Trong generalized hierarchies, splitting levels ko nhất thiết phải đc joined. VD có 3 loại publications: journals, books, conference proceedings. Chỉ có loại cuối đc aggragated thành conference level.



- Trong parallel dependent hierarchies, một leaf member có thể đc related tới nhiều members trong một shared level, mà ko phải là case alternative hierarchies mà share levels. VD, đi theo các hierarchies Lives và Territory từ Employee đến State level sẽ dẫn tới các states khác nhau cho các employees sống tại một state nhưng đc gán cho state khác. KQ là, aggregated measure values có thể đc dùng lại cho shared levels trong alternative hierarchies, trong khi điều này ko đúng với parallel dependent hierarchies.



(a)



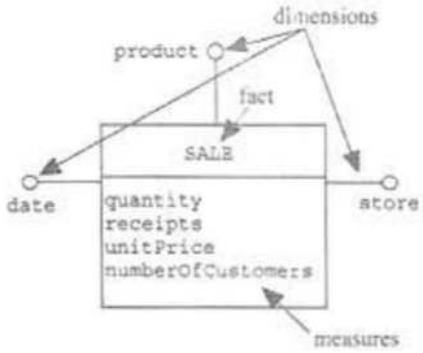
(b)

- Phân biệt: Trong generalized hierchy, một child member chỉ tham gia vào một path, còn trong alternative hierarchy một child member tham gia vào tất cả paths, và user phải chọn một cái để phân tích.

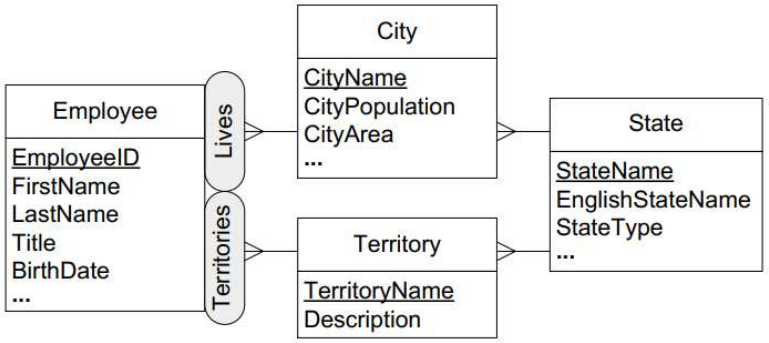
4.2.5 Parallel Hierarchies

- Khi một dimension có nhiều hierarchies, ta sẽ có **Parallel hierarchies.** Các parallel hierarchies có thể là dependent hoặc independent tùy vào component hierarchies có chia sẻ levels ko.

- Một **measure** là một numerical property of a fact và mô tả một quantitative fact aspect liên quan tới analysis.
  VD: Mỗi sale đc measured bởi số lượng units đc bán, unit price, total receipts. Lí do các measures phải là numeric là vì chúng đc dùng cho calculations.
- Một **dimension** là một fact property với một finite domain và mô tả một analysis coordinate của fact.
  VD: sales fact có các dimensions là products, stores, dates.
- Một **primary event** là một particular occurrence of a fact, đc xđ bởi một n tạo nên một giá trị cho mỗi dimension. Một giá trị cho mỗi measure đc đi kèm với mỗi primary event.
  VD: Một primary event ghi nhận có 10 gói bột giặt Shiny đc bán với giá $25 vào ngày 10/10/2008 tại SmartMart store.
- VD một fact schma cho sales:

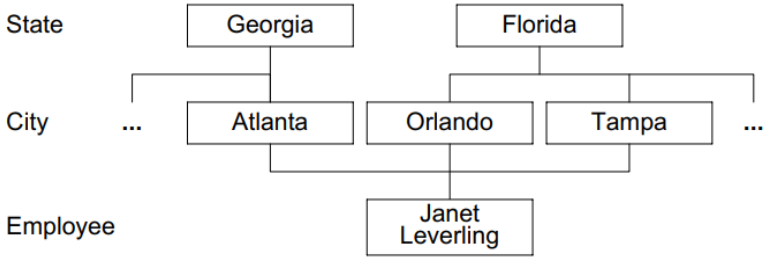


- **Dimension attributes** chỉ dimensions và các attributes có thể có, luôn đi cùng với các giá trị cụ thể mô tả chúng.
  VD: Một product đc mô tả bởi type, category, brand, department; thì product, type, category, brand, department sẽ là dimensional attributes. Quan hệ giữa các dimensional attributes đc mô tả bởi herarchies
- Một **hierarchy** là một directed tree có các nodes là dimensional attributes và các arcs sẽ model các many-to-one associations giữa các cặp dimensional attribute. Nó bao gồm một dimenion tại tree's root, và tất cả dimensional attributes mô tả nó.



4.2.6 Nonstrict Hierarchies

- Một hierarchy có ít nhất một many-to-many relationship đgl **nonstrict**, còn lại đgl **strict**.



DWDMPM 5.1 - The Dimensional Fact Model: Basic Concepts

- **Dimensional Fact Model (DFM)** là một conceptual model dùng để hỗ trợ data mart design
- Một **fact** là một concept liên quan đến decision-making processes. Nó models một tập các sự kiện trong một công ty.
  VD: sales, shipments, purchases, complaints.

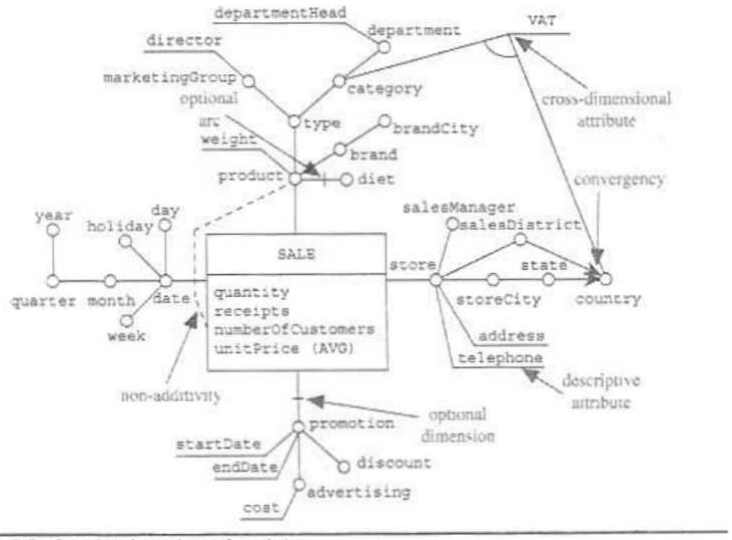
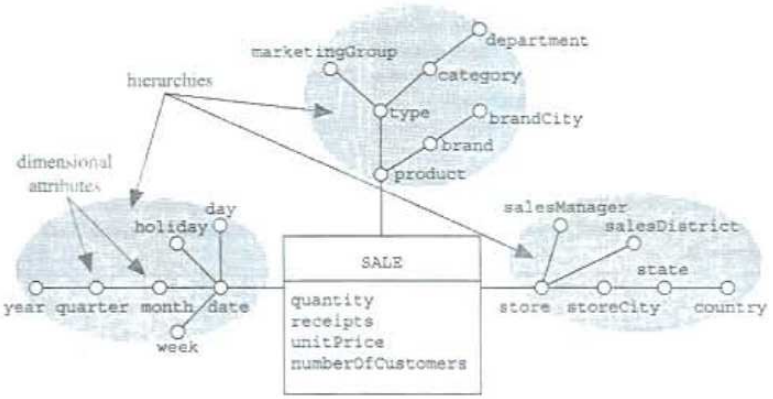


Figure 5-8 Complete fact schema for sales

5.2.2 Cross-Dimensional Attributes

- Một **cross-dimensional attribute** là một dimensional hoặc descriptive attribute có giá trị đc xđ bằng kết hợp của hai hay nhiều dimensional attributes, có thể thuộc về các hierarchies khác nhau.
  VD ở hình 5.8, VAT của một product phụ thuộc vào category và country mà product đc bán.
- VD: Area và species attributes cùng xđ residentPopulation attribute mà ước lượng số lượng cá thể mỗi animal species sống trong mỗi area.

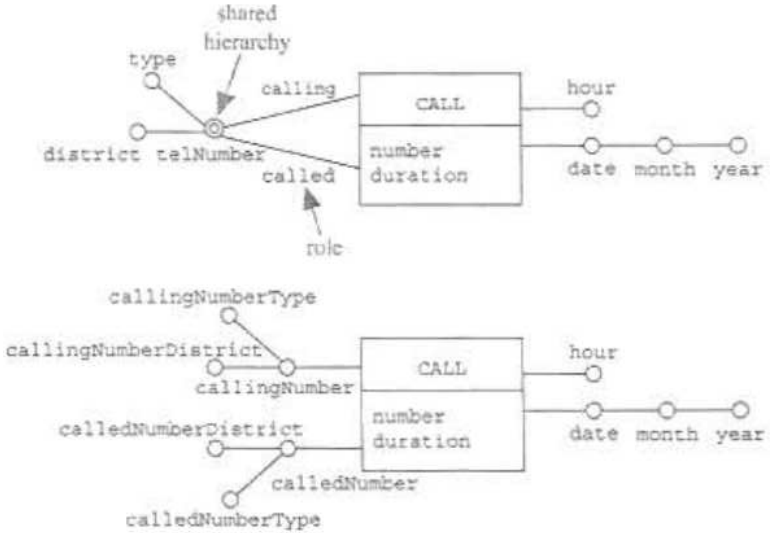
- VD hierarchies đc xây dựng trên các dimensions:



DWDMPM 5.2 - Advanced Modeling

5.2.1 Descriptive Attributes

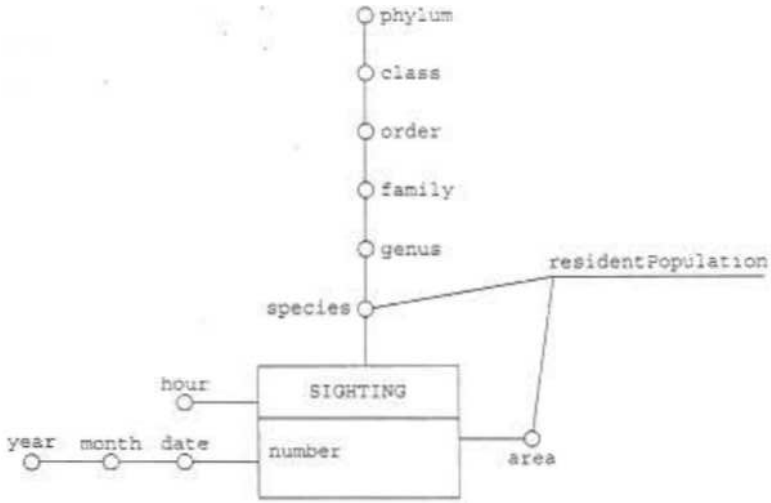
- Một **descriptive attribute** đc xđ bởi một dimensional attribute và mô tả một property của attribute này. Các descriptive attributes thường đi kèm với dimensional attributes bằng one-to-one associations và ko thực sự thêm levels of aggregation.
  VD: store's address, telephone number, product weight.
- Các descriptive attributes luôn là leaves of hierarchies, đc mô tả bằng horizontal lines trong DFM. VD:



- Hình phía trên là fact về telephone calls bao gồm number calling, number called, date, hour of call là dimensions. Double circle thể hiện và nhấn mạnh first attribute đc shared. Tất cả descendants của shared attributes cũng luôn đc shared. Nếu một hoặc nhiều descendants ko đc shared, chúng nên đc biểu diễn riêng.
  Hình phía dưới cũng là fact này nhưng ko dùng shared hierarchies.
- Khi hierarchies đc shared từ dimensions của chúng, ta cần thêm một **role** thể hiện significance to each incoming arc (calling và called).

5.2.5 Multiple Arcs

- Một multuple arc từ attribute a đến attribute b là một many-to-many association giữa a và b. VD, ta muốn aggregate và chọn sales dựa trên book authors. Tuy nhiên, sẽ ko chính xác nếu model author là một dimensional child attribute of book vì nhiều authors có thể viết nhiều books.

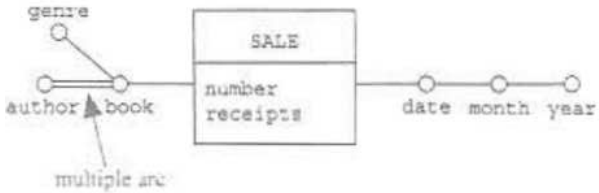


5.2.3 Convergence

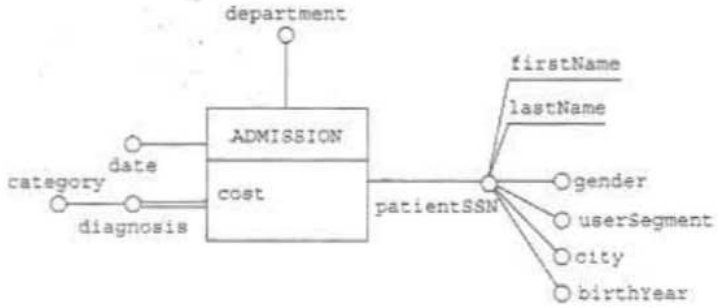
- Hai hay nhiều arcs thuộc về cùng một hierarchy và kết thúc tại cùng một dimensional attribute đánh dấu **convergences**.
  VD: stores có thể đc nhóm thành cities, từ đó nhóm thành states thuộc về countries. Stores cũng có thể đc nhóm thành sales districts và ko có inclusive relationship giữa districts và states. Giả sử thêm mỗi district chỉ thuộc về đúng một country, cũng là country mà store thuộc về. Như vậy, mỗi store chỉ thuộc về một country, độc lập với path sau:
  store → storeCity → state → country, hay
  store → salesDistrict → country

5.2.4 Shared Hierarchies

- Ta có một notation để nhấn mạnh việc sharing hierarchies. VD:



- Multiple arc có thể nối với một dimension thay vì dimensional attribute. VD, một fact là hostipal admissions có các dimensions là admission dates, admitted patient identifications, deparments nhận admission. Ta cần aggregate và chọn admissions dựa trên diagnoses đc cấp. Nhưng nhiều diagnoses thường tương ứng với một single admission, mỗi cái thuộc về một category. Hình dưới cho thấy ta có thể chuyển start arc trong diagnosis dimension thành một multiple arc.



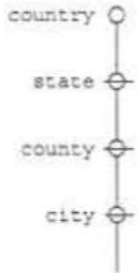
5.2.6 Optional Arcs

- **Optionality** đc dùng để model các scenarios trong đó một association trong một fact schema ko đc xđ cho một subset of events.
- Có 2 loại optional arcs r:
  1. r xđ một dimensional attribute a. Nếu b là dimensional attribute mà xđ a thông qua r, optimality nghĩa là a và tất cả descendants của nó có thể là undefined đối với một hoặc nhiều giá trị của b
     VD: Trên hình 5.8, diet attribute đc nhận giá trị (sugar-free, etc) chỉ đối với food, còn lại là null.

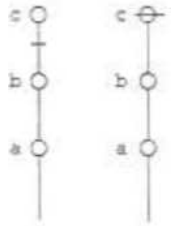
  2. r xđ một dimension d là optional nếu một số primary events đc xđ chỉ bằng các dimensions khác.
     VD: Trên hình 5.8, giá trị của promotion dimension sẽ là “No promotion” đối với một số product-store-date combinations.

5.2.7 Incomplete Hierarchies

- Một **incomplete hierarchy** có một hoặc nhiều levels of aggregation ko tồn tại ở một số instances.



- Phân biệt incomplete hierarchies và optional arcs:
  - Trong incomplete hierarchies, một hoặc nhiều attribute values đối với các hierarchy instances nhất định ở bất cứ hierarchy positions sẽ ko tồn tại.
  - Trong optional arc, ta model the fact giá trị của một attribute và các giá trị của tất cả attribute descedants là ko tồn tại.
  - Nếu chỉ có end attribute là ko tồn tại, hai cách trên là như nhau (hình dưới).



Logical Design

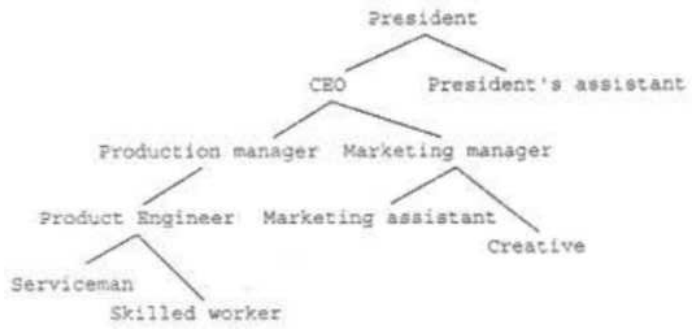
Course	INFOH419 - DW
Resource(s)	DWSDI 5.1, 5.2, 5.3, 5.5, 5.7

DWSDI 5.1 - Logical Modeling of Data Warehouses

- Có các cách khác nhau để triển khai một multidimensional model, phụ thuộc vào cách lưu data cube.
- **Relational OLAP (ROLAP)** systems: Lưu multidimensional data trong relational databases, bao gồm SQL features và special access methods để triển khai OLAP operations. Ngoài ra, để tăng performance, các aggreagates đc precomputed trong relational tables. Các aggregates này cùng với indexing structures sẽ chiếm KG lớn trong DB. Lợi ích của ROLAP systems là các relational DB đc well standardized và cung cấp dung lượng lưu trữ lớn. Tuy nhiên, vì các OLAP operations phải đc thực hiện trên relational tables, nó thường sinh ra các SQL queries phức tạp.
- **Multidimensional OLAP (MOLAP)** systems: Lưu data trong specialized multidimensional data structures (VD: arrays), đc KH với hashing và indexing. Do đó, OLAP operations có thể đc thực hiện efficiently, vì các operations là natural và đơn giản. Nhìn chung MOLAP systems cung cấp ít dung lượng lưu trữ hơn ROLAP. Ngoài ra, MOLAP là proprietary, giảm portability của chúng.
- **Hybrid OLAP (HOLAP)** systems: KH hai phương pháp trên để vừa có storage capacity của ROLAP vừa có processing capablities của MPLAP. VD, một HOLAP server có thể lưu lượng lớn data ở một relational DB, trong khí các aggregations dc giữ ở một MOLAP store riêng biệt.

5.2.8 Recursive Hierarchies

- Khác với incomplete hierarchies, các quan hệ parent-child giữa các levels trong **recursive** (hay **unbalanced**) **hierarchies** là consistent, nhưng các instances của chúng có thể có different lengths. VD:

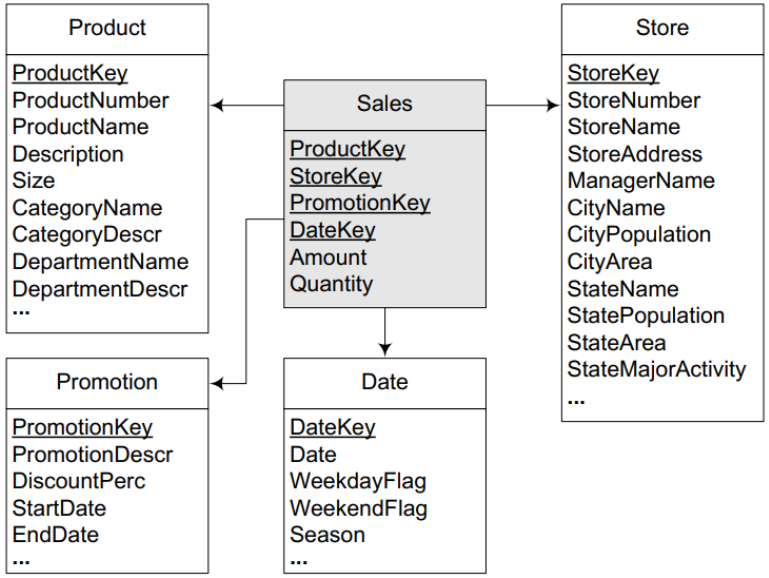


5.2.9 Additivity

- Một measure đgl **additive** dọc theo một dimension khi ta có thể dùng SUM operator để aggregate các giá trị của nó dọc theo dimension hierarchy. Nếu ko thì nó đgl **non-additive**. Một non-additive measure đgl **non-aggregable**.
  VD: Trên hình 5.8, meaure numberOfCustomers nối với dimension product là non-addtive.

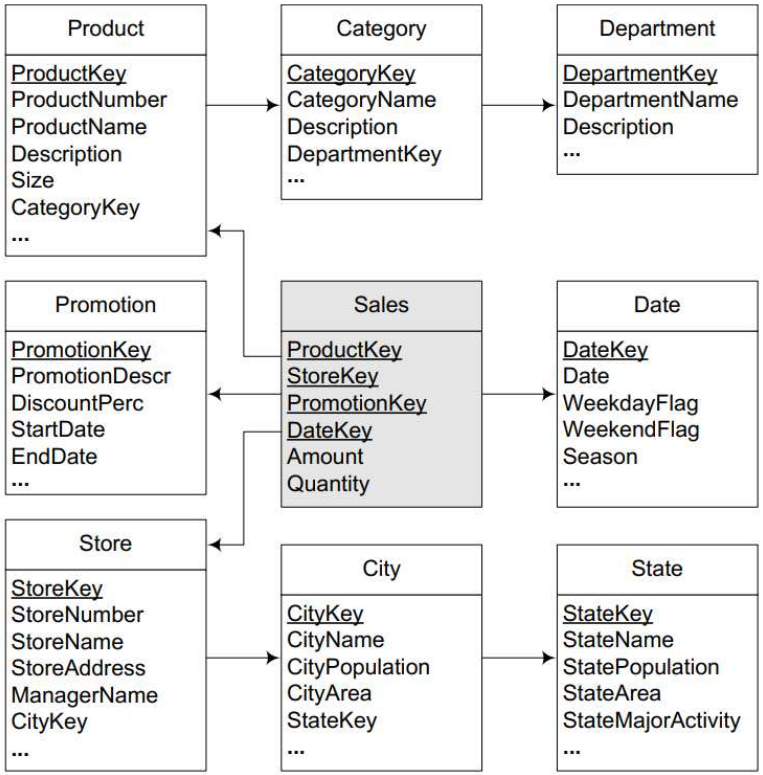
DWSDI 5.2 - Relational Data Warehouse Design

- Một relational representation của multidimensional model là **star schema**, trong đó có một **central fact table** và một tập **dimensional tables**.



- Trong một star schema, các dimension tables nhìn chung là not normalized, và có thể có redundant data. VD, trong dimension Product, các products thuộc về cùng một category sẽ có redundant info cho các attributes mô tả category và department. Điều tương tự trong dimension Store với các attributes mô tả city và state.
- Ngược lại, các fact tables thường là normalized. Key của chúng là union của các foreign keys, và ko có functional dependency giữa các foreign key attributes.
- Một **snowflake schema** tránh redundancy của star schemas bằng cách normalize dimension tables. Do đó, một dimension đc biểu diễn bởi một số tables đc related

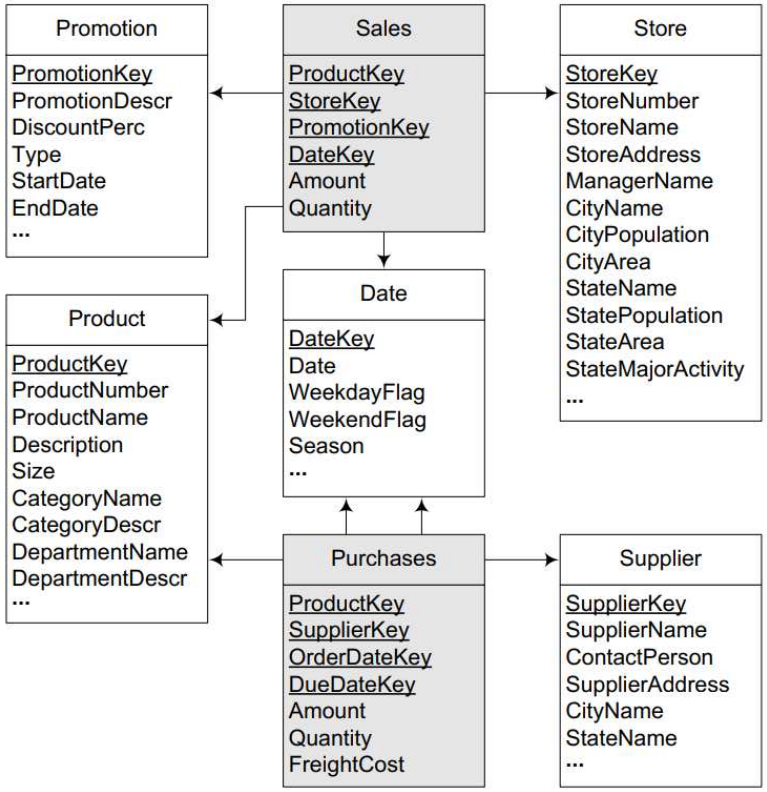
bởi referential **integrity constraints**.



- Các normalized tables dễ để duy trì và tối ưu KG lưu trữ. Tuy nhiên, performance bị hảnh hưởng vì cần thực hiện nhiều joins khi thực hiện queries.
- Một **starflake schema** là KH của star và snowflake schemas, trong đó một số dimensions đc normalized còn lại thì ko.
- Một **constellation schema** có nhiều fact tables mà chia sẻ dimension tables. Constellation schemas có thể có cả normalized và unnormalized dimension tables.

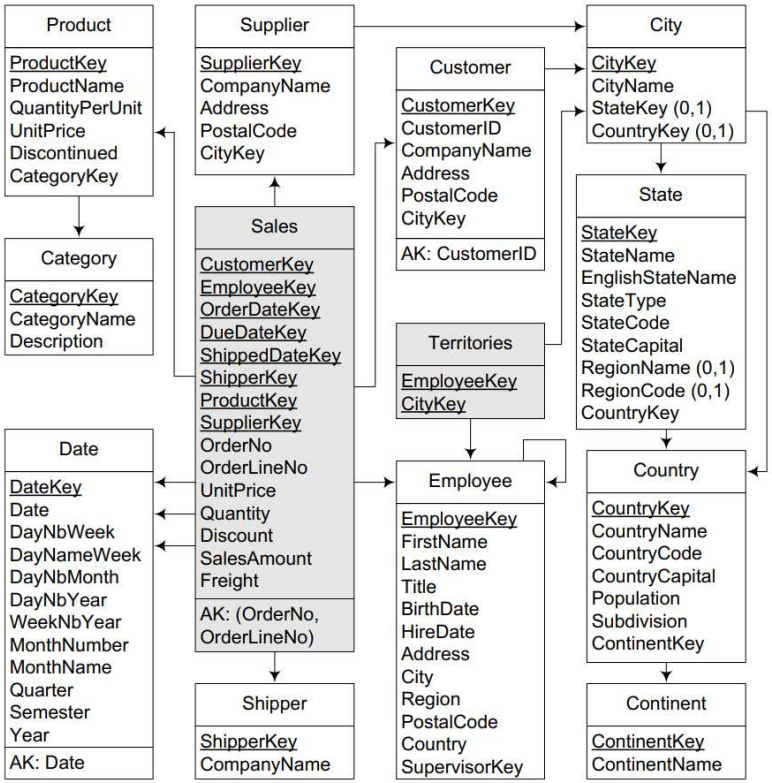
thể đc thêm vào table, nếu ko thì identifier của level sẽ là key của table. Additional attributes sẽ đc thêm vào table khi mapping relationships theo Rule 3.

- **Rule 2:** Một fact F đc mapped thành một table T_F có tất cả measures của fact làm attributes. Một surrogate key có thể đc thêm vào table. Additional attributes sẽ đc thêm vào table khi mapping relationships theo Rule 3.
- **Rule 3:** Một relationship giữa một fact F và một dimension level L, hoặc giữa các dimension levels L_P và L_C (parent và child), có thể đc mapped theo ba cách tùy theo cardinalities:
  - **Rule 3a:** Nếu relationship là one-to-one, table tương ứng với fact (T_F) hoặc child level (T_C) đc mở rộng với tất cả attributes của lần lượt dimension level hoặc parent level.
  - **Rule 3b:** Nếu relationship là one-to-many, table tương ứng với fact (T_F) hoặc child level (T_C) đc mở rộng với surrogate key của lần lượt dimension level hoặc parent level; nghĩa là, có một foreign key trong fact hoặc child table chỉ tới table còn lại.
  - **Rule 3c:** Nếu relationship là many-to-many, một new table T_B (bridge) đc tạo ra có attributes là tất cả surrogate keys của tables tương ứng với fact (T_F) và dimension level (T_L), hoặc với parent (T_P) và child levels (T_C). Nếu relationship có một distributing attribute, một additional attribute đc thêm vào table để lưu thông tin này.
- Áp dụng các rules này vào conceptual data cube:

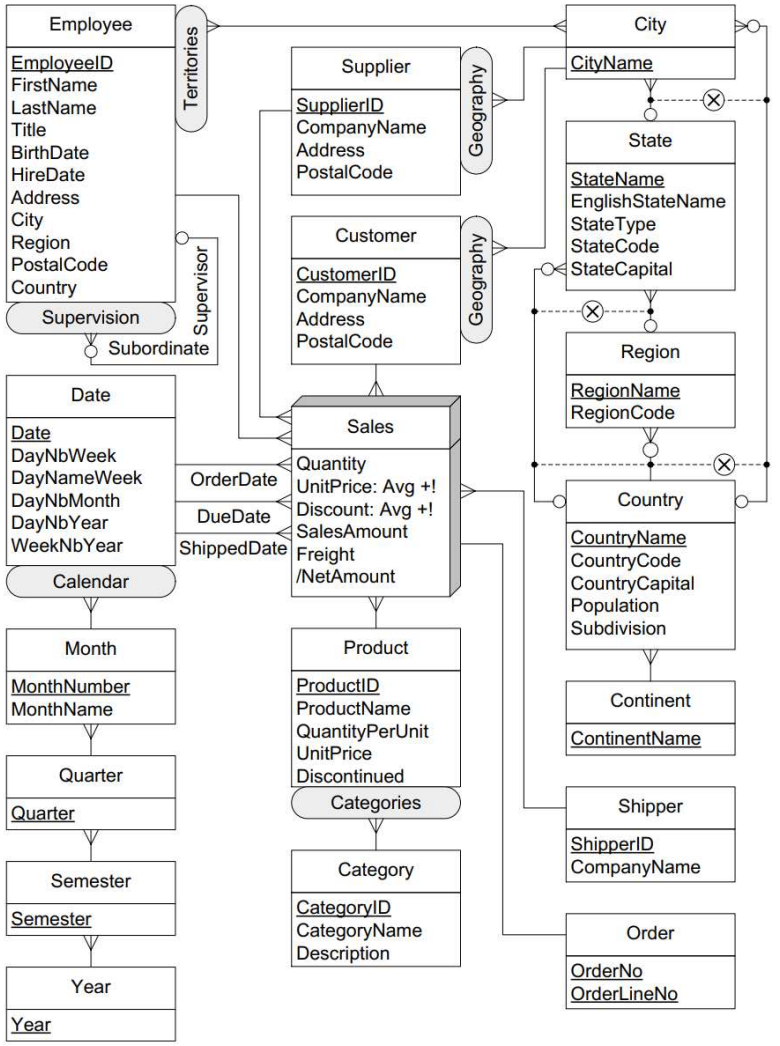


DWSDI 5.3 - Relational Representation of Data Warehouses

- Ta define một tập rules để translate một multidimensional conceptual schema sang một relational schema
- **Rule 1:** Một level L, khi ko related tới một fact với một one-to-one relationship, sẽ đc mapped thành một table T_L có tất cả attributes của level. Một surrogate key có



- Sales talbe có tám foreign keys, mỗi cái cho một level related tới fact bằng một one-to-many relationship**.** Date là role-playing dimensions, mỗi role đc biểu diễn bởi một foreign key**.** Dimension Order đc related tới fact bằng một one-to-one relationship, do đó các attributes của dimension đc đưa hết vào fact table**.** Dimension như vậy đgl một **fact** (hay **degenerate**) **dimension.** Many-to-many parent-child relationship giữa Employee và Territory đc mapped thành table Territories, có hai foreign keys**.**

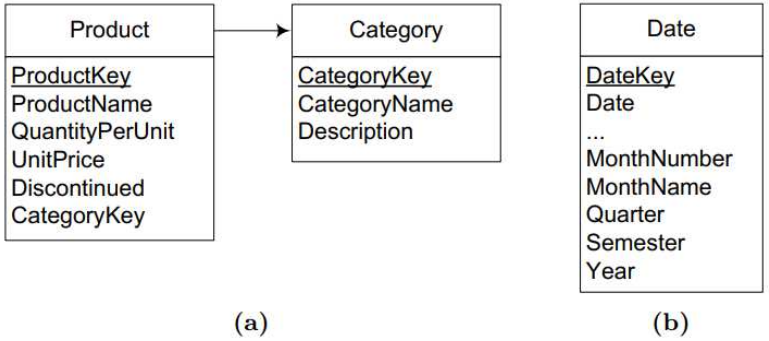


sẽ cho ra các tables:

DWSDI 5.5. Logical Representation of Hierarchies

5.5.1 Balanced Hierarchies

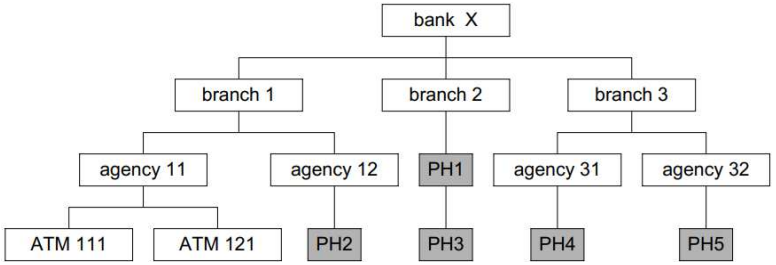
- Áp dụng các rules ở 5.3 vào balanced hierarchies sẽ cho ra snowflake schemas. VD áp dụng rules 1 và 3b vào Categories hierarchy sẽ cho ra một snowflake structure với tables Product và Category (hình a).



- Tuy nhiên, nếu star schems đc y/c, thì ta biểu diễn hierarchies bằng flat tables, trong đó key và attributes của tất cả levels đc đưa vào table (hình b).

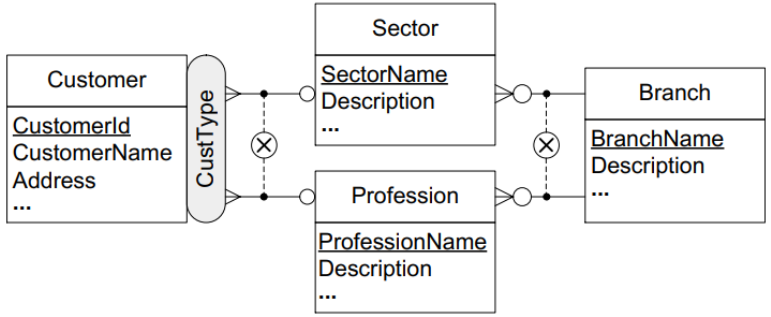
5.5.2 Unbalanced Hierarchies

- Một unbalanced hierarchy đc transformed thành một balanced one bằng placeholders. Sau đó, áp dụng logical mapping cho balanced hierarchies.

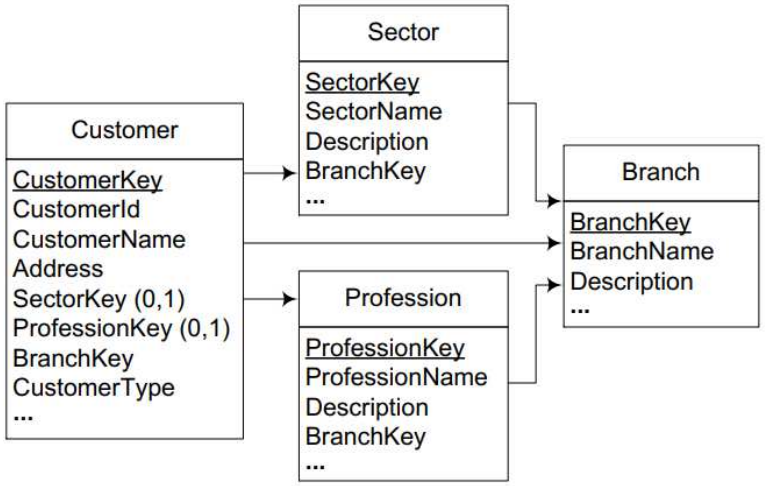


5.5.3 Generalized hierarchies

- Áp dụng mapped ở 5.3 vào hierarchy:

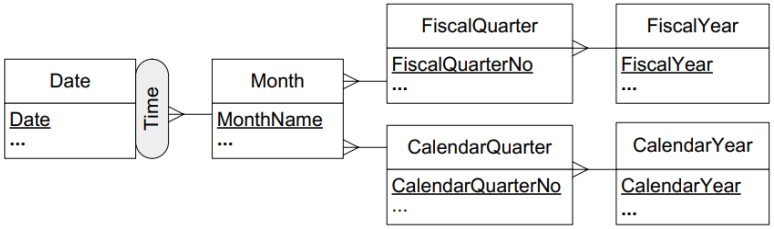


sẽ cho ra relations:

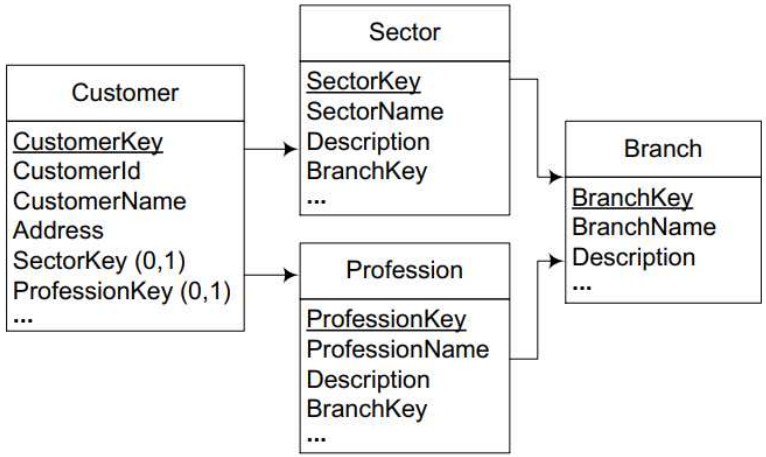


5.5.4 Alternative Hierarchies

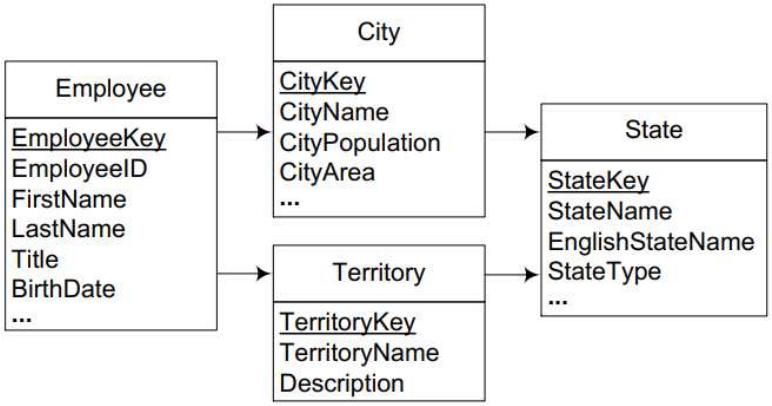
- Có thể áp dụng traditional mapping to relation tabkes. VD với conceptual schema:



sẽ cho ra:



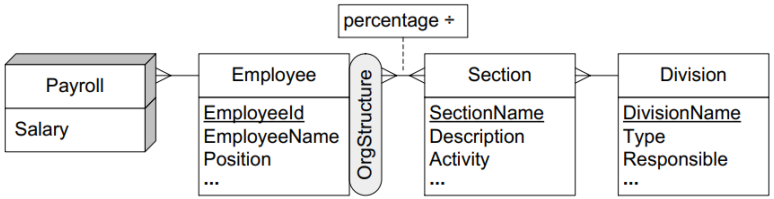
- Mặc dù schema này biểu diễn rõ ràng hierarchical structure, nó ko cho phép chỉ đi trên các common levels (VD đi thẳng từ Customer đến Branch). Để có khả năng này, ta thêm mapping rule.
- **Rule 4:** Một table tương ứng với một splitting level trong một generalized hierarchy có một additional attribute (BranchKey) là một foreign key of next joining level, với việc nó tồn tại. Table cũng có thể có một discriminating attribute (CustomerType) xđ specific aggregation path của mỗi member.



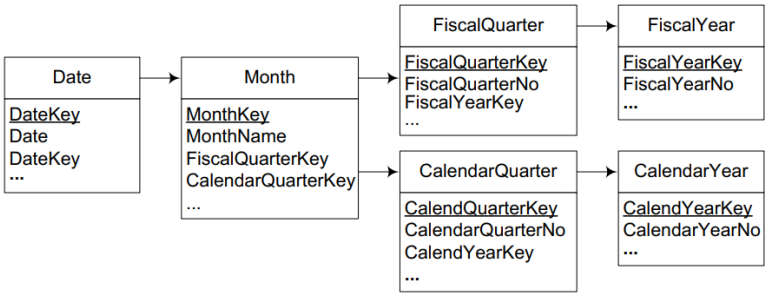
- Chú ý là mặc dù alternative và parallel dependent hierarchies có thể đc phân biệt dễ dàng ở conceptual level, chúng trông tương đồng ở logical level, mặc dù có một số characteristics phân biệt chúng.

5.5.6 Nonstrict Hierarchies

- Áp dụng các mapping rules ở 5.3 vào nonstrict hierarchies sẽ tạo ra relations cho các levels và một additional relation, đgl bridge table, cho many-to-many relationship giữa chúng. VD áp dụng với conceptual hierarchy:



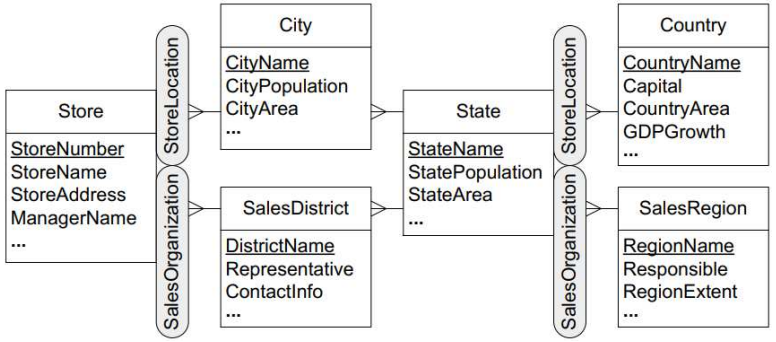
sẽ cho ra:



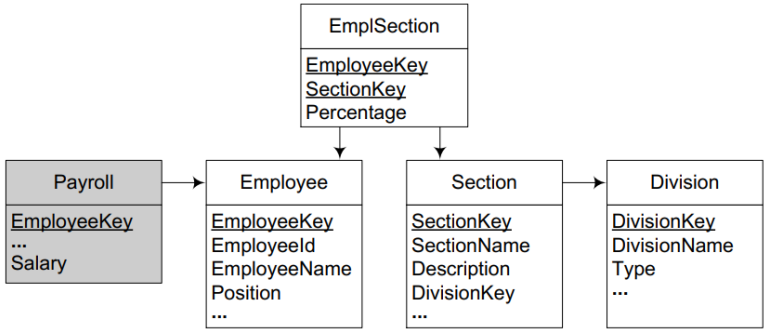
- Chú ý là mặc dù generalized và alternative hierarchies có thể đc phân biệt dễ dàng ở conceptual level, ta ko thể phân biệt chúng ở logical level.

5.5.5 Parallel Hierarchies

- Vì parallel hierarchies gồm nhiều hierarchies, logical mapping của chúng KH các mappings cho specific types of hierarchies. VD áp dụng với schema:



sẽ cho ra:



- Ở đây, bridge table EmplSection biểu diễn many-to-many relationship. Nếu parent-child relationship có một distributing attribute, nó sẽ đc biểu diễn trong bridge table như một additional attribute.

DWSDI 5.7 - Slowly Changing Dimensions

- Xét một Sales fact related tới chỉ các dimensions Date, Employee, Customer, Product, và một SalesAmount measure. Ta giả sử một star (denormalized) representation của table Product, và category date đc đưa vào table này.

DateKey	EmployeeKey	CustomerKey	ProductKey	SalesAmount
t1	e1	c1	p1	100
t2	e2	c2	p1	100
t3	e1	c3	p3	100
t4	e2	c4	p4	100

ProductKey	ProductName	UnitPrice	CategoryName	Description
p1	prod1	10.00	cat1	desc1
p2	prod2	12.00	cat1	desc1
p3	prod3	13.50	cat2	desc2
p4	prod4	15.00	cat2	desc2

- Các new tuples sẽ đi vào Sales fact table khi có new sales, và new tuples cũng có thể đi vào Product table khi có new product. Ngoài ra, data về một product cũng có thể sai, và tuples tương ứng phải đc sửa lại. Cuối cùng, category của một product có thể thay đổi. Giả sử các thay đổi này là ko thường xuyên, khi các dimensions đc thiết kế để hỗ trợ, chúng đgl **slowly changing dimensions**.
- Xét một query lấy total sales per employee and product category:

```
SELECT E.EmployeeKey, P.CategoryName, SUM(SalesAmount)
FROM Sales S, Product P
WHERE S.ProductKey = P.ProductKey
GROUP BY E.EmployeeKey, P.CategoryName
```

KQ là:

EmployeeKey	CategoryName	SalesAmount
e1	cat1	100
e2	cat1	100
e1	cat2	100
e2	cat2	100

- Giả sử sau last sale trên table, category của product prod1 chuyển thành cat2. Giải pháp đơn giản là cập nhật category của product thành cat2 sẽ ko giữ đc category trước đó. KQ là, nếu thực hiện lại query trên, ta mong muốn sẽ có cùng KQ như trên, nhưng vì tất cả sales diễn ra trước reclassification, ta sẽ có KQ:

EmployeeKey	CategoryName	SalesAmount
e1	cat2	200
e2	cat2	200

KQ này là sai. Ngược lại, nếu new category là KQ của một error correction (nghĩa là category đúng của prod1 là cat2), thì KQ trên là đúng. Trong case trước, để có đc

KQ đúng y/c lưu lại KQ khi prod1 có category cat2, và chắc chắn rằng new aggregations sẽ đc tính với new category cat2.

- Có 3 cánh để xử lí slowly changing dimensions. Cách 1, đgl **type 1**, sẽ overwrite old value của attribute bằng new value, nghĩa là ta mất lịch sử của attribute. Cách này phù hợp khi việc chỉnh sử là do lỗi trong dimension data.
- Trong cách 2, đgl **type 2**, các tuples trong dimension table đc versioned, và một new tuple đc đưa vào mỗi khi có một sự thay đổi. VD, ta sẽ thêm một row cho product p1 trong Product table với new category cat2 sao cho tất cả sales trước t sẽ đóng góp vào aggregation cat1, còn các sales sau t sẽ đóng góp vào cat2. Cách này y/c có một surrogate key trong dimension bên cạnh business key để mà tất cả versions của một dimension members có surrogate key khác nhau nhưng có cùng business key. Ngoài ra, ta cũng cần mở rộng table với hai attributes xđ validity interval của tuple.

Product Key	Product ID	Product Name	UnitPrice	Category Name	Description	From	To
k1	p1	prod1	10.00	cat1	desc1	2010-01-01	2011-12-31
k11	p1	prod1	10.00	**cat2**	desc2	2012-01-01	9999-12-31
k2	p2	prod2	12.00	cat1	desc1	2010-01-01	9999-12-31
...	...	...	...	...	...	...	...

- Trong type 2 approach, đôi khi một additional attribute đc thêm vào để chỉ rõ current row.

Product Key	Product ID	Product Name	Unit Price	Category Name	Description	From	To	Row Status
k1	p1	prod1	10.00	cat1	desc1	2010-01-01	2011-12-31	Expired
k11	p1	prod1	10.00	**cat2**	desc2	2012-01-01	9999-12-31	Current
...	...	...	...	...	...	...	...	...

- Giờ xét một snowflake representation cho Product dimension, trong đó categories đc biểu diễn trong một table Category.

Product Key	Product Name	Unit Price	Category Key
p1	prod1	10.00	c1
p2	prod2	12.00	c1
p3	prod3	13.50	c2
p4	prod4	15.00	c2

Category Key	Category Name	Description
c1	cat1	desc1
c2	cat2	desc2
c3	cat3	desc3
c4	cat4	desc4

- Type 2 approach cho snowflake representation đc xử lí tương tự như trên, nhưng giờ một surrogate key và hai temporal attributes From và To phải đc thêm vào cả Product và Category tables. Vẫn giả sử product prod1 đổi category sang cat2. Theo type 2, Product table sẽ là:

Product Key	Product ID	Product Name	Unit Price	Category Key	From	To
k1	p1	prod1	10.00	l1	2010-01-01	2011-12-31
k11	p1	prod1	10.00	**l2**	2012-01-01	9999-12-31
k2	p2	prod2	12.00	l1	2010-01-01	9999-12-31
k3	p3	prod3	13.50	l2	2010-01-01	9999-12-31
k4	p4	prod4	15.00	l2	2011-01-01	9999-12-31

và Category table ko đổi. Tuy nhiên, nếu thay đổi diễn ra ở upper level trong hierarchy, nó cần đc truyền xuống dưới. VD, giả sử description của category cat1 thay đổi như sau:

Category Key	Category ID	Category Name	Description	From	To
l1	c1	cat1	desc1	2010-01-01	2011-12-31
l11	c1	cat1	**desc11**	2012-01-01	9999-12-31
l2	c2	cat2	desc2	2010-01-01	9999-12-31
l3	c3	cat3	desc3	2010-01-01	9999-12-31
l4	c4	cat4	desc4	2010-01-01	9999-12-31

Physical Datawarehouse Design

Course	INFOH419 - DW
Resource(s)	DWSDI 8.1, 8.2, 8.4, 8.5, 8.6

DWSDI 8.1 - Physical Modeling of Datawarehouses

- Một **view** trong relational model chỉ là một query đc lưu trong DB với một associated name, và có thể đc dùng như một normal table.
- Một **materialized view** là một view đc physically stored trrong một DB. Materialized views tăng query performance bằng cách tính trước các costly operations như joins và aggregations và lưu KQ vào DB. Đương nhiên, performance tăng bị đánh đổi bởi storage space.
- Một nhược điểm của materialized view là nó y/c phải tính trước queries đc materialized. Tuy nhiên, DW queries là ko thể tính trước, do đó các indexing methods đc dùng để đảm bảo xử lí query hiệu quả. Các kĩ thuật indexing truyền thống cho OLTP ko phù hợp với multidimensional data, nên DW cần các mechanismss khác.
- **Bitmap indexes** là một loại index đặc biệt đc dùng cho columns có ít distince values (low cardinality attributes).
- **Join indexes** sẽ materialize một relational join giữa hai tables bằng cách giữ các cặp row identifiers tham gia vào join. Trong DW, join indexes sẽ relate values of dimensions tới rows trong fact table.
- **Partitioning** là một cơ chế đc dùng trong relational DB để tăng hiệu quả queries. Nó chia contents của một relation thành một số files mà có thể đc xử lí hiệu quả. VD, một table có thể đc chia sao cho các attributes thường đc dùng đc lưu trong

Thay đổi này cần đc truyền sang Product table để cho tất cả sales trước khi thay đổi chỉ tới old version của category cat1 (với key l1), còn new sales phải chỉ đến new version (với key l11), như sau:

Product Key	Product ID	Product Name	Unit Price	Category Key	From	To
k1	p1	prod1	10.00	l1	2010-01-01	2011-12-31
k11	p1	prod1	10.00	**l11**	2012-01-01	9999-12-31
k2	p2	prod2	12.00	l1	2010-01-01	2011-12-31
k21	p2	prod2	12.00	**l11**	2012-01-01	9999-12-31
k3	p3	prod3	13.50	l2	2010-01-01	9999-12-31
k4	p4	prod4	15.00	l2	2011-01-01	9999-12-31

- Cách 3, đgl **type 3**, sẽ đưa ra một additional column cho mỗi attribute có khả năng thay đổi, và sẽ lưu new value của attribute. Hình dưới minh họa cách này:

Product Key	Product Name	UnitPrice	Category Name	New Category	Description	New Description
p1	prod1	10.00	**cat1**	**cat2**	**desc1**	**desc2**
p2	prod2	12.00	cat1		desc1	
p3	prod3	13.50	cat2		desc2	
p4	prod4	15.00	cat2		desc2	

Lưu ý là chỉ hai last versions của attribute đc biểu diễn và validity interval của tuples ko đc lưu.

- Ta có thể áp dụng 3 cách trên, hoặc KH chúng, vào cùng dimension. VD, ta có thể áp dụng correction (type 1), tuple versioning (type 2), attribute addition (type 3) vào các attributes khác nhau trong cùng dimension table.

sử các products duy nhất cho category food là p1 và p2 và các tuples đc thấy là các tuples duy nhất cho các products này.

ProductKey	CustomerKey	Quantity
p1	c1	20
p1	c2	100
p2	c2	50
...	...	...

(a)

CustomerKey	Count
c1	1
c2	2

(b)

CustomerKey	Count
c1	1
c2	1

(c)

- Giử sử ta xóa tuple (p1, c2, 100) từ Sales. Mặc dù c2 trong FoodCustomers đc derived từ tuple này, nó cũng có một alternative derivation qua (p2, c2, 50). Do đó, xóa (p1, c2, 100) ko xóa c2 khỏi view. Counting algorithm sẽ tính một relation $\Delta^{-}(\mathrm{FoodCustomers})$ chứa các tuples có thể đc derived từ (p1, c2, 100), do đó sẽ bị ảnh hưởng bởi việc xóa tuple này, và thêm -1 vào mỗi tuple. Trong VD này, $\Delta^{-}(\mathrm{FoodCustomers})$ sẽ chứa các tuples {(c2, -1)}. Tương tự, với insertions, $\Delta^{+}(\mathrm{FoodCustomers})$ mở rộng các tuples với 1. Updated view ở hình (c) có đc từ việc join $\Delta^{-}(\mathrm{FoodCustomers})$ với materialized view FoodCustomers bằng attribute CustomerKey, và trừ đi $\Delta^{-}(\mathrm{FoodCustomers}).\mathrm{Count}$ từ $\mathrm{FoodCustomer.Count}$. Nếu sau đó tuple (p2, c2, 50) bị xóa, c2 cũng sẽ bị xóa khỏi view.
- Giờ ta xét views đc xđ bởi một outer join. Xét hai relations Product(ProdID, ProdName, ShipID) và Shipper(ShipID, ShipName) ở hình (a) và (b). Một VD outer join là:

```
CREATE VIEW ProductShipper AS (
    SELECT P.ProdID, P.ProdName, S.ShipID, S.ShipName
    FROM Product P
    FULL OUTER JOIN Shipper S
    ON P.ShipID = S.ShipID )
```

View này ở hình (d). View maintenance đc xử lí bằng cách rewrite full outer join bằng left hoặc right outer joins, tùy vào việc ta xử lí updates ở left hay right table of full outer join. Sau đó, ta merge KQ với view.

một parition, còn lại đc lưu trong một partition khác. Một scheme khác là mỗi parition chưa data về một khoảng TG cụ thể.

DWSDI 8.2 - Materialized Views

- Khi base relations đc cập nhật, materialized views có từ chúng cũng phải đc cập nhật. Quá trình này đgl **view maintenance.** Dưới một số conditions, ta có thể tính các sự thay đổi của views gây ra bởi các sự thay đổi trong underlying relations mà ko cần tính lại toàn bộ view từ đầu. Đây đgl **incremental view maintenance.**
- Có hai algorithms cho view maintenance tùy theo loại update và operations trong view definition;
  - Algorithms using full information: nghĩa là dùng cả views và base relations.
  - Algorithms using partial information: dùng materialized views và key constraints.

8.2.1 Algorithms Using Full Information

- Có ba loại viewss có thể đc giải quyết bằng algorithm này: nonrecursive views, outer-join views, và recursive views. Ở đây ta chỉ xét hai cái đầu.
- Algorithm cơ bản cho nonrecursive views (gồm join, union, negation, aggregation) là **counting algorithm.** Algorithm này đếm số lượng alternative derivations mà mỗi tuple trong view có. Theo cách này, nếu ta xóa một tuple trong một base relation, ta có thể kiểm tra xem nó nên xóa nó ở view ko. Xét relation FoodCustomer và view đc tạo ra như sau:

```
CREATE VIEW FoodCustomers AS (
    SELECT DISTINCT CustomerKey
    FROM Sales S, Product P
    WHERE S.ProductKey = P.ProductKey AND P.CategoryName = 'F
```

Một instance của relation Sales ở hình (a); view FoodCustomers trên relation này ở hình (b). Ta thêm một column Count xđ số lượng possible derivations cho mỗi tuple. VD, (c2, 2) nghĩa là customer c2 mua hai products từ category food. Ngoài ra, ta giả

nhật summary tables.

- Một attribute đgl **distinguished** trong một view V nếu nó xuất hiện trong SELECT clause of view definition. Một attribute A thuộc về một relation R đgl **exposed** trong một view V nếu A đc dùng trong một predicate trong V. Một số KQ của view maintenance theory là:
  - Một select-project-join view là not self-maintainable đối với insertions.
  - Mộ select-project-join view là self-maintainable đối với deletions trong một relation R nếu key attributes từ mỗi occurence of R in the join là included trong view hoặc equated một constant trong view definiton.
  - Một left hoặc full outer join view V đc defined từ hai relations R và S, sao cho keys của R và S là distinguished và tất cả exposed attributes của R là distinguished, là self-maintainable đối với tất cả các loại modificaitons trong S.
- Xét lại outer join view trong 8.2.1. View này TMĐK thứ ba và là self-maintainable đối với tất cả các loại modifications trong Product. Đầu tiên ta tính projection of view over Shipper:

$$\text{Proj\_Shipper} = \pi_{\text{ShipID, ShipName}}(\text{Product} \sqsupset\!\bowtie\!\sqsubset \text{Shipper}),$$

trên hình (a). Các tables $\Delta^+(\text{Product})$ và $\Delta^-(\text{Product})$ ở hình (b) và (c). Vì view là self-maintainable, ta có thể join các delta tables này với Proj_Shipper thay vì Shipper, tránh truy cập vào base relations. Các joins giữa delta tables và Proj_Shipper ở hình (d) và (e). Cuối cùng, KQ của hai joins đc merged với original view. KQ cuối cùng ở hình (f).


```
SELECT P.ProdID, P.ProdName, S.ShipID, S.ShipName
FROM Δ(Product) P
LEFT OUTER JOIN Shipper S
ON P.ShipID = S.ShipID

SELECT P.ProdID, P.ProdName, S.ShipID, S.ShipName
FROM Product P
RIGHT OUTER JOIN Δ(Shipper) S
ON P.ShipID = S.ShipID
```

- Query thứ nhất tính ảnh hưởng lên view từ changes to Product, và query thứ hai làm tương tự với changes to Shipper. Xét $\Delta^+(\text{Product})$ ở hình (c) chứa các tuples đc thêm vào Product. Khi ta thêm một matching tuple như (p3, MP3, s2), projection của left outer join với Shipper sẽ là (p3, MP3, s2, DHL). Trong case này, algorithm cũng xóa (NULL, NULL, s2, DHL) vì (s2, DHL) giờ đã có một matching tuple, đồng thời thêm (p3, MP3, s2, DHL). Với tuple (p4, PC, NULL) đc thêm vào Product, left outer join giữa (p4, PC, NULL) và Shipper sẽ cho ra (p4, PC, NULL, NULL) và đc thêm vào view. Hình (e) là final state của view.

ProdID	ProdName	ShipID
p1	TV	s1
p2	Tablet	NULL

(a)

ShipID	ShipName
s1	Fedex
s2	DHL

(b)

ProdID	ProdName	ShipID
p3	MP3	s2
p4	PC	NULL

(c)

ProdID	ProdName	ShipID	ShipName
p1	TV	s1	Fedex
p2	Tablet	NULL	NULL
NULL	NULL	s2	DHL

(d)

ProdID	ProdName	ShipID	ShipName
p1	TV	s1	Fedex
p2	Tablet	NULL	NULL
p3	MP3	s2	DHL
p4	PC	NULL	NULL

(e)

8.2.2 Algorithms Using Partial Information

- Một view đgl **self-maintainable** nếu nó có thể đc maintained chỉ dùng view và key constraints. Điều này là quan trọng cho DW vì ta ko mốn truy cập base data để cập

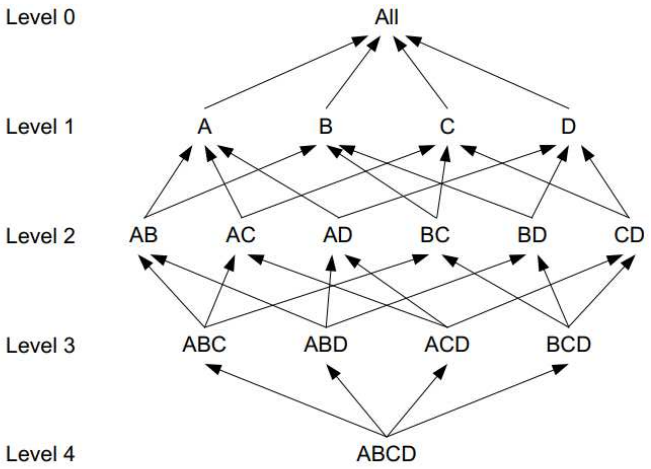


Fig. 8.6 A data cube lattice

- Các optimizations đơn giản nhất để tính cube lattice là;
  - Smallest parent: Tính mỗi view từ smallest previously computed one. VD, AB có thể đc tính từ ABC, ABD, hoặc ABCD. Cách này chọn cái nhỏ nhất.
  - Cache-results: Lưu vào cache một aggregation mà từ đó các aggregations khác có thể đc tính.
  - Amortize-scans: Tính trong memory nhiều aggregations nhất có thể, từ đó giảm số lượng table scans.
  - Share-sorts: Chỉ áp dụng với các ph/ph dựa trên sorting và nhắm tới sharing costs giữa một số aggregations.
  - Shart-partitions: Dành riêng cho algorithms dựa trên hashing.

8.4.1 PipeSort Algorithm

- **PipeSort algorithm** cung cấp một global strategy để tính data cube, bao gồm bốn optimization methods đầu tiên.

ShipID	ShipName
s1	Fedex
s2	DHL

(a)

ProdID	ProdName	ShipID
p3	MP3	s2
p4	PC	NULL

(b)

ProdID	ProdName	ShipID
p1	TV	s1

(c)

ProdID	ProdName	ShipID	ShipName
p3	MP3	s2	DHL
p4	PC	NULL	NULL

(d)

ProdID	ProdName	ShipID	ShipName
p1	TV	s1	Fedex

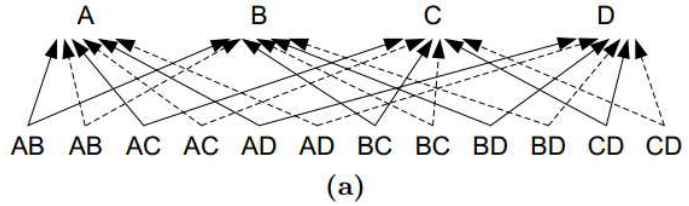
(e)

ProdID	ProdName	ShipID	ShipName
p2	Tablet	NULL	NULL
p3	MP3	s2	DHL
p4	PC	NULL	NULL
NULL	NULL	s1	Fedex

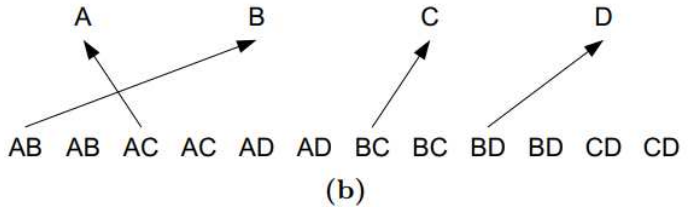
(f)

8.4. Computation of a Data Cube

- Đầu tiên ta define một **data cube latice.** Trong latice này, mỗi node biểu diễn một possible aggregation of fact data, trong đó một edge từ node i tới node j nếu $j$có thể đc tính toán từ i và số lượng grouping attributes of i là số lượng attributes của j cộng một. Hình 8.6 là một four-dimensional data cube, với các dimensions A, B, C, D. Trong lattice này, một edge từ ABC đến AB nghĩa là summary table AB có thể đc tính từ ABC.



(a)



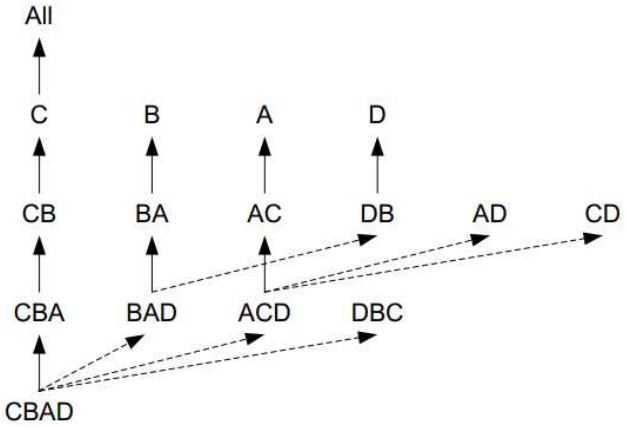
(b)

- Matching đc thực hiện N lần, với N là số lượng grouping attributes, từ đó tạo ra một evaluation plan. Heuristics là nếu với mọi cặp levels có cost là minimum, điều đó sẽ đúng trên toàn bộ plan. Output lattice sẽ cho một sorting order để tính mỗi node. KQ là, PipeSort algorithm đưa ra chiến lược sau: trong mỗi chain sao cho một node ở level k là một prefix của node ở level $k+1$ (trong output graph), tất cả aggregations có thể đc tính trong một pipeline.
- Genral scheme của PipeSort algorithm là:

- Input của algorithm là một data cube lattice trong đó mỗi edge e_{ij}, với node i là parent của node j, đc labeled với hai costs, $S(e_{ij})$ và $A(e_{ij})$. $S(e_{ij})$ là cost tính j từ i nếu i chưa đc sorted. $A(e_{ij})$ là cost tính j từ i nếu i đã đc sorted. Do đó $A(e_{ij}) \leq S(e_{ij})$. Ngoài ra, ta xét lattice đc tổ chức thành các levels, trong đó mỗi level k có đúng k attributes, bắt đầu từ All với $k = 0$. Data structure này đgl một **search lattice.**
- Output của algorithm là một subgraph của search lattice sao cho mỗi node có đúng một parent mà từ đó nó sẽ đc tính toán theo một certain mode, nghĩa là sorted or not. Nếu attribute order của node j là prefix of the order of its parent i, thì j có thể đc tính từ i mà ko cần sorting i, và trong graph KQ, edge sẽ có cost $A(e_{ij})$. Nếu ko, i cần đc sorted để tính j và edge sẽ có cost $S(e_{ij})$. Chú ý là với mọi node i trong output graph, sẽ chỉ có tối đa một outgoing edge marked A và nhiều outgoing edges marked S. M/đ của algorithm là tìm một output graph biểu diễn một execution plan sao cho tổng costs trên edges là nhỏ nhất.
- Để có minimum cost output graph, algorithms xử lí từng level một, bắt đầu từ 0 tới $N-1$, với N là số levels trong search latice. Ta tìm best way of computing nodes trong mỗi level k từ level $k+1$, reduce bài toán thành một **weighted bipartite matching** problem, như sau. Xét một cặp levels $(k, k+1)$. Đầu tiên algorithm sẽ transform level $k+1$ bằng cách tạo k copies của từng node của nó. Do đó, mỗi node tại level $k+1$ sẽ có $k+1$ children, nghĩa là $k+1$ outgoing edges. Tất cả original edges sẽ có cost $A(e_{ij})$ và tất cả replicated edges sẽ có cost $S(e_{ij})$. Cuối cùng, ta tính minimum cost matching sao cho mỗi node j tại level k sẽ đc matched với một node i tại level $k+1$. Nếu j đc kết nối tới i bằng một $A()$ edge, j sẽ xđ attribute order mà trong đó i sẽ đc sorted during its computation. Còn nếu j đc kết nối tới i bằng một $S()$ edge, i sẽ đc sorted để tính j.
- Xét VD hình dưới, graph đc xây dựng cho levels 1 và 2 của lattice trong hình 8.6. Các edges loại $A(e_{ij})$ là nét liền còn các edges loại $S(e_{ij})$ là nét đứt. Ở hình (a) ta đã thêm một copy của mỗi node ở level 2. Ở hình (b) ta thấy tất cả views sẽ đc tính với cost $A(e_{ij})$.

PipeSort Algorithm
INPUT: A search lattice with the $A()$ and $S()$ edges costs
OUTPUT: An evaluation plan to compute all nodes in the search lattice
BEGIN
    For level $k = 0$ to level $N - 1$
        Generate-Plan($k + 1 \rightarrow k$);
        For each node i in level $k + 1$
            Fix the sort order of i as the order of the level k node
            connected to i by an $A()$ edge;
    Generate-Plan($k + 1 \rightarrow k$);
    Create k additional copies of each level $k + 1$ node;
    Connect each copy node to the same set of level k nodes as the original node;
    Assign cost $A(e_{ij})$ to edges e_{ij} from the original nodes and cost $S(e_{ij})$
        to edges from the copy nodes;
    Find the minimum cost matching on the transformed level $k + 1$ with level k;
END

- Hình dưới là evaluation plan để tính cube lattice trong hình 8.6 bằng PipeSort algorithm. Đầu tiên minimum cost sort sẽ sort base fact table theo CBAD order và tính CBA, CB, C, và All aggregations theo pipelined fashion. Sau đó ta sort base fact table theo BADC rồi tính aggregates BAD, BA, B. Tương tự với ACDB, DBCA.



8.4.2 Cube Size Estimation

- Hầu hết các algorithms tính summary tables đều y/c biết trc size of each aggregate. Tuy nhiên, nhìn chung nó là ko biết trc. Do đó, ta cần dự đoán chính xác sizes của các aggregates khác nhau. Có ba cách truyền thống để làm điều này.
- **Analytical algorithm:** Chọn r elements (giả sử là các tuples trong một relation) ngẫu nhiên từ một tập n elements (là tất cả giá trị khác nhau mà một tập các attributes có thể có), thì expected number of distinct elements có đc sẽ là $n - n \times (1 - 1/n)^r$. Việc này giả sử data là uniformly distributed. VD ta có một relation R(ProductKey, CustomerKey, DateKey). Nếu ta muốn ước lượng size of aggregation over ProductKey and CustomerKey, ta cần biết số lượng các giá trị khác nhau của mỗi attributes. Theo đó $n = |\text{ProductKey}| \times |\text{CustomerKey}|$, và r là số lượng tuples trong R. Cách này đơn giản và hiệu quả, nhưng chỉ dùng đc khi ta biết data là uniformly distributed.
- **Sampling-based algorithm:** Lấy một random subset của DB và tính cube trên subset này. Gọi D là DB, S là sample, và $Cube(S)$ là size của cube tính từ S. Size của cube sẽ đc ước lượng là $Cube(S) \times \frac{|D|}{|S|}$. Cách này đơn giản và nhanh, KQ trên real-world data sets cũng là chấp nhận đc.
- **Probablistic counting algorithm:** Bằng cách ước lượng số lượng distinct tuples trong một group, ta có thể ước lượng số lượng tuples trong group này.

8.4.3 Partial Computation of a Data Cube

- Nói chung, có ba cách triển khai một DW: materialize toàn bộ data cube, materialize một selected portion của cube, và ko materialize aggregation nào. Một good trade-off giữa các lựa chọn này là chỉ materialize một portion of views trong data cube. Vấn đề là views nào sẽ đc materialized.
- Ta xét **view selection algorithm**, là một greedy algorithm, khi cho trc một cube lattice, sẽ tìm best set of views để materialize dưới một criterion nhất định.
- Algorithm dùng một lattice mà có xét tới hai loại dependencies giữa các nodes. Loại đầu tiên là cho cases trong đó các attributes của một view đc included trong view khác. VD, trong lattice biểu diễn các possible aggregations of fact table Sales(ProductKey, CustomerKey, DateKey), có một dependency giữa node (ProductKey, CustomerKey) và node (ProductKey), nói rằng node sau có thể đc tính từ node trước vì $\{\text{ProductKey}\} \subseteq \{\text{ProductKey}, \text{CustomerKey}\}$. Loại

dependency thứ hai là cho hierarchies. VD, cho một hierarchy Month→Year, nếu ta có một aggregation trên Month, ta có thể dùng nó để tính aggregation trên Year mà ko cần dùng fact table. Do đó, dependency lattice biểu diễn một relation $v_i \preceq v_j$ giữa các views sao cho view v_i có thể đc tính từ v_j.

- View selection algorithm đc dựa trên việc tính toán costs of computing views trong lattice. Ta giả sử một linear cost model với các characteristics sau:
  - Cost để tính view v từ một materialized view v_m là số rows trong v_m.
  - Tất cả queries là giống nhau đối với một view nào đó trong dependency lattice.
- Algorithm này cũng y/c biết expected number of rows cho mỗi view trong lattice. Cuối cùng, ta giả sử lowest node trong lattice (thường là base fact table) luôn đc materialized.
- M/đ của algorithm là minimize TG cần để tính một view, constrained to materialized a fixed number of views regardless of space available, đc biết đến là một NP-complete. Greedy algorithm dùng một heuristic mà chọn một sequence of views sao cho mỗi choice trong sequence là the best, given what was selected before.
- Gọi $C(v)$ là cost of view, k là số lượng views cần materialize, và S là một tập materialized views. Benefit của việc materialize một view v ko thuộc S so với các materialized views đã thuộc S đgl $B(v, S)$, và đc tính như sau:

  View Materialization Benefit Algorithm
  INPUT: A lattice L, each view node labeled with its expected number of rows
  A node v, not yet selected to materialize
  A set S containing the nodes already selected to materialize
  OUTPUT: The benefit of materializing v given S
  BEGIN
  For each view $w \preceq v$, $w \notin S$, Bw is computed by
  Let u be the view of least cost in S such that $w \preceq u$
  If $C(v) < C(u)$, $Bw = C(u) - C(v)$, otherwise $Bw = 0$
  $B(v, S) = \sum_{w \preceq v} Bw$
  END

- Algorithm hoạt động như sau. Với một view w (chưa đc materialized), ta gọi u là (materialized) view of minimum cost mà từ đó w có thể đc tính. Với một candidate view v đc chọn cho materialization, với mỗi view w phụ thuộc vào v, benefit của

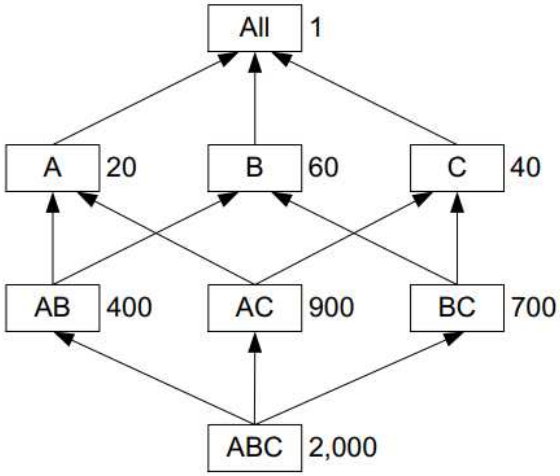
việc materializing v đc tính bằng hiệu giữa costs của v và u. Nếu tính w từ v là more expensive hơn từ u $(C(v) > C(u))$, việc materializing candidate view ko benefit the computation of w $(Bw = 0)$. Algorithm lặp trên tất cả views w, và benefit of materializing v là tổng các individual benefits.

- View selection algorithm tính, trong mỗi iteration, view v mà materialization của nó sẽ cho benefit tối đa.

  View Selection Algorithm
  INPUT: A lattice L, each view node v labeled with its expected number of rows
  The number of views to materialize, k
  OUTPUT: The set of views to materialize
  BEGIN
  $S = \{$The bottom view in $L\}$
  FOR $i = 1$ TO k DO
  Select a view v not in S such that $B(v, S)$ is maximized
  $S = S \cup \{v\}$
  END DO
  S is the selection of views to materialize
  END

  Set S chứa các views đã đc materialized. Trong mỗi vòng lặp, algorithm tính benefit của materialization mỗi views chưa có trong S. View có benefit lớn nhất sẽ đc thêm vào S, và tiếp tục vòng lặp mới.

- Giờ áp dụng vào lattice ở hình dưới. Ngoài node label, bên cạnh mỗi node ta cũng có cost of view mà node biểu diễn. Giả sử ta có thể materialize ba views và bottom views đã đc materialized.



- Giờ ta xét cách chọn view đầu tiên để materialize. Ta cần tính benefit of materializing each view, biết rằng $S = \{ABC\}$. Ta bắt đầu với node AB, là một good candidate, vì nó giảm cost đi 1600 units với mỗi view phụ thuộc vào nó. Hiện tại, tính A có cost 2000, vì nó đc thực hiện từ ABC. Nếu ta materialize AB, cost tính A giảm còn 400.
- Benefit of materializing AB given S là:

$B(AB, S) = \sum_{w \preceq AB} Bw$.
Do đó, với mỗi view
w covered bởi AB, ta tính $C(ABC) - C(AB)$, vì ABC là materialized view duy nhất lúc đầu. VD, để tính B mà ko materializing AB, ta cần scan ABC với cost 2000. Với AB đc materialized, cost giảm còn 400. Điều này đúng với tất cả views có một path tới All đi qua AB, đó là A, B, All, và chính AB. Do đó:

$B(AB, S) = 1600 + 1600 + 1600 + 1600 = 6400.$
Tương tự:

$B(BC, S) = \sum_{w \preceq BC} Bw = 1300 \times 4 = 5200.$
Cứ tiếp tục ta sẽ có AB là view để materialize vì nó có maximum benefit. Vậy khi ta

bắt đầu vòng lặp thứ hai ta sẽ có
$S = \{ABC, AB\}$.

- Giờ xét vòng lặp thứ hai. Benefit of materializing BC là $B(BC, S) = \sum_{w \preceq BC} Bw = 1300 + 1300 = 2600$, tương ứng với C và BC vì materializing BC ko ảnh hưởng đến các nodes tới All qua AB vì chúng có thể đc tính từ AB với cost 400. Mặt khác, benefit of materializing B là $B(B, S) = \sum_{w \preceq B} Bw = 340 \times 2 = 680$ vì cả B và All đều đc tính từ AB với benefit $Bw = 400 - 60 = 340$. Ngoài ra, benefit of materializing C là $B(C, S) = \sum_{w \preceq C} Bw = 1960 + 360 = 2320$. Cứ tiếp tục cuối cùng ta sẽ chọn BC trong iteration thứ hai.
- Cuối cùng, ba view đc chọn sẽ là AB, BC, AC. Bảng sau tổng hợp toàn bộ computation.

View	First Iteration	Second Iteration	Third Iteration
AB	**1,600 × 4 = 6,400**		
AC	1,100 × 4 = 4,400	1,100 × 2 = 2,200	**1,100 × 1 = 1,100**
BC	1,300 × 4 = 5,200	**1,300 x 2 = 2,600**	
A	1,980 × 2 = 3,960	380 × 2 = 760	380 × 2 = 760
B	1,940 × 2 = 3,880	340 × 2 = 680	340 × 2 = 680
C	1,960 × 2 = 3,920	1,960 + (400 - 40) = 2,320	660 + 360 = 1,020
All	1,999 × 1 = 1,999	399 × 1 = 399	399 × 1 = 399

DWSDI 8.5 - Indexes for Data Warehouses

8.5.1 Bitmap Indexes

- Xét table Product dưới đây. Ta xét cách xây dựng một bitmap index trên các attributes QuantityPerUnit và UnitPrice. Ta tạo một bit vector có length 6 (số rows trong Product) cho mỗi possible attribute value, như ở hình (b) và (c).

	45	50	OR1
p1	0	0	0
p2	1	0	1
p3	0	1	1
p4	0	1	1
p5	0	1	1
p6	0	0	0

(a)

	100	110	120	OR2
p1	0	0	0	0
p2	0	0	0	0
p3	0	0	0	0
p4	1	0	0	1
p5	0	0	1	1
p6	0	1	0	1

(b)

	OR1	OR2	AND
p1	0	0	0
p2	1	0	0
p3	1	0	0
p4	1	1	1
p5	1	1	1
p6	0	1	0

(c)

- Best opportunities cho các indexes này là khi cardinality of attributes being indexed là thấp. VD, giả sử Product table có 100,000 rows. Một bitmapped index trên attribute UnitPrice sẽ chiếm $100000 \times 6/8 \text{ bytes} = 0.075 \text{ MB}$. Một traditional B-tree index (dùng cho OLTP) sẽ chiếm $100000 \times 4 \text{ bytes} = 0.4 \text{ MB}$ (giả sử cần 4 bytes để lưu một record identifier). Như vậy space y/c bởi một bitmapped index sẽ tỉ lệ với số lượng entries trong index và số lượng rows, còn space y/c bởi traditional indexes phụ thuộc vào số lượng records đc indexed. Thông thường OLAP systems sẽ inde attributes có cardinality thấp.

8.5.2 Bitmap Compression

- **Run-length encoding:** Ý tưởng cơ bản là, nếu một bit có giá trị v lặp lại n lần, ta thay chúng bằng số n. Chuỗi bits này đgl một run of length n.
- Đối với bitmap indexes, vì các bit vectors có rất ít 1 giữa nhiều 0, nếu một bit 0 lặp lại n lần liên tiếp, ta thay chúng bằng số n. VD, chuỗi bits sau 00000001110000000000011 có hai runs of length 7 và 10. Vector này có thể đc biểu diễn bằng chuỗi integers $7, 1, 1, 10, 1, 1$. Tuy nhiên, cách encoding này là ko rõ ràng vì ta ko phân biệt đc 1 là actual bit hay là length of a run. Ta xử lí như sau. Gọi j là số bits cần để biểu diễn n là length of a run. Ta biểu diễn run là một chuỗi $j-1$ 1 bits, tiếp theo là 0, tiếp theo là n ở dạng binary. VD, run đầu tiên 0000000 sẽ đc encoded là 110111 (7 cần 3 bits để biểu diễn).

8.5.3 Join Indexes

- Ý tưởng chính của join indexes là precomputing join. Xét dimension table Product và fact tabke Sales. Hình (a) là table Product, với một additional attribute

ProductKey	ProductName	QuantityPerUnit	UnitPrice	Discontinued	CategoryKey
p1	prod1	25	60	No	c1
p2	prod2	45	60	Yes	c1
p3	prod3	50	75	No	c2
p4	prod4	50	100	Yes	c2
p5	prod5	50	120	No	c3
p6	prod6	70	110	Yes	c4

(a)

	25	45	50	70
p1	1	0	0	0
p2	0	1	0	0
p3	0	0	1	0
p4	0	0	1	0
p5	0	0	1	0
p6	0	0	0	1

(b)

	60	75	100	110	120
p1	1	0	0	0	0
p2	1	0	0	0	0
p3	0	1	0	0	0
p4	0	0	1	0	0
p5	0	0	0	0	1
p6	0	0	0	1	0

(c)

- Giờ giả sử query “Products with unit price equal to 75”. Một query processor sẽ chỉ cần biết có một bitmap index trên UnitPrice trong Product, và tìm bit vector có giá trị 75. Các vector positions có giá trị 1 đc tìm thấy sẽ xđ vị trí của records TMĐK.
- Với queries liên quan đến một search range, xét query “Products having between 45 and 55 pieces per unit, and with a unit price between 100 and 200”. Đầu tiên ta tìm index trên QuantityPerUnit, và bit vectors có labels giữa 45 và 55, rồi dùng một OR operation giữa các bit vectors này. Ta làm tương tự với UnitPrice. Ta có đc hai vectors OR1 và OR2 (hình (a) và (b)). Cuối cùng, ta dùng AND giữa hai vectors này để có các rows TMĐK (hình (c)).

```
SELECT C.CustomerName, P.ProductName, SUM(S.SalesAmount)
FROM Sales S, Customer C, Product P
WHERE S.CustomerKey = C.CustomerKey AND S.ProductKey = P.Proc
GROUP BY C.CustomerName, P.ProductName
```

Hình (a), (c), (d) là Product và Customer dimension tables, và Sales fact table, còn bitmap indexes ở hình (b), (e), (f).

RowIDProd, và hình (b) là table Sales với một additional attribute RowIDSales. Hình (c) là join index tương ứng, đơn giản là một table chứa các pointers tới matching rows.

RowID Product	Product Key	Product Name	...	Discontinued	...
1	p1	prod1	...	No	...
2	p2	prod2	...	Yes	...
3	p3	prod3	...	No	...
4	p4	prod4	...	Yes	...
5	p5	prod5	...	No	...
6	p6	prod6	...	Yes	...

(a)

RowID Sales	Product Key	Customer Key	Date Key	Sales Amount
1	p1	c1	t1	100
2	p1	c2	t1	100
3	p2	c2	t2	100
4	p2	c2	t3	100
5	p3	c3	t3	100
6	p4	c3	t4	100
7	p5	c4	t5	100

(b)

RowID Sales	RowID Product
1	1
2	1
3	2
4	2
5	3
6	4
7	5

(c)

Yes	No
0	1
0	1
1	0
1	0
0	1
1	0
0	1

(d)

- Một case đặc biệt của join index là bitmap join index. Giả sử query y/c total sales of discontinued products. Trong case này, một bimap index đc tạo ra trên table Sales theo attribute Discontinued, như ở hình (d).

DWSDI 8.6 - Evaluation of Star Queries

- Queries trên star schemas đgl star queries vì chúng dùng star schema structure, từ đó join fact table với dimension tables. VD một typical star query là “Total sales of discontinued products, by customer name and product name”.

Product Key	Product Name	...	Discontinued	...
p1	prod1	...	No	...
p2	prod2	...	Yes	...
p3	prod3	...	No	...
p4	prod4	...	Yes	...
p5	prod5	...	No	...
p6	prod6	...	Yes	...

(a)

Yes	No
0	1
1	0
0	1
1	0
0	1
1	0

(b)

Customer Key	Customer Name	Address	Postal Code	...
c1	cust1	35 Main St.	7373	...
c2	cust2	Av. Roosevelt 50	1050	...
c3	cust3	Av. Louise 233	1080	...
c4	cust4	Rue Gabrielle	1180	...

(c)

Product Key	Customer Key	Date Key	Sales Amount
p1	c1	t1	100
p1	c2	t1	100
p2	c2	t2	100
p2	c2	t3	100
p3	c3	t3	100
p4	c3	t4	100
p5	c4	t5	100

(d)

c1	c2	c3	c4
1	0	0	0
0	1	0	0
0	1	0	0
0	1	0	0
0	0	1	0
0	0	1	0
0	0	0	1

(e)

p1	p2	p3	p4	p5	p6
1	0	0	0	0	0
1	0	0	0	0	0
0	1	0	0	0	0
0	1	0	0	0	0
0	0	1	0	0	0
0	0	0	1	0	0
0	0	0	0	1	0

(f)

Yes	No
0	1
0	1
1	0
1	0
0	1
1	0
0	1

(g)

Fig. 8.13 An example of evaluation of star queries with bitmap indexes: (**a**) Product table, (**b**) Bitmap for Discontinued, (**c**) Customer table, (**d**) Sales fact table, (**e**) Bitmap for CustomerKey, (**f**) Bitmap for ProductKey, and (**g**) Bitmap join index for Discontinued

- Giờ ta mô tả cách query đc evaluated bởi một OLAP engine. Đầu tiên ta cần lấy các record numbers của các records TMĐK trên dimension, đó là Discontinued='Yes'. Như ở hình (b), các records như vậy có ProductKey p2, p4, p6. Sau đó ta truy cập bitmap vectors với các labels này trong hình (f), từ đó thực hiện một join giữa Product (hình (a)) và Sales. Chỉ có các vectors labeled p2 và p4 match the search vì ko có record cho product p6., Hàng ba, bốn, và sáu trong fact table là answer vì chúng có 1 trong corresponding vectors ở hình (f). Sau đó ta có key values cho

CustomerKey (c2 và c3) bằng bitmap index ở hình (e). Với các values này ta tìm trong B+-tree index theo keys trong tables Product và Customer để tìm names of products and customers TMĐK. KQ ta có các names cust2, cust3, prod2, prod4. KQ cuối cùng là (cust2, prod2, 200), (cust3, prod4, 100).

DWSDI 8.7 - Partitioning

- **Partitioning** là chia một table thành các tables nhỏ hơn (đgl **partitions**) để hỗ trợ tốt hơn management and processing of large volumes of data. Từ application perspective, một partitioned table là **identical** với một nonpartitioned table, do đó partitioning là transparent đối với SQL statements.
- Có hai cách để partitioning một table. **Horizontal paritioning** chia một table thành các phần nhỏ hơn có cùng structure với full table nhưng ít records hơn. VD, nếu một số queries chỉ y/c most recent data, một fact table có thể đc horizontally partitioned theo time frame. **Vertical partitioning** chia các attributes của một table thành các groups đc lưu độc lập. Trong case này, một key của relation phải đc lưu ở tất cả partitions để có thể khôi phục tuples. VD, một table có thể đc partitioned sao cho most often used attributes đc lưu trong một partition, còn lại đc lưu ở partition khác. Theo cách này, nhiều records có thể đc đưa vào memory hơn với một single operation, từ đó giảm processing time.

Explain the benefits of the bitmap-join index for data warehouses and give an example of a query that would benefit from such an index (give the relevant part of a star-schema, on which table(s) and attribute(s) the index is built, the query and an explanation of the gain obtained by the index). Then explain why such indices are less suitable for OLTP databases.

Bitmap-Join Index is particularly beneficial for data warehouses due to its ability to efficiently handle queries involving large datasets, especially those with complex joins. Some key advantages include:

Space Efficiency:

Bitmap indexes use a compact representation for multiple values, making them more space-efficient compared to traditional indexes. This is crucial for large data warehouses where storage optimization is essential.

Efficient Joins:

Bitmap-Join indexes are designed for join operations, where they can significantly speed up the processing of complex queries by exploiting bitmap intersections. This is especially advantageous in data warehouses that often involve star or snowflake schema structures with numerous joins.

Support for Star-Schema Queries:

In star-schema data warehousing, where there is a central fact table surrounded by dimension tables, Bitmap-Join indexes are particularly useful. They facilitate efficient query processing by minimizing the need for full table scans during join operations.

Example Scenario:

Consider a simplified star-schema involving a fact table Sales and a dimension table Product. The relevant part of the schema might look like this:

Fact Table - Sales:

Columns: TransactionID, ProductID, Quantity, Amount, DateID, CustomerID

Dimension Table - Product:

Columns: ProductID, Category, Brand, Color, Size, ...

Suppose you have a query to find the total sales amount for a specific product category and color for a given date range. Without a Bitmap-Join index, this query might involve multiple full scans or inefficient index lookups.

```
SELECT    p.Category,    p.Color,    SUM(s.Amount) AS TotalSales
FROM    Sales s
JOIN    Product p ON s.ProductID = p.ProductID
WHERE    p.Category = 'Electronics'    AND p.Color = 'Blue'    AND s.DateID BETWEEN '2022-01-01' AND '2022-12-31'
GROUP BY    p.Category, p.Color;
```

Bitmap-Join Index:

```
-- Assuming the bitmap-join index is created on the columns used in the join
-- Product.ProductID and Sales.ProductI
DCREATE BITMAP JOIN INDEX bj_product_sales
ON Product(ProductID)
FROM Sales(ProductID);
```

Explanation:

The Bitmap-Join index, in this case, is built on the Product.ProductID attribute, facilitating efficient joins with the Sales table's ProductID.

During the query execution, the Bitmap-Join index helps in quickly identifying relevant rows, reducing the need for full scans, and enhancing the performance of join operations.

This is especially beneficial when dealing with large datasets in a data warehousing environment.

Bitmap-Join Index Limitations in OLTP Databases:

While Bitmap-Join indexes are advantageous for data warehouses, they are less suitable for Online Transaction Processing (OLTP) databases due to the following reasons:

Write Performance:

Bitmap indexes can be expensive to maintain during frequent write operations (inserts, updates, deletes). In OLTP databases, where transactions involve frequent data modifications, the overhead of maintaining bitmap indexes can be a performance bottleneck.

Real-Time Querying:

OLTP databases often require real-time responsiveness for individual transactions, and the benefits of bitmap indexing are more pronounced in scenarios involving analytical queries over large datasets. Bitmap indexes are typically optimized for analytical processing and may not be as effective in real-time transactional environments.

Space Overhead:

Bitmap indexes can consume significant storage space, which might be a concern in OLTP databases where storage efficiency is crucial for maintaining high transactional throughput.

In summary, while Bitmap-Join indexes offer substantial benefits for data warehouses, they are less suitable for OLTP databases primarily due to concerns related to write performance, real-time querying, and storage overhead.

Jensen et al. make in their book Multidimensional Databases and Data Warehousing the following statement regarding the use of a surrogate key as the primary key in a dimension table of a star-schema:

[A surrogate key as key column in a dimensional table] has several advantages over the option of information- bearing keys from the source systems,including better storage use, prevention of problems associated with keyre -use, better support for dimension updates, and more efficient queryprocessing.

Explain, in your own words, and with an example of your own the secondand the third benefit; i.e., (a) why a surrogate key prevents problems associatedwith key re-use, and (b) why it better supports dimension updates.

(a) Prevention of Problems Associated with Key Re-use:

When using information-bearing keys from source systems as primary keys in a dimension table, you might face challenges when the keys in the source systems change or are reused over time. This can lead to issues of historical data consistency and integrity. For example:

Example:

Consider a product dimension in a data warehouse where the product code from the source system is used as the primary key. If a product code changes due to a system upgrade or business rule modification, updating the primary key in the dimension table becomes cumbersome. Moreover, if the product code is reused for a different product in the future, it can lead to confusion and incorrect historical associations.

By contrast, using a surrogate key, which is typically an internal identifier generated by the data warehouse, helps avoid these problems. The surrogate key remains stable even if the source system keys change, providing a consistent and unchanging reference point for historical data.

(b) Better Support for Dimension Updates:

Updating dimensions is a common operation in data warehousing, especially when dealing with slowly changing dimensions (SCDs). Surrogate keys offer advantages in supporting these updates more efficiently compared to using information-bearing keys.

Example:

Consider an employee dimension where the employee ID from the source system is used as the primary key. If an employee gets married and changes their last name, updating the employee table with the new last name involves changing the primary key if the employee ID is information-bearing.

With a surrogate key, updates to non-key attributes, like the last name in this example, can be performed without changing the primary key. The surrogate key acts as a stable identifier, and the data warehouse can maintain a historical record of the changes. This supports efficient updates without the need to modify primary key values, reducing the impact on related tables and queries.

In summary, using a surrogate key in a dimension table helps prevent issues associated with key re-use and provides better support for dimension updates, contributing to improved data consistency and ease of maintenance in a data warehousing environment.