Appendix B

CASE STUDIES: SOLUTIONS

It is likely that the solutions shown here will turn out to be not perfect. If you disagree with an answer, please feel free to mail us.

B.1 Hospital

Requirements specification

Each business question is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Hospitalization

Requirements analysis	**Dimensions**	**Measures**	**Metrics**
Total billed amount for hospitalizations, by diagnosis code and description, by month (year).	Diagnosis (ICD, Description), Date (Month, Year)	Amount	Total Amount
Total number of hospitalizations and billed amount, by ward, by patient gender (age at date of admission, city, region).	Ward, Patient (Gender, Age, City, Region)	Amount	Total number Total Amount
Total billed amount, average length of stay and average waiting time by diagnosis code and description, by name (specialization) of the physician who admitted the patient.	Diagnosis (ICD code, Description), Physician (Name, Specialization)	Amount, Duration, WaitingTime	Total Amount Average Duration Average WaitingTime
Total billed amount, and average waiting time for admission by patient age (region), by treatment code (description).	Patient (Age, Region), Treatment (Code, Description)	Amount, Duration, WaitingTime	Total Amount Average WaitingTime

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is a hospitalization of a patient, assuming that they may require one treatment only
Preliminary dimensions	Patient, Date, Ward, Diagnosis, Treatment, Physician
Preliminary measures	Duration, WaitingTime, Amount

The measure **Amount** is **additive**. The measures **Duration** and **WaitingTime** are **non-additive** because in the analysis they are used only to average them.

Conceptual Design

The data mart conceptual design is shown in Figure B.1.

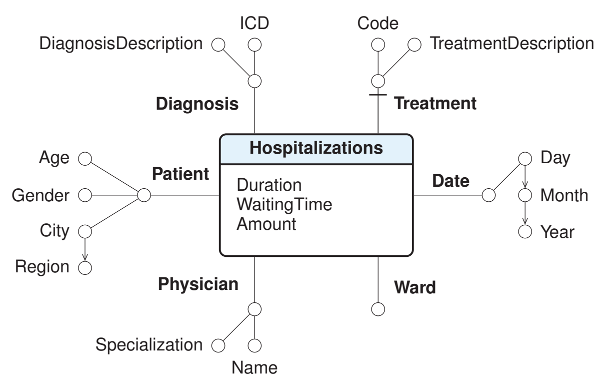


Figure B.1: The conceptual design of a data mart for the hospitalizations

Logical design

In the logical design, the facts are stored in the relation Hospitalizations, with the measures, the degenerate dimension Ward and a foreign key for each dimension table, with its own surrogate primary key (Figure B.2). The surrogate primary key for the Date dimension is a day, an integer of the form YYYYMMDD.

This solution is correct, assuming that if a patient is hospitalized several times with different values of age, its value in the dimension Patient is that of the last hospitalization. If we are interested in storing the value of a patient age at each hospitalization, as desired by the requirements, with the admission of a patient with an age different from the one already present in Patient, a new record is created in the table Patient with a different surrogate primary key (changes dealt with mode Type 2). To find out which data refer to the same patient hospitalizations (for example, to count the different patients hospitalized), InitialPatientKey is added as the attribute in the fact table, with the first surrogate key value assigned to a patient (Figure B.3). This solution also allows us to deal with cases in which, at each new hospitalization, the patient also changes the city and region of residence.

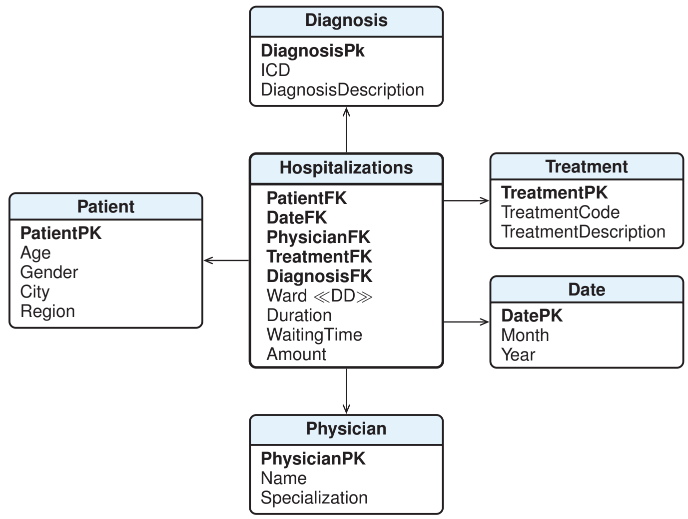


Figure B.2: The initial logical design of a data mart for the hospitalizations

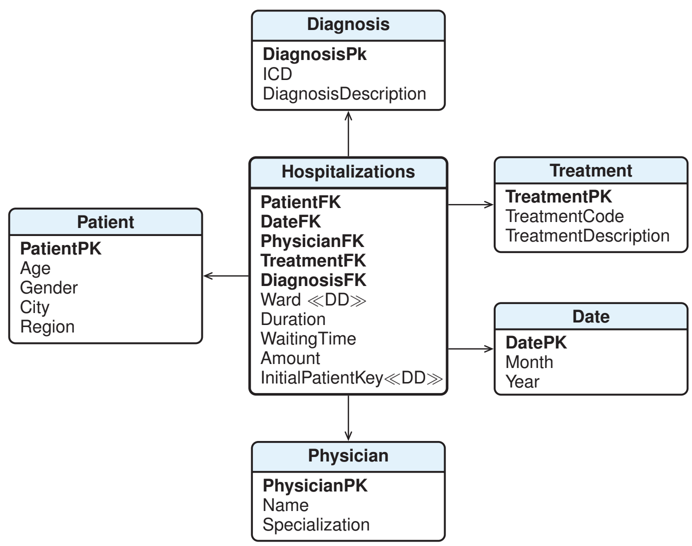


Figure B.3: The final logical design of a data mart for the hospitalizations

B.2 Airline Companies

Requirements specification

Each business requirement analysis is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Airline companies

Requirements analysis	**Dimensions**	**Measure**	**Metrics**
Number of unoccupied seats in a given year, by flight code, by company name (or type), by class, by departure time (time, day, month, year)	FlightCode, Class, Company(Name, Type), DepartureTime (Time, Day, Month, Year)	UnoccupiedSeats	Total UnoccupiedSeats
Number of unoccupied seats in a given class and year, by flight code, by company name, by class, by departure (destination) city (country, continent).	FlightCode, Class, Company(Name), DepartureCity (Country, Continent), DestinationCity (Country, Continent)	UnoccupiedSeats	Total UnoccupiedSeats
Number of unoccupied seats and revenue of the Alitalia company, by year, by month, by destination country.	Company(Name), DepartureTime (Month, Year), DepartureCity(Country)	UnoccupiedSeats Revenue	Total UnoccupiedSeats, Revenue

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is the information on the number of unoccupied seats on a flight of a class of a company
Preliminary dimensions	Class, FlightCode, Company, Departure time, Departure city, Destination city
Preliminary measures	UnoccupiedSeats, Revenue

The measures are **additive**.

Conceptual Design

The data mart conceptual design is shown in Figure B.4.

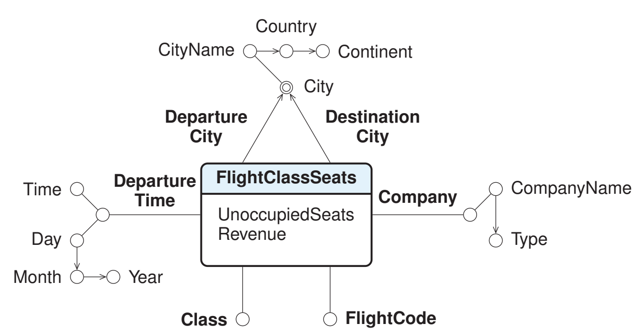


Figure B.4: The conceptual design of a data mart for the airline companies

Logical design

In the logical design, the facts are stored in the relation FlightClassSeats, with the measures, the degenerate dimensions Class, FlightCode and a foreign key for each dimension table, with its own surrogate primary key (Figure B.5).

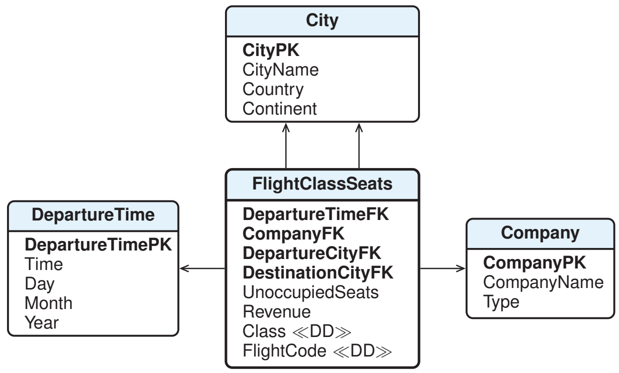


Figure B.5: The logical design of a data mart for the airline companies

B.3 Airline Flights

Requirements specification

Each business requirement analysis is analyzed to identify the dimensions and the measures used, and the aggregations to compute (*metrics*):

Flight Process

Requirements analysis	Dimensions	Measures	Metrics
Number of first-class passengers in a given month and year, by country, by age range of passengers.	Passenger (Nationality, AgeRange), Class, DepartureDate(Month, Year)		Number of passengers
Number of passengers from Europe to the U.S. in a given month and year, and the total revenue, by country, by age range of passengers.	Passenger (Nationality, AgeBand), DepartureDate(Month, Year), DepartureAirport(Continent), DestinationAirport(Country)	Price	Number of passengers, Total price
Number of flights by departure city, by destination city.	DepartureAirport(City), DestinationAirport(City)		Number of flights
Average number of airline passengers by month, by aircraft type, by destination country.	DepartureDate(Month), Aircraft(Type), DestinationAirport(Country)		Average number of passengers
Average number of airline passengers by class, by holiday date.	Class, DepartureDate(HolidayFlag)		Average number of passengers
Number of passengers per year, by size of the destination airport.	DestinationAirport(Size), DepartureDate(Year)		Number of passengers
Number of flights to airports in Germany from the October to December quarter of a given year, and total management cost of the aircraft, by aircraft type.	DepartureDate(Month, Year), DestinationAirport(Country), Aircraft(Type, ManagementCost)		Number of flights, Total management cost
Average profit of all flights, by country of departure, by destination country. The profit of a flight is the total passenger price minus the total flight cost.	DepartureAirport(Country), DestinationAirport(Country), Flight(Duration), Aircraft(ManagementCost, HourlyOperatingCost)	Price	Average profit
Total revenue in a given year of flights, by month, by destination country. The total revenue by month, total revenue by destination country, and the total revenue are also of interest.	DestinationAirport(Country), DepartureDate(Month, Year)	Price	Total Price

From the requirements specification the following fact granularity arises:

Fact granularity

Description	A fact is the information on the ticket of a passenger flight
Preliminary dimensions	Passenger, Flight, Class, Aircraft, Departure Airport, Destination Airport
Preliminary measures	Price

The measure **Price** is **additive**.

Conceptual Design

The data mart conceptual design is shown in Figure B.6:

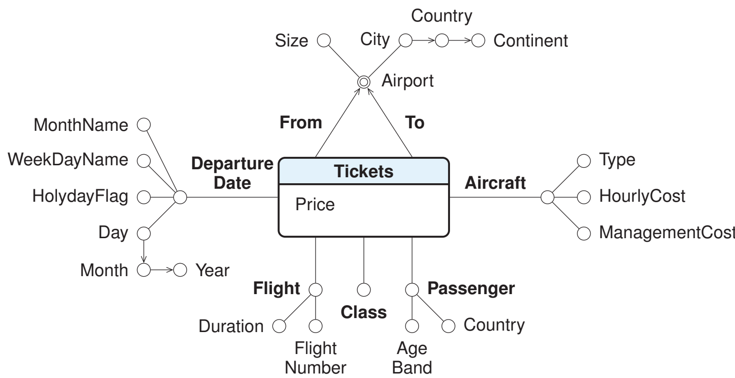


Figure B.6: The conceptual design of a data mart for the airline flights

Logical design

In the logical design, the facts are stored in the relation Tickets, with the measures, the degenerate dimension Class and a foreign key for each dimension table, with its own surrogate primary key (Figure B.7). The surrogate primary key for the DepartureDate dimension is a day, an integer of the form YYYYMMDD.

To simplify the SQL analysis, the degenerate dimension FlightID has been added to the fact table to identify the flight of a ticket, with a value the chaining together of the FlightFK and DepartureDateFK values. If FlightID is not used, in the SQL analysis it will be substituted by the expression:

```
(CAST (FlightFK AS varchar) + CAST (DepartureDateFK AS varchar))
```

The table Passenger has as many elements as are the different combinations of Nationality and AgeBand values.

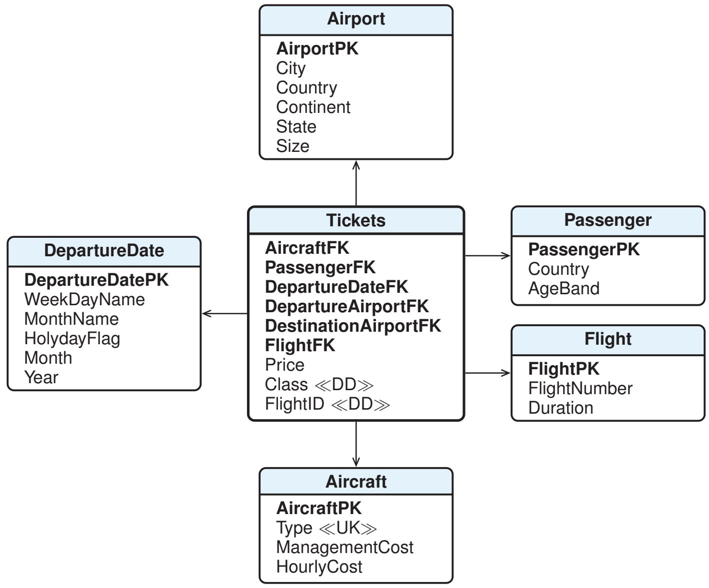


Figure B.7: The logical design of a data mart for the airline flights

Data Analysis

Let us assume that and a month is represented as the integer YYYYMM, and a holiday date has the HolidayFlag = true.

1. Number of first-class passengers in a given month and year, by country, by age range of passengers.

```
SELECT     Country, AgeBand, COUNT(*) AS NoOfPassengers
FROM       Tickets, DepartureDate, Passenger
WHERE      PassengerFK = PassengerPK AND DepartureDateFK = DepartureDatePK
           AND  Month = 200812 AND Class = 1
GROUP BY   Country, AgeBand;
```

2. Number of passengers from Europe to the U.S. in a given month and year, and the total revenue, by country, by age range of passengers.

```
SELECT     Country, AgeBand
           , COUNT(*) AS NoOfPassengers, SUM(Price) AS Revenue
FROM       Tickets, Airport FRM, Airport TO, DepartureDate, Passenger
WHERE      DepartureAirportFK = FRM.AirportPK
           AND DestinationAirportFK = TO.AirportPK
           AND PassengerFK = PassengerPK AND DepartureDateFK = DepartureDatePK
           AND Month = 200812 AND FRM.Continent = 'Europa' AND TO.Country = 'USA'
GROUP BY   Country, AgeBand;
```

3. Number of flights, by departure city, by destination city.

```
SELECT    FRM.City AS DepartureCity, TO.City AS DestinationCity
          , COUNT(DISTINCT FlightID) AS NoOfFlights
FROM      Tickets, Airport FRM, Airport TO
WHERE     DepartureAirportFK = FRM.AirportPK
          AND DestinationAirportFK = TO.AirportPK
GROUP BY  FRM.City, TO.City;
```

4. Average number of airline passengers by month, by aircraft type, by destination country.

```
SELECT    MonthName , Type AS AircraftType, Country AS DestinationCountry
          , COUNT(∗) / COUNT(DISTINCT FlightID) AS AvgNoOfPassengers
FROM      Tickets, Airport, DepartureDate, Aircraft
WHERE     DestinationAirportFK = AirportPK
          AND DepartureDateFK = DepartureDatePK AND AircraftFK = AircraftPK
GROUP BY  MonthName, Type, Country;
```

5. Average number of airline passengers, by class, by holiday date.

```
SELECT    Class, DepartureDateFK AS HolydayDate
          , COUNT(∗) / COUNT(DISTINCT FlightID) AS AvgNoOfPassengers
FROM      Tickets, DepartureDate
WHERE     DepartureDateFK = DepartureDatePK AND HolydayFlag
GROUP BY  Class, DepartureDateFK;
```

6. Number of passengers, by year, by size of the destination airport.

```
SELECT    Year, Size AS SizeDestinationAirport
          , COUNT(∗) AS NoOfPassengers
FROM      Tickets, Airport, DepartureDate
WHERE     DestinationAirportFK = AirportPK
          AND DepartureDateFK = DepartureDatePK
GROUP BY  Year, Size;
```

7. Number of flights to airports in Germany from the October to December quarter of a given year, and total management cost of the aircraft, by aircraft type.

```
SELECT    Type
          , COUNT(DISTINCT FlightID) AS NoOfFlights
          , COUNT(DISTINCT Month)∗ManagementCost AS TotalManagementCost
FROM      Tickets, Airport, DepartureDate, Aircraft
WHERE     DestinationAirportFK = AirportPK
          AND DepartureDateFK = DepartureDatePK AND AircraftFK = AircraftPK
          AND Country = 'Germania' AND Month IN (200710 , 200711 , 200712)
GROUP BY  Type, ManagementCost;
```


8. Average profit of all flights by country of departure and by destination country. The profit of a flight is the total passenger price minus the total flight cost.

```
WITH        Price-FlightHourlyCost-FlighManagementCost AS
            ( SELECT    Type
                        , FRM.Country AS DepartureCountry
                        , TO.Country AS DestinationCountry
                        , SUM(Price) AS TotalPrice
                        , HourlyCost*Duration*COUNT(DISTINCT FlightID)
                              AS FlightHourlyCost
                        , ManagementCost*COUNT(DISTINCT Month)
                              AS FlighManagementCost
              FROM      Tickets, Airport FRM, Airport TO, Flight, Aircraft, DepartureDate
              WHERE     DepartureAirportFK = FRM.AirportPK
                        AND DestinationAirportFK = TO.AirportPK
                        AND FlightFK = FlightPK
                        AND AircraftFK = AircraftPK
                        AND DepartureDateFK = DepartureDatePK
              GROUP BY  FlightFK, Type, FRM.Country, TO.Country, HourlyCost,
                        ManagementCost, Duration
            )

SELECT      DepartureCountry
            , DestinationCountry
            , (SUM(TotalPrice) −
                SUM(FlightHourlyCost) − SUM(FlighManagementCost))/COUNT(*)
                    AS FlightsAvgProfit
FROM        Price-FlightHourlyCost-FlighManagementCost
GROUP BY    DepartureCountry, DestinationCountry;
```

9. Total revenue in a given year of flights by month and by destination country. The total revenue by month, total revenue by destination country, and the total revenue are also of interest.

```
SELECT      MonthName, Country AS DestinationCountry
            , SUM(Price) AS TotalRevenue,
FROM        Tickets, Airport, DepartureDate
WHERE       DestinationAirportFK = AirportPK AND DepartureDateFK = DepartureDatePK
            AND Year = 2008
GROUP BY    CUBE (MonthName, Country);
```

B.4 Inventory

Requirements specification

From the examples of business questions the following fact granularity arises:

Fact granularity

Description	A fact is the information on the monthly values of the quantities of products on hand, acquired and shipped
Preliminary dimensions	Product (SKUProduct, Name, Category), Date (Month, Quarter, Year) Warehouse (Name, City, Region, Area)
Preliminary measures	QtyOnHand, QtyAcquired, QtyShipped

The measures **QtyAcquired** and **QtyShipped** are **semi-additive** with respect to the dimension **Product**. In fact, for analysis of inventories, it makes sense to aggregate quantities of a specific product, e.g., in order to calculate the Inventory turns of a specific product. Aggregation of different products makes sense instead when considering measures of weight, space, or monetary value.

The measure **QtyOnHand** is **semi-additive** with respect to both the dimension **Date** and the dimension **Product**.

The metrics **Inventory Turns** and **Days in Inventory**, defined with a ratio, are **non-additive** and cannot be considered as measures.

Conceptual Design

The data mart conceptual design is shown in Figure B.8.

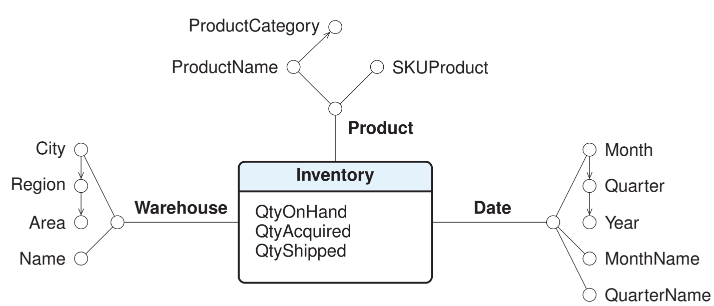


Figure B.8: The conceptual design of a data mart for the Inventory

Logical design

In the logical design, the facts are stored in the relation Inventory, with the measures, and a foreign key for each dimension table, with its own surrogate primary key (Figure B.9). The surrogate primary key for the Date dimension is a month, an integer of the form YYYYMM.

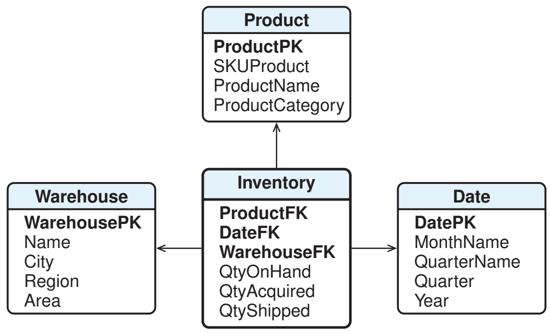


Figure B.9: The logical design of a data mart for the Inventory

Data Analysis

1. **Report 1.** Total of **QtyOnHand** in January 2010, by product (SKU and Product Name), by region. The subtotal by all regions is also of interest.

```
SELECT     SKUProduct, ProductName, Region
           , SUM(QtyOnHand) AS TotalQtyOnHand
FROM       Inventory, Product, Warehouse
WHERE      ProductFK = ProductPK AND WarehouseFK = WarehousePK
           AND DateFK = 201001
GROUP BY   SKUProduct, ProductName, ROLLUP(Region);
```

2. **Report 2.** Total of **QtyOnHand** in the first quarter 2010, by product category, by month name.

   A value of the attribute Quarter is an integer of the form YYYYQ.

   This report has no subtotals, as the previous one, because a subtotal for each category would require totalling the quantities over time, which is meaningless. Adding together the month-end quantities for January, February, and March produces a number that has no meaning. It does not represent the quantity on hand at the end of the period; the March value alone tells us that.

   When summing a semi-additive measure such as **QtyOnHand**, the dimension across which it is not additive (time) must be used to constrain the query, as in **Report 1**, or the semi-additive measure must be grouped by the dimension in question, as in this report, without a further total or subtotal.

   As "subtotals" we can compute the average of the values, but attention is required in correctly computing the average of a set of values as the sum of the values divided by the number of values. In this example the standard SQL average function will not perform this calculation correctly because it assumes as cardinality of a set of values the number of elements of a group of records. For example, if we have two products of the same category available in two warehouses every month of a quarter,

QtyOnHand of product category C1 First Quarter 2010					
Product Category	**DateFK**	**WarehouseFK**	**ProductFK**	**Month Name**	**QtyOnHand**
C1	201001	1	1	January	500
C1	201001	2	2	January	400
C1	201002	1	1	February	100
C1	201002	2	2	February	100
C1	201003	1	1	March	200
C1	201003	2	2	March	300

grouping the data on ProductCategory, C1 will appear in 6 records and so

$$\text{AVG(QtyOnHand)} = \frac{\text{SUM(QtyOnHand)}}{6}$$

while the correct value is

$$\text{AVG(QtyOnHand)} = \frac{\text{SUM(QtyOnHand)}}{3\ \text{(months of the quarter)}}$$

The problem is avoided by computing the average without using the SQL average function, as follows.

```
SELECT      ProductCategory, MonthName AS Month
            , SUM(QtyOnHand) / COUNT(DISTINCT DateFK) AS TotalQtyOnHand
FROM        Inventory, Product, Date
WHERE       ProductFK = ProductPK AND DateFK = DatePK AND Quarter = 20101
GROUP BY    ProductCategory, ROLLUP(MonthName);
```

3. **Report 3.** Values of the *Inventory Turns* and *Days in Inventory* in the year 2010, by product category, by quarter name.

   The **non-additive** metrics *Inventory Turns* and *Days in Inventory*, must be computed by a "*ratio of sum* and not by a *sum of ratio*".

```
SELECT      ProductCategory, QuarterName AS Quarter
            ,SUM(QtyShipped) / (SUM(QtyOnHand) / COUNT(DISTINCT DateFK))
                AS InventoryTurns
            ,90 * ( SUM(QtyOnHand) / COUNT(DISTINCT DateFK) )
                /    SUM(QtyShipped)
                AS DaysInInventory
FROM        Inventory, Product, Date
WHERE       ProductFK = ProductPK AND DateFK = DatePK AND Year = 2010
GROUP BY    ProductCategory, QuarterName;
```

B.5 Hotels

Requirements specification

From the requirements the following fact granularity arises :

Fact granularity

Description	A fact is the information on the daily room type utilization and revenue of each hotel
Preliminary dimensions	Room type, Date, Hotel
Preliminary measures	NOccupiedRooms, NVacantRooms, NUnavailableRooms, NOccupants, Revenue

The dimension **Room type** has as many attributes as the properties of a room, with the attributes for the optional features available with values 'Y' or 'N'.

The measures **NOccupants** and **Reveue** are **additive**.

The measures **NoOccupiedRooms**, **NVacantRooms** and **NUnavailableRooms** are **semi-additive** with respect to **Date**.

The metrics **Occupancy Rate**, **Average Available Room Revenue** and **Average Room Revenue** are **non-additive** and must not be defined as measures.

Conceptual Design

The conceptual design of a data mart is shown in Figure B.10.

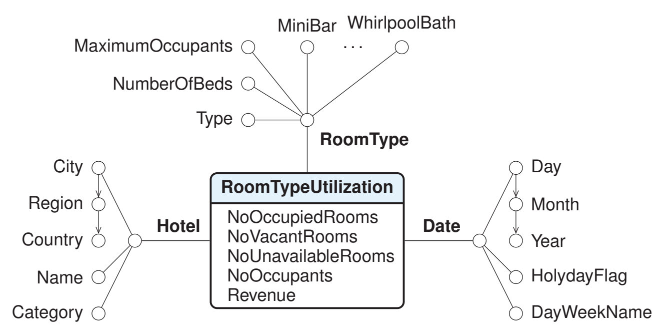


Figure B.10: The conceptual design of a data mart for the hotel room type utilization

Logical design

In the logical design, the facts are stored in the relation RoomTypeUtilization, with the measures, and a foreign key for each dimension table, with its own surrogate primary key (Figure B.11). The surrogate primary key for the Date dimension is a day, an integer of the form YYYYMMDD.

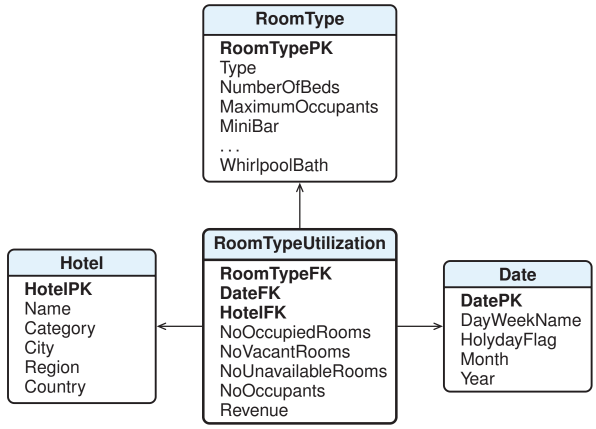


Figure B.11: The logical design of a data mart for the hotel room type utilization

Data Analysis

1. The *room occupancy rate* of hotels of a given city and day, by hotel.

```
SELECT      H.Name
            , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                       SUM(F.NVacantRooms) +
                                       SUM(F.NUnavailableRooms) )
                AS OccupancyRate
FROM        RoomTypeUtilization F, Hotel H
WHERE       F.HotelFK = H.HotelPK AND F.DateFK = 20100717
            AND H.City = 'Florence'
GROUP BY    F.HotelFK, H.Name;
```

2. The *room occupancy rate* of hotels of a given region and day, by room type.

```
SELECT      R.Type
            , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                       SUM(F.NVacantRooms) +
                                       SUM(F.NUnavailableRooms) )
                AS OccupancyRate
FROM        RoomTypeUtilization F, RoomType R, Hotel H
WHERE       F.HotelFK = H.HotelPK AND F.RoomTypeFK = R.RoomTypePK
            AND H.Region = 'Tuscany' AND F.DateFK = 20100717
GROUP BY    R.Type;
```

3. The *room occupancy rate* at a given month and year, by hotel in a given city.

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate,
FROM      RoomTypeUtilization F, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
           AND D.Month = 201007 AND H.City = 'Florence'
GROUP BY  F.HotelFK, H.Name;
```

4. The *room occupancy rate* and *average room revenue* of hotels in a given city, at a given month and year, by hotel.

```
SELECT    H.Name
          , SUM(F.NOccupiedRooms) / (SUM(F.NOccupiedRooms) +
                                     SUM(F.NVacantRooms) +
                                     SUM(F.NUnavailableRooms) )
              AS OccupancyRate
          , SUM(F.Revenue)/SUM(F.NOccupiedRooms) AS AvgRevenueByRoom
FROM      RoomTypeUtilization F, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
           AND D.Month = 201007 AND H.City= 'Milan'
GROUP BY  F.HotelFK, H.Name;
```

5. The monthly revenue and the cumulative revenue of 4-star hotels in a given year, by country and by month.

```
SELECT    H.Country, D.Month
          , SUM(F.Revenue) AS MonthlyRevenue
          , SUM(SUM(F.Revenue)) OVER
              (PARTITION BY H.Country ORDER BY D.Month
                  ROWS UNBOUND PRECEDING)
                      AS CumulativeRevenue
FROM      RoomTypeUtilization F, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK AND D.Year = 2010
GROUP BY  H.Country, D.Month;
```

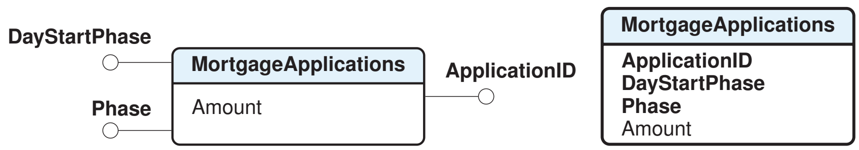
6. In a given year, the total revenue, and the cumulative revenue, of the rooms with the maximum number of occupants and whirlpool bath, by hotel.

```
SELECT    F.HotelFK, H.Name, SUM(F.Revenue) AS TotalRevenue
          , SUM(SUM(F.Revenue)) OVER
              (ROWS UNBOUND PRECEDING)
                  AS CumulativeRevenue
FROM      RoomTypeUtilization F, RoomType R, Date D, Hotel H
WHERE     F.HotelFK = H.HotelPK AND F.DateFK = D.DatePK
          AND F.RoomTypeFK = R.RoomTypePK
          AND D.Year = 2010 AND R.WhirpoolBath = 'Y'
          AND F.NOccupants = R.MaximumOccupants
GROUP BY  F.HotelFK, H.Name;
```

B.6 Mortgage Applications

First Solution: A Transaction Fact

Let us assume that a fact is a phase of the mortgage application and the data mart conceptual and logical designs are those in Figure B.12. When a mortgage application is submitted, a row is inserted into the fact table, with the application code, the phase number, and the start day number. Each time the application enters the next phase, an additional row will be added for the application, with the start day number of the new phase, and so on.



(a) *Conceptual design* **(b)** *Logical design*

Figure B.12: A transaction fact

The business questions about *processing volumes* are easy to write in SQL.

1. Number of applications, by phase.

```
SELECT      Phase
            , COUNT(*) AS No
FROM        MortgageApplications
GROUP BY    Phase;
```

2. Number of closed applications and total amount of applications.

```
SELECT      COUNT(*) AS No
            , SUM(Amount) AS TotalAmount
FROM        MortgageApplications
WHERE       Phase = 4;
```

3. Number of applications not yet closed and total amount of applications.

```
SELECT      COUNT(*) AS No
            , SUM(Amount) AS TotalAmount
FROM        MortgageApplications A
WHERE       A.Phase = 1
            AND NOT EXISTS (
            SELECT   *
            FROM     MortgageApplications B
            WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4);
```

   The business questions about *process efficiency* are neither easy to write in SQL nor likely to be efficient on large fact tables, because it is necessary to correlate fact rows that represent the phase changes. For example, to find the duration of an approved application requires computing the number of days between its submission and its closing phase start days, and this information is stored in separate rows.

4. Number of applications and average processing time, by phase completed.

```
SELECT      B.Phase AS Phase
            , COUNT(*) AS No
            , AVG(A.DayStartPhase − B.DayStartPhase) AS AvgProcTime
FROM        MortgageApplications A, MortgageApplications B
WHERE       A.ApplicationID = B.ApplicationID AND B.Phase = A.Phase − 1
GROUP BY    B.Phase;
```

5. Total processing time of closed applications, by application.

```
SELECT      A.ApplicationID AS ApplicationID
            , MAX(A.DayStartPhase) − MIN(A.DayStartPhase) AS ProcTime
FROM        MortgageApplications A
WHERE       EXISTS (
            SELECT   *
            FROM     MortgageApplications B
            WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4)
GROUP BY    A.ApplicationID;
```

6. Number of closed applications and average processing time.

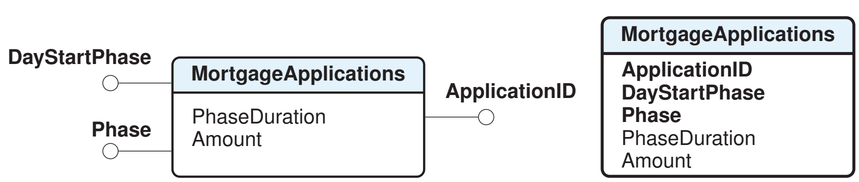
```
WITH        DurationClosedApplications AS (
            SELECT   A.ApplicationID AS ApplicationID
                     , MAX(A.DayStartPhase) − MIN(A.DayStartPhase) AS ProcTime
            FROM     MortgageApplications A
            WHERE    EXISTS (
                     SELECT   *
                     FROM     MortgageApplications B
                     WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4)
            GROUP BYA.ApplicationID
            )

SELECT      COUNT(*) AS No,
            , AVG(ProcTime) AS AvgProcTime
FROM        DurationClosedApplications;
```

Second Solution: A Transaction Fact with Duration of Phases

The business questions about *process efficiency* can be simplified in SQL by using two pieces of information for each application phase: the day on which the phase begins and duration of the phase (the number of days) (Figure B.13).

When a mortgage application is submitted, a row is inserted into the fact table, with the application code, the phase number, the start day number, and the duration of the phase have the value 0. Each time the application enters a new phase, an additional row will be added for the application, with the start day number, and the duration in the fact table row of the previous phase is updated with its value.

The duration of the *Closing* phase has the value 0.



(a) *Conceptual design* **(b)** *Logical design*

Figure B.13: A transaction fact with duration of phases

The following table displays the rows of a simple Mortgage Applications transaction fact table with the duration of phases.

Mortgage Applications
Current Year

Application Code	Day Start Phase	Phase	Phase Duration	Amount
1	100	1	5	100
1	105	2	25	100
1	130	3	20	100
1	150	4	0	100
2	110	1	10	200
2	120	2	30	200
2	150	3	20	200
2	170	4	0	200
3	120	1	20	300
3	140	2	30	300
3	170	3	0	300
4	120	1	0	400
5	115	1	20	500
5	135	2	0	500

The SQL queries for the first three business questions are the same. Let us see those that change.

4. Number of applications and average processing time, by phase completed.

```
SELECT     Phase, COUNT(∗) AS No
           , AVG(PhaseDuration) AS AvgProcTime
FROM       MortgageApplications
WHERE      PhaseDuration > 0
GROUP BY   Phase;
```

5. Total processing time of closed applications, by application.

```
SELECT     ApplicationID, SUM(A.DayStartPhase − B.DayStartPhase) AS ProcTime
FROM       MortgageApplications
WHERE      EXISTS (
           SELECT   ∗
           FROM     MortgageApplications B
           WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4)
GROUP BY   ApplicationID;
```

6. Number of closed applications and average processing time.

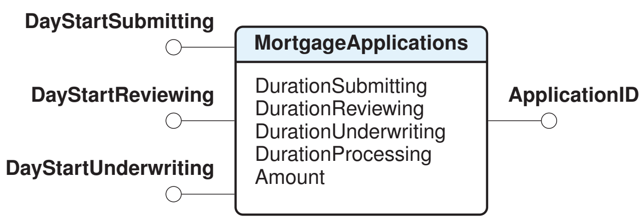
```
SELECT     COUNT(DISTINCT A.ApplicationID) AS No
           , SUM(A.PhaseDuration) / COUNT(DISTINCT A.ApplicationID)
               AS AvgProcTime
FROM       MortgageApplications A
WHERE      EXISTS (
           SELECT   ∗
           FROM     MortgageApplications B
           WHERE    A.ApplicationID = B.ApplicationID AND B.Phase = 4);
```

Unfortunately, this approach does not eliminate the correlated subquery when looking at the time spent across multiple phases. Let us see another solution to simplify the SQL queries.

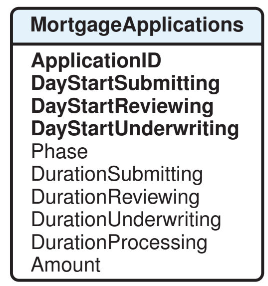
Third Solution: An Accumulating Snapshot Fact

Phase start day and phase duration do not simplify all the business questions about *process efficiency* in SQL using a *transaction fact* table. A better solution is a different data mart design based on an *accumulating snapshot fact*: there is one row for each application, independently of the number of phases, and the fact table rows are updated when a phase begins or terminates (Figure B.14).

Constructed in this manner, the accumulating snapshot fact is a useful and powerful tool for studying time spent at any phase or any combination of phases. Duration of phases can be studied in terms of their minimums, maximums, or averages across any relevant dimensions, simply by aggregating the appropriate measures as required.



(a) *Conceptual design*



(b) *Logical design*

Figure B.14: An accumulating snapshot fact

The following table displays the rows of a simple Mortgage Applications accumulating snapshot fact table.

Mortgage Applications Current Year				
Application ID	**Day Start Submitting**	**Day Start Reviewing**	**Day Start Underwriting**	**Phase**
1	100	105	130	4
2	110	120	150	4
3	120	140	170	3
4	120	0	0	1
5	115	135	0	2

Mortgage Applications Current Year				
Duration Submitting	**Duration Reviewing**	**Duration Underwriting**	**Duration Processing**	**Amount**
5	25	20	50	100
10	30	20	60	200
20	30	0	50	300
0	0	0	0	400
20	0	0	20	500

Let us assume that the following table exists about the phases of mortgage applications.

Phases

PhaseNo	Description
1	Submitting
2	Reviewing
3	Underwriting
4	Closing

Let us see how the SQL queries change and perform better on large fact tables.

1. Number of applications, by phase.

```
SELECT     PhaseNo AS Phase, COUNT(*) AS No
FROM       MortgageApplications, Phases
WHERE      Phase >= PhaseNo
GROUP BY   PhaseNo
ORDER BY   PhaseNo;
```

2. Number of closed applications and total amount of applications.

```
SELECT     COUNT(*) AS No
           , SUM(Amount) AS TotalAmount
FROM       MortgageApplications
WHERE      Phase = 4;
```

3. Number of applications not yet closed and total amount of applications.

```
SELECT     COUNT(*) AS No
           , SUM(Amount) AS TotalAmount
FROM       MortgageApplications
WHERE      Phase < 4;
```

4. Number of applications and average processing time, by phase completed.

```
SELECT     PhaseNo AS Phase, COUNT(*) AS No
           , AVG(CASE PhaseNo
                 WHEN 1 THEN DurationSubmitting
                 WHEN 2 THEN DurationReviewing
                 WHEN 3 THEN DurationUnderwriting END) AS AvgProcTime
FROM       MortgageApplications, Phases
WHERE      Phase > PhaseNo
GROUP BY   PhaseNo
ORDER BY   PhaseNo;
```

5. Total processing time of closed applications, by application.

```
SELECT     ApplicationID, DurationProcessing
FROM       MortgageApplications
WHERE      Phase = 4;
```

6. Number of closed applications and average processing time.

```
SELECT     COUNT(*) AS No
           , AVG(DurationProcessing) AS AvgProcTime
FROM       MortgageApplications
WHERE      Phase = 4;
```

Fourth Solution: A More General Accumulating Snapshot Fact

Let us assume that the bank is also interested in the following business questions:

– Number of mortgage applications, and total funding requested, by interest rate, by year, by quarter, by month.
– Number of mortgages underwritten, the total amount underwritten, the average difference between the amount requested by the application and the amount underwritten, by type rate, by year, by quarter, by month.
– Number of mortgage applications denied, and the average duration of the review and processing phase, by the applicant's income range.
– Minimum, maximum and average duration of mortgage applications underwritten, by the employee who reviewed and processed the application.

Figure B.15 shows the conceptual design of a possible data mart to support in addition the new business questions.

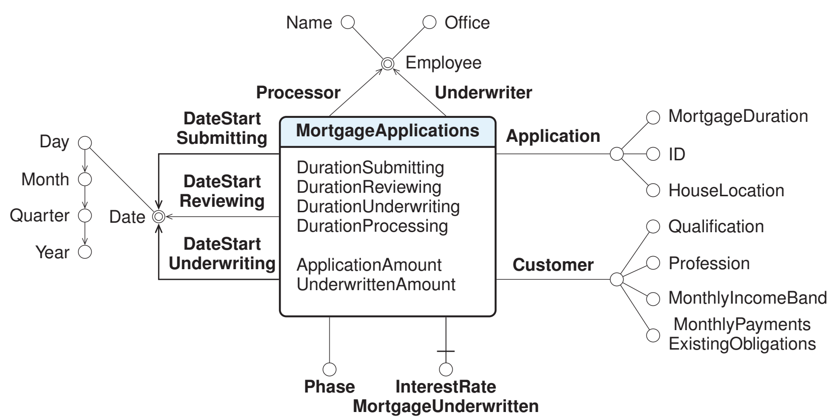


Figure B.15: The final data mart Mortgage Applications conceptual design