Chapter 5
Bridge Tables

A *Bridge Table* is a table that links two dimensions, and only one of these two dimensions is linked to the fact. As a result, the star schema becomes a *snowflake* schema. The term snowflake schema was introduced in the previous chapter on *Hierarchy*, where a string of dimensions is linked through a hierarchy relationship, and the parent dimension is linked to the fact. It is called a snowflake because the hierarchy of dimension is like a snowdrop. Remember that the cardinality relationship in the hierarchy from the parent dimension to the child dimension is *many*-1. A Bridge Table is also a snowflake, but the relationship between the two dimensions that are linked through a Bridge Table has a cardinality of 1-*many* and *many*-1.

The following example shows the difference between a normal star schema (Fig. 5.1), a snowflake with a Bridge Table (Fig. 5.2) and a snowflake with a hierarchy (Fig. 5.3). In the snowflake with a Bridge Table, Dim-4b is not linked directly to the fact but is connected to another dimension (e.g. Dim-4a) through a Bridge Table. Dim-4a has a 1-*many* relationship with the Bridge Table, which in turn has a *many*-1 relationship with Dim-4b.

On the other hand, the snowflake with a hierarchy (Fig. 5.3) shows that dimensions 4a, 4b and 4c are linked through a hierarchy relationship with *many*-1 cardinalities between each pair of parent-child dimensions.

A Bridge Table is needed when a dimension cannot be connected directly to the Fact as it has to go through another dimension. For example, Dim-4b cannot be connected directly to the fact without going through Dim-4a. For Dim-4b to be connected to Dim-4a, it has to go through a Bridge Table. This is because the cardinality relationship between Dim-4a and Dim-4b is actually a *many-many* relationship which is why a Bridge Table needs to be an intermediate table between these two dimensions.

D. Taniar, W. Rahayu, *Data Warehousing and Analytics*, Data-Centric Systems and Applications, https://doi.org/10.1007/978-3-030-81979-8_5

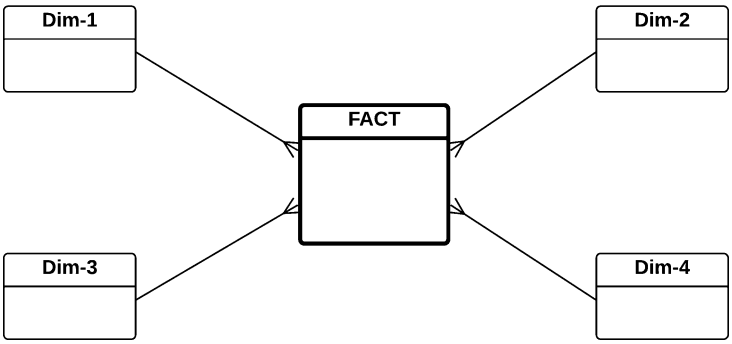

Fig. 5.1 A star schema

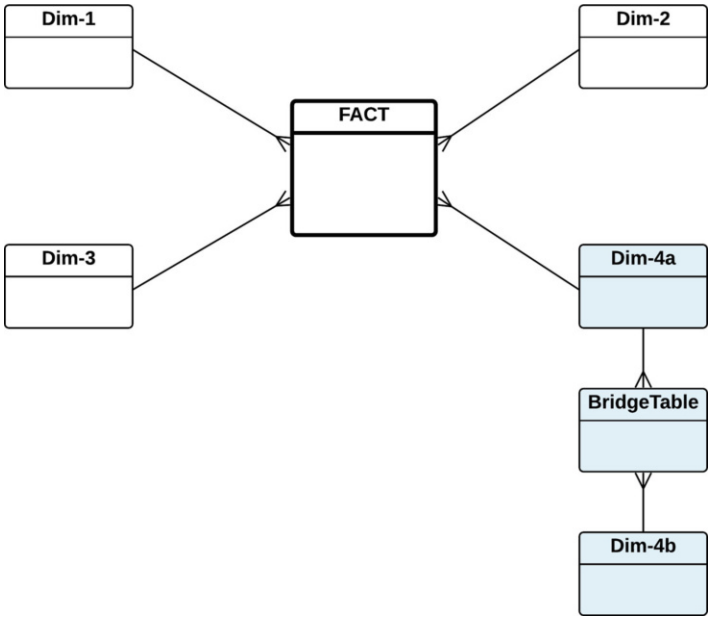

Fig. 5.2 A snowflake with a Bridge Table

There are at least two reasons, why a dimension cannot be connected directly to the Fact:

(a) The Fact Table has a fact measure, and the dimension has a key identity. In order to connect a dimension to the Fact, the dimension's key identity must contribute directly to the calculation of the fact measure. Unfortunately, this cannot happen if the operational database does not have this data.
(b) The operational database does not have this data if the relationship between two entities in the operational database that hold the information about the dimension's key identity and the intended fact measure is a *many-many* relationship.

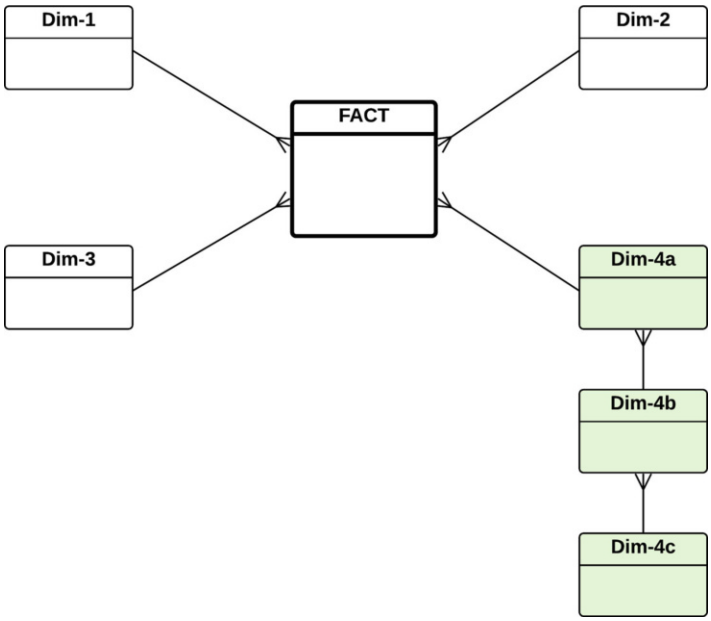

Fig. 5.3 A snowflake with a Hierarchy

In this chapter, we are going to study two case studies that highlight the importance and the use of a Bridge Table. Additional features, including a Weight Factor attribute and a List Aggregate attribute, will also be explained.

5.1 A Product Sales Case Study

To understand the concept of Bridge Tables in data warehousing, let's use a Product Sales example as a case study. The E/R diagram is shown in Fig. 5.4. The Sales entity on the left side of the E/R diagram is a transaction entity, where all sales transactions are recorded. This includes the customers and the staff who performed the sales. The Product entity, in the middle of the E/R diagram, maintains a list of the company's products. The relationship between Sales and Product is a *many-many*, through the SalesItem entity, where the QtySold attribute is recorded. Each product may be stored in several locations, as shown by the Stock entity, where the QtyinStock attribute is recorded. The Supplier entity maintains a list of suppliers. Finally, the relationship between Stock and Supplier is a *many-many*, which basically records each supply, including the supplied quantity.

In this case study, the management of the company would like to analyse the statistics of its product sales history. The analysis is needed to identify popular products, the suppliers who supplied these products, the best time to purchase more

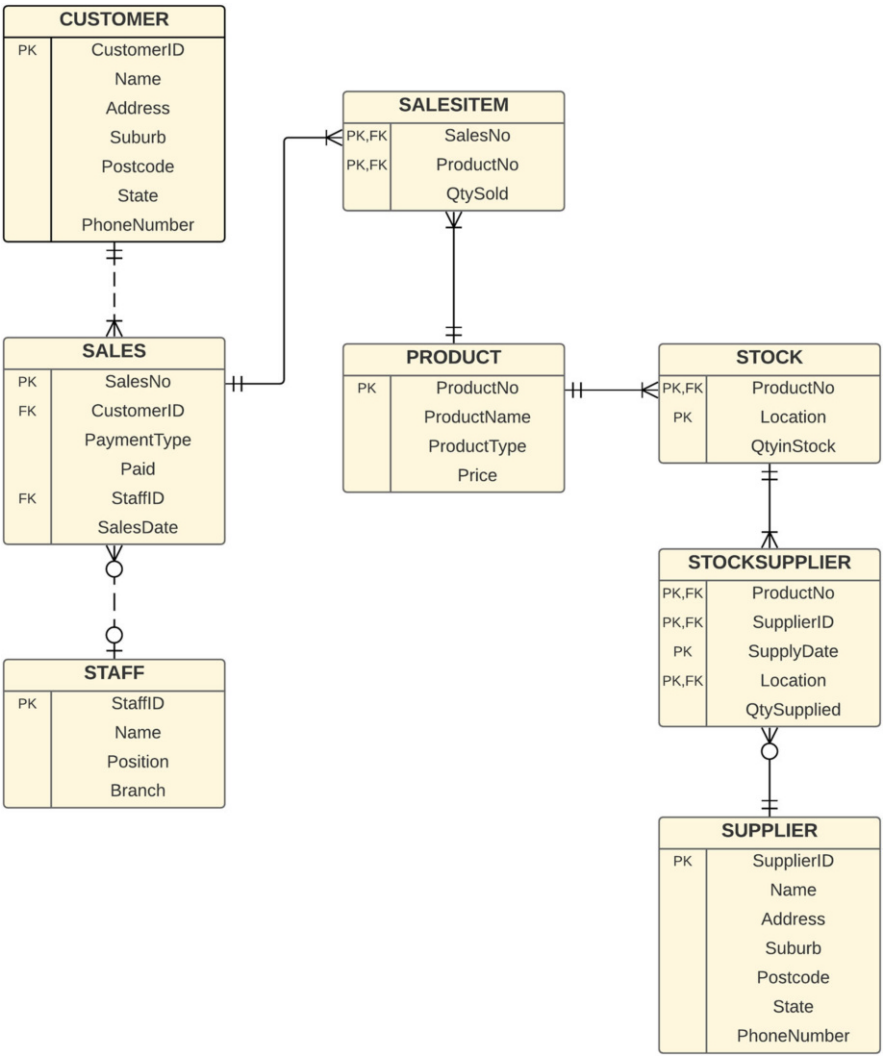

Fig. 5.4 A Product Sales E/R diagram

stock, etc. Hence, a small data warehouse is to be built to keep track of the statistics. The management is particularly interested in analysing the *total sales* (*quantity* * *price*) by *product*, *customer suburbs*, *sales time periods* (month and year) and *supplier*.

We start the data warehouse by designing two-column tables (see Tables 5.1, 5.2, and 5.3). Here are some possible scenarios (note that the total sales figures are imaginary).

The imaginary data in Tables 5.1, 5.2, and 5.3 seem to be logical, and hence a star schema for the Product Sales case study is shown in Fig. 5.5.

Table 5.1 Product point of view

ProductNo	TotalSales
A1	$130,000
B2	$15,900
C3	$2,500,000
...	...

Table 5.2 Time point of view

TimeID	TotalSales
201801	$25,000
201802	$4,700
201803	$3,500
...	...

Table 5.3 Suburb point of view

Suburb	TotalSales
Caulfield	$6,500
Chadstone	$12,000
Clayton	$1,800
...	...

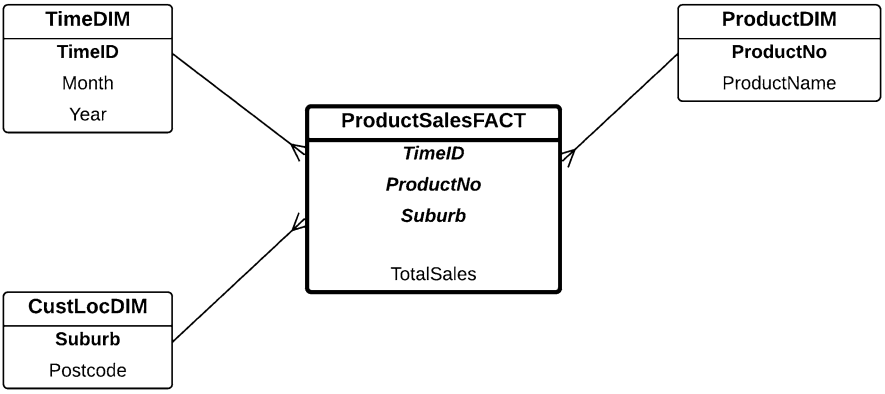

Fig. 5.5 An initial star schema

The contents of the Product Sales Fact table are shown in Table 5.4.

However, the management would also like information about the supplier which is still missing from the report in Table 5.4. Therefore, one could have another two-column table for suppliers, which may look like the one in Table 5.5, and a revised star schema is shown in Fig. 5.6 in which a new supplier dimension is added to the star schema. Therefore, the contents of the new Product Sales Fact are modified to include information about the supplier (see Table 5.6).

The revised ProductSalesFact table (with the addition of the Supplier Dimension) seems to be accurate. Actually, it is impossible to produce such a report because in the E/R diagram (in the operational database), it shows that one supplier may supply many products (or one product is supplied by many suppliers). However, the sales

Table 5.4 ProductSalesFact table

TimeID	Suburb	ProductNo	TotalSales
201801	Caulfield	A1	$450
201801	Caulfield	B2	$100
201801	Caulfield	C3	$320
201801	Caulfield	...	...
201801	...	...	...
201801	Chadstone	A1	$75
201801	Chadstone	B2	$600
201801	Chadstone	C3	$55
201801	Chadstone	...	...
201801	...	...	...
201801	Clayton	A1	$130
201801	...	...	...
201802	Caulfield	A1	$500
201802	Caulfield	B2	$430
201802	Caulfield	C3	$120
...	...	...	...

Table 5.5 Supplier point of view

SupplierID	TotalSales
S1	$77,000
S2	$5,700
S3	$12,500
...	...

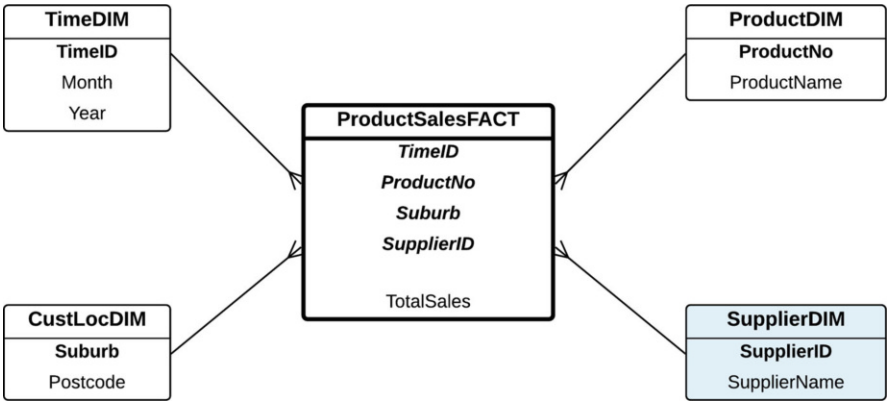

Fig. 5.6 A revised star schema

do not record which supplier supplied a product. In other words, there is no direct relationship between suppliers and total sales. Therefore, the two-column table for the supplier as illustrated in Table 5.5, although seeming to be correct, is actually wrong.

The problem will even be bigger if we add the date the supplies were delivered (e.g. the date attribute from the StockSupplier table; see the E/R diagram in Fig. 5.4).

Table 5.6 ProductSalesFact table with Supplier

TimeID	Suburb	ProductNo	SupplierID	TotalSales
201801	Caulfield	A1	S1	...
201801	Caulfield	A1	S2	...
201801	Caulfield	A1	S3	...
201801	Caulfield	A1	...	...
201801	Caulfield	B2	S1	...
201801	Caulfield	B2	S2	...
201801	Caulfield	B2	S3	...
201801	Caulfield	B2	...	...
201801	Caulfield	C3	S1	...
201801	Caulfield	C3	S2	...
201801	Caulfield	C3	S3	...
201801	Caulfield	C3	...	...
201801	...	...	...	...
201801	Chadstone	A1	S1	...
201801	Chadstone	A1	S2	...
201801	Chadstone	A1	S3	...
201801	Chadstone	A1	...	...
201801	...	...	...	...
201802	Caulfield	A1	S1	...
201802	Caulfield	A1	S2	...
201802	Caulfield	A1	S3	...
201802	Caulfield	A1	...	...
...	...	...	...	

Imagine that from the ProductSales Fact, we would like to produce a report that takes all attributes from the Fact Table and adds them to one additional column next to the SupplierID, called SupplyDate. This will create a conflict between the TimeID column and the SupplyDate column. In addition, it is also impossible to include a SupplyDate column because a one product-supplier pair may have multiple supply dates.

The reasons why this problem occurs are the following: (i) there is no direct relationship between supplier and product sales, and (ii) supplier information is not available in the sales of a particular product.

One solution is to move the Supplier Dimension from being connected to the Fact and to create a relationship with the Product Dimension, resulting a **Bridge Table** between Product Dimension and Supplier Dimension (see Fig. 5.7). Note that three additional pieces of information are added: location, date and quantity of supplies.

The aim of using a Bridge Table is to "drill down" for information on products. For example, after examining the fact report, management might be interested in drilling down further on a particular product. Perhaps the product performs extremely well in sales or for any other reasons. Hence, we can drill down for information on that product and find out, in this example, details of the supply history and the suppliers as well.

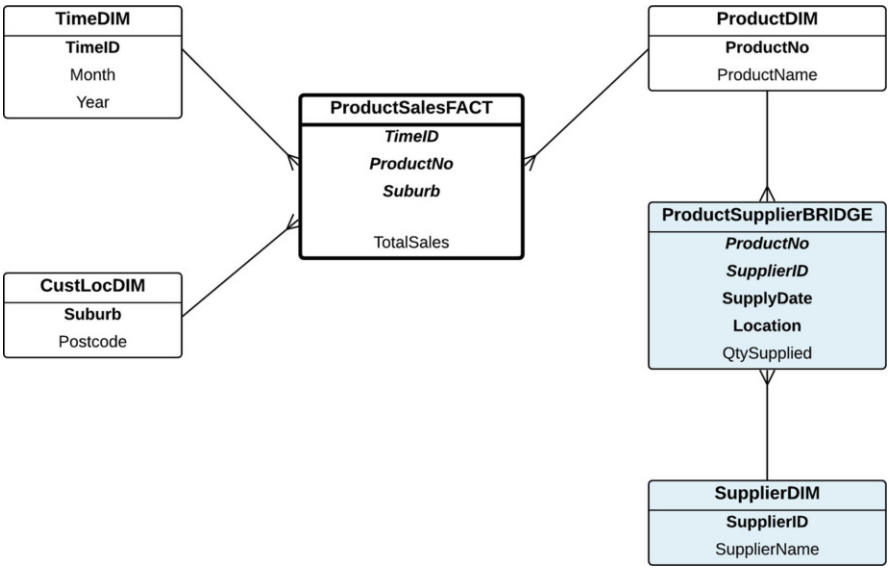

Fig. 5.7 A snowflake schema with a Bridge Table

The SQL commands to create fact and dimension tables are quite straightforward, which are as follows:

```
-- Time Dimension
create table TimeDim as
select
   distinct to_char(SalesDate, 'YYYYMM') as TimeID,
   to_char(SalesDate, 'YYYY') as Year,
   to_char(SalesDate, 'MM') as Month
from Sales;

-- Customer Location Dimension
create table CustLocDim as
select distinct Suburb, Postcode
from Customer;

-- Product Dimension
create table ProductDim as
select distinct ProductNo, ProductName
from Product;

-- Bridge Table
create table ProductSupplierBridge as
select *
from StockSupplier;

-- Supplier Dimension
create table SupplierDim as
```

```
Select SupplierID, Name as SupplierName
from Supplier;

-- Fact Table
create table ProductSalesFact as
Select
   to_char(S.SalesDate, 'YYYYMM') as TimeID,
   P.ProductNo,
   C.Suburb,
   sum(SI.QtySold*P.Price) as TotalSales
from Sales S, Product P, Customer C, SalesItem SI
where S.SalesNo = SI.SalesNo
and SI.ProductNo= P.ProductNo
and C.CustomerID = S.CustomerID
group by
   to_char(S.SalesDate, 'YYYYMM'),
   P.ProductNo,
   C.Suburb;
```

If, for example, the operational database does not maintain a master list of all suppliers (assume that each supplier may supply any product in an ad hoc manner), we will not have the Supplier Dimension in the schema (see Fig. 5.8). Instead, we will only have the Bridge Table, but for each Product-Supplier pair, we will have Location and Date implemented as a weak entity, because for each product-supplier, there are multiple supplies. Therefore, the Bridge Table acts as a *temporal dimension*. This means, for each product, there is a list of the history of supplies.

The Product Dimension table and the Bridge Table may look something like the ones in Tables 5.7 and 5.8.

In this case, the Product Supplier Bridge Table acts as a *temporal dimension* since the history of supplies is recorded.

If the history is not needed (in a non-temporal data warehouse), then the Product Supplier Bridge Table will only have ProductNo and SupplierName attributes, without the history of supplies (see Fig. 5.9).

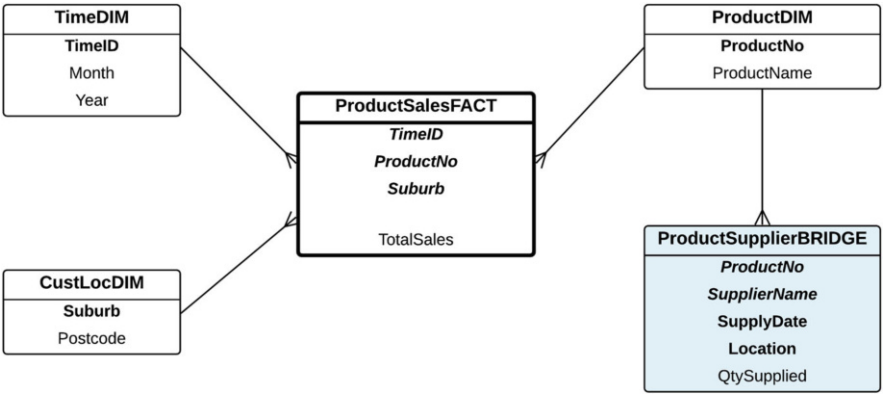

Fig. 5.8 A snowflake schema with a Bridge Table, but without a Supplier Dimension

Table 5.7 ProductDim table

ProductNo	ProductName
A1	Fandy Handbag
B2	Mercer Women Shoes
C3	Plain T-Shirt
...	...

Table 5.8 ProductSupplierBridge table

ProductNo	SupplierName	DateSupplied	Location	QtySupplied
A1	Cheap Goods Pty	21-May-2018	Caulfield	100
A1	Cheap Goods Pty	15-Aug-2018	Caulfield	150
A1	Cheap Goods Pty	19-Dec-2018	Clayton	50
A1	Cheap Goods Pty	...	...	...
A1	Just Bags	21-May-2018	Chadstone	200
A1	Just Bags	30-Jun-2018	Clayton	80
A1	JustBags	...	...	...
A1	Baggy	...	...	...
A1	...	...	...	...
B2	Cheap Goods Pty	...	...	...
...	...	...	...	...

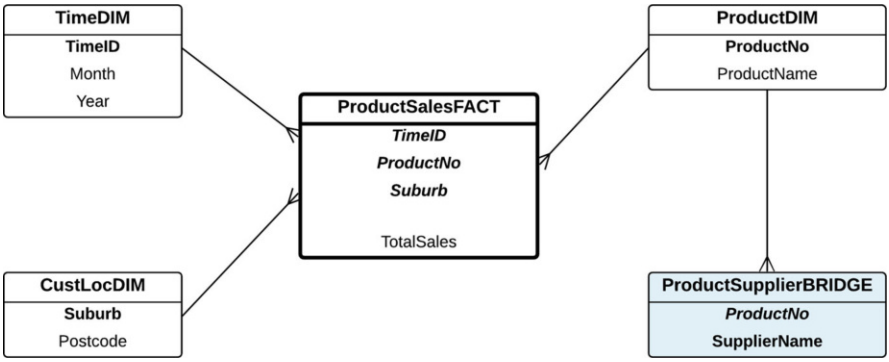

Fig. 5.9 A snowflake schema with a Bridge Table, but without maintaining the history of supplies

5.2 A Truck Delivery Case Study

A trucking company is responsible for picking up goods from the warehouses of a retail chain company and delivering the goods to individual retail stores. A truck carries goods on a single trip, which is identified by TripID and delivers these goods to multiple stores. Trucks have different capacities for both the volumes they can hold and the weights they can carry. Currently, a truck makes several trips each week. An operational database is being used to keep track of the deliveries,

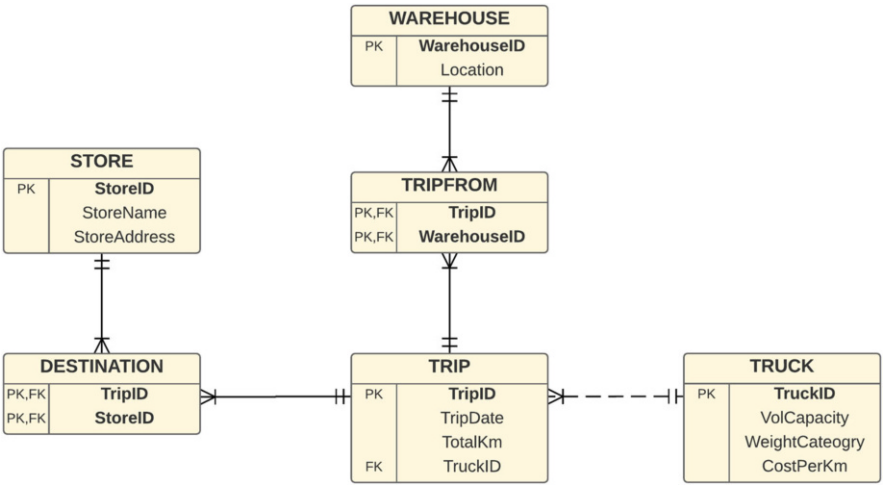

Fig. 5.10 An E/R diagram of the truck delivery system

including the scheduling of trucks to provide timely deliveries to stores. The E/R diagram of the truck delivery system is shown in Fig. 5.10.

Based on the E/R diagram, (*i*) a trip may pick up goods from many warehouses (i.e. a *many-many* relationship between Warehouse and Trip); (*ii*) a trip uses one truck only, and obviously a truck may have many trips in the history (i.e. a *many-1* relationship between Trip and Truck); and (*iii*) a trip delivers goods (e.g. TVs, fridges, etc.) potentially to several stores (a *many-many* relationship between Trip and Store, which is represented by the Destination table).

The tables from the operational database are as follows:

- **Warehouse**(WarehouseID, Location)
- **Trip**(TripID, Date, TotalKm, *TruckID*)
- **TripFrom**(*TripID*, *WarehouseID*)
- **Truck**(TruckID, VolCapacity, WeightCategory, CostPerKm)
- **Store**(StoreID, StoreName, Address)
- **Destination**(*TripID*, *StoreID*)

Some sample data in the operational database are shown in Tables 5.9, 5.10, 5.11, 5.12, 5.13, and 5.14.

Table 5.9 Warehouse table

WarehouseID	Location
W1	Warehouse1
W2	Warehouse1
W3	Warehouse1
...	...

Table 5.10 Trip table

TripID	Date	TotalKm	TruckID
Trip1	14-Apr-2018	370	Truck1
Trip2	14-Apr-2018	570	Truck2
Trip3	14-Apr-2018	250	Truck3
Trip4	15-Apr-2018	450	Truck1
...	...	...	...

Table 5.11 TripFrom table

TripID	WarehouseID
Trip1	W1
Trip1	W2
Trip1	W3
Trip2	W1
Trip2	W2
...	...

Table 5.12 Truck table

TruckID	VolCapacity	WeightCategory	CostPerKm
Truck1	250	Medium	$1.20
Truck2	300	Medium	$1.50
Truck3	100	Small	$0.80
Truck4	550	Large	$2.30
Truck5	650	Large	$2.50
...	...	...	...

Table 5.13 Store table

StoreID	StoreName	Address
M1	MyStore City	Melbourne
M2	MyStore Chaddy	Chadstone
M3	MyStore HiPoint	High Point
M4	MyStore Westfield	Doncaster
M5	MyStore North	Northland
M6	MyStore South	Southland
M7	MyStore East	Eastland
M8	MyStore Knox	Knox City
...	...	...

The management of this trucking company would like to analyse the delivery cost based on *trucks*, *time period* and *store*.

5.2.1 Solution Model 1: Using a Bridge Table

The fact measure to be included in the Fact Table is “Total Delivery Cost”, which is calculated by *distance* (total kilometres in the Trip table) and *cost per kilometre*

Table 5.14 Destination table

TripID	StoreID
Trip1	M1
Trip1	M2
Trip1	M4
Trip1	M3
Trip1	M8
Trip2	M4
Trip2	M1
Trip2	M2
...	...

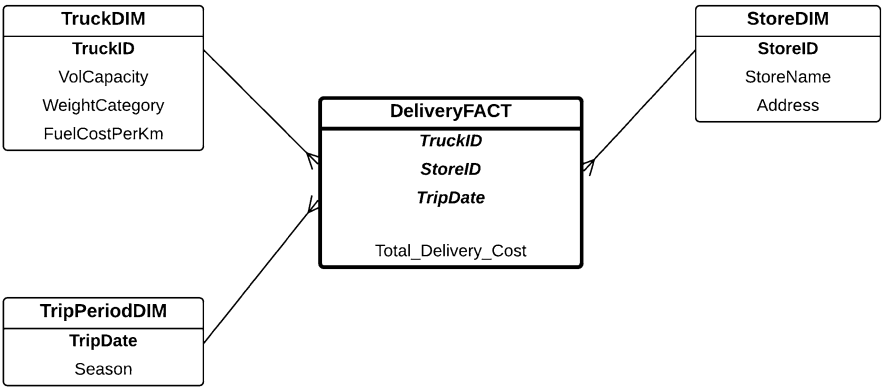

Fig. 5.11 A possible truck delivery star schema

(from the Truck table). The dimensions are Truck, Trip Period and Store. A possible star schema is shown in Fig. 5.11.

Based on the sample data as shown previously in Tables 5.9, 5.10, 5.11, 5.12, 5.13, and 5.14:

- From the Truck point of view, Truck1 makes two trips (e.g. Trip1 and Trip4), travelling a total of 820 km (370 km + 450 km). The cost for Truck1 is $1.20/km. Hence, calculating the cost for Truck1 is straightforward. The cost of the other trucks can also be calculated this way.
- From the Period point of view (say from a date point of view), three trips were made on 14 April 2018 (e.g. Trip1, Trip2 and Trip3). Trip1 (370 km) is delivered by Truck1 which costs $1.20/km. Trip2 and Trip 3 on the same day can be calculated in the same way. Hence, on 14 April 2018, the total cost can be calculated.
- From the Store point of view, the cost is calculated based on Trip, but a trip delivers goods to many stores. Therefore, the delivery cost for each store cannot be calculated. The delivery cost is for the trip, not for the store.

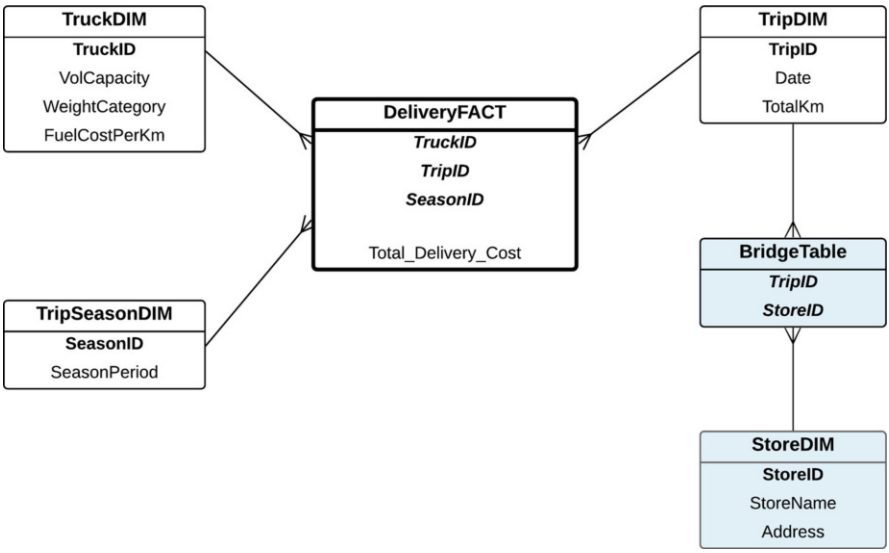

Fig. 5.12 A correct truck delivery snowflake schema with a Bridge Table

Therefore, the star schema as shown in Fig. 5.11 may look correct, but it is actually **incorrect**, because there is **no** direct relationship between Store and Total Delivery Cost in the Fact Table. This is due to the *many-many* relationship between the Trip entity and Store entity in the E/R diagram of the operational database. To solve this problem, a Bridge Table can be used, as shown in Fig. 5.12.

5.2.2 Solution Model 2: Add a Weight Factor Attribute

The Solution Model 1 shown in the previous section demonstrates that there is no direct relationship between Total Delivery Cost and Store. Total delivery cost is calculated at a Trip level and not at a Store level. For example, The length of Trip 1 was 370 km on 14 April using Truck1, and the cost per kilometre for Truck1 is $1.20. Hence, Trip1 delivered goods to five stores at a cost of $444. Using the current data that we have, it is impossible to calculate the delivery cost for each of these five stores for Trip1. What we can say is that the cost for Trip1 is $\$x$ and Trip1 delivered to y number of stores. However, we can estimate the total delivery cost per store, if we want to. This can be estimated through the "Weight Factor" (see the Weight Factor attribute in the Trip Dimension in Fig. 5.13).

A *weight factor* is a proportion of the trip that goes to each store for that particular trip. For example, if Trip1 went to five stores, then the Weight Factor is 0.2 (or 20%). This implies that each store "contributes" 20% of the total delivery cost for that trip. This is certainly inaccurate, but this is the only estimate that we can make, based on the data that we have.

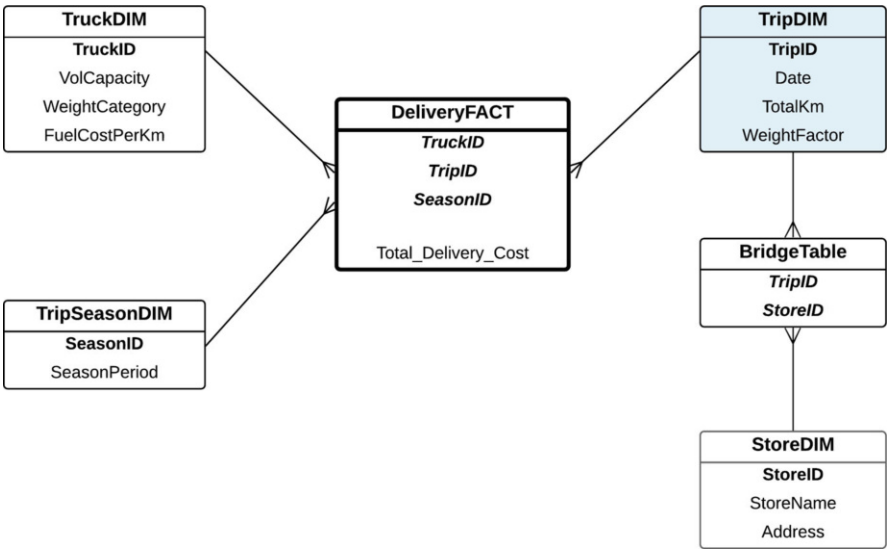

Fig. 5.13 A truck delivery snowflake schema with a Weight attribute

Table 5.15 Trip Dimension table

TripID	Date	TotalKm	WeightFactor
Trip1	14-Apr-2018	370	0.20
Trip2	14-Apr-2018	570	0.33
...	...	...	...

Table 5.16 Bridge Table

TripID	StoreID
Trip1	M1
Trip1	M2
Trip1	M4
Trip1	M3
Trip1	M8
Trip2	M4
Trip2	M1
Trip2	M2
...	...

Not all snowflake schemas with Bridge Tables must have a weight factor. A weight factor is only needed if we want to estimate the contribution that a dimension made to the fact. Using the star schema in Fig. 5.11, if a weight factor is not used, then we will not be able to estimate the total delivery cost per store. Is missing this information very critical? It depends. After all, the information on total delivery cost per store is an estimated figure anyway.

The data in the Trip, Bridge Table and Store Dimensions are shown in Tables 5.15, 5.16, and 5.17.

Table 5.17 Store table

StoreID	StoreName	Address
M1	MyStore City	Melbourne
M2	MyStore Chaddy	Chadstone
M3	MyStore HiPoint	High Point
M4	MyStore Westfield	Doncaster
M5	MyStore North	Northland
M6	MyStore South	Southland
M7	MyStore East	Eastland
M8	MyStore Knox	Knox City
...	...	...

Table 5.18 Trip Dimension table

TripID	Date	TotalKm
Trip1	14-Apr-2018	370
Trip2	14-Apr-2018	570
...	...	...

Table 5.19 Bridge Table

TripID	StoreID	WeightFactor
Trip1	M1	0.20
Trip1	M2	0.20
Trip1	M4	0.20
Trip1	M3	0.20
Trip1	M8	0.20
Trip2	M4	0.33
Trip2	M1	0.33
Trip2	M2	0.33
...	...	...

The 0.2 weight factor for Trip1 does not mean that Trip1 has 20% of the total delivery cost; rather, Trip1 delivers to five stores, and each store contributes 20% (or 0.20) to the total delivery cost. If this is what it means, why not have the Weight Factor attribute in the Bridge Table, as shown in Tables 5.18 and 5.19, to indicate, for example, that store M1 (from Trip 1) contributes 20% (0.20) to the total delivery cost of Trip1? The answer is yes, we could do that, but then the Bridge Table will contain redundant information, because all the stores in Trip1 should have a weight factor of 20%. So, instead of storing the 0.20 in each of the stores for Trip1, we store the 0.20 in the Trip1 record itself in the Trip Dimension table (refer to Table 5.15).

Creating the Trip Dimension table with the WeightFactor attribute is done using the following SQL:

```
create table TripDim2 as
select T.TripID, T.TripDate, T.TotalKm,
   1.0/count(*) as WeightFactor
from Trip T, Destination D
where T.TripID = D.TripID
group by T.tripid, T.tripdate, T.totalkm;
```

5.2.3 *Solution Model 3: A List Aggregate Version*

In the *ListAgg* (short for *List Aggregate*) option, we have one additional attribute in the parent dimension (e.g. the Trip Dimension) which holds the information on the group of each record in the parent dimension table (e.g. stores within each trip). Figure 5.14 includes an attribute called "StoreGroupList" in the Trip Dimension. In this attribute, all stores for each trip are concatenated to become a store group list.

The contents of the Trip Dimension and the Bridge Table are shown in Tables 5.20 and 5.21.

To create the Trip Dimension table with a StoreGroupList attribute, we can use the `listagg` function in SQL. The `listagg` function has the following format:

```
listagg (Attr1, '_') within group
   (order by Attr1) as ColumnName
```

where Attr1 is the StoreID and the "_" indicates that the StoreIDs are concatenated with the "_" symbol (e.g. M1_M2_M3_M4_M8). If we want to list the stores listed in descending order (e.g. M8_M4_M3_M2_M1), then we use `within`

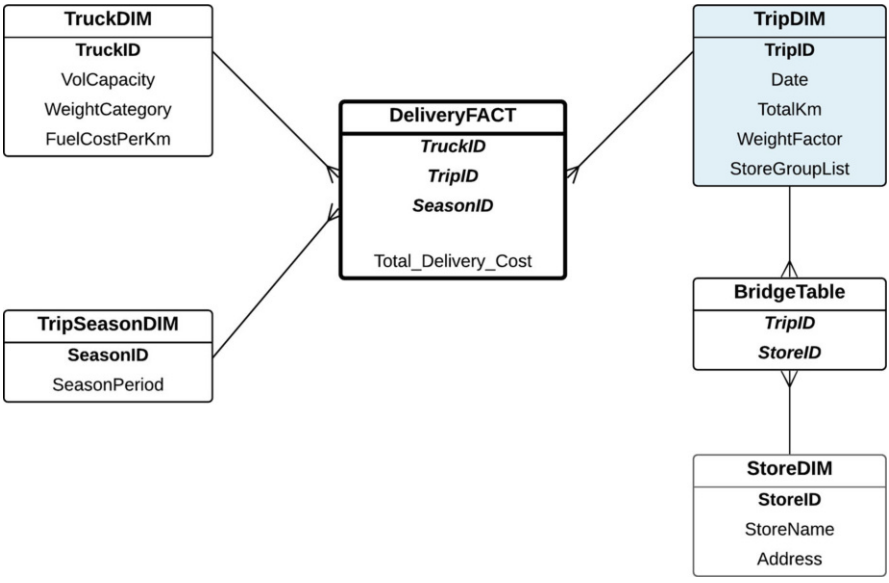

Fig. 5.14 A truck delivery snowflake schema with a StoreGroupList attribute

Table 5.20 Trip Dimension table

TripID	Date	TotalKm	WeightFactor	StoreGroupList
Trip1	14-Apr-2018	370	0.20	M1_M2_M3_M4_M8
Trip2	14-Apr-2018	570	0.33	M1_M2_M4
...	...	...	...	...

Table 5.21 Bridge Table

TripID	StoreID
Trip1	M1
Trip1	M2
Trip1	M4
Trip1	M3
Trip1	M8
Trip2	M4
Trip2	M1
Trip2	M2
...	...

`group (order by Attr1 Desc)`. The SQL to create the Trip Dimension table becomes:

```
create table TripDim3 as
select T.TripID, T.TripDate, T.TotalKm,
   1.0/count(D.StoreID) as WeightFactor,
   listagg (D.StoreID, '_') within group
   (order by D.StoreID) as StoreGroupList
from Trip T, Destination D
where T.TripID = D.TripID
group by T.TripID, T.TripDate, T.TotalKm;
```

Visually, the List Aggregate attribute (e.g. the StoreGroupList attribute in the Trip Dimension) is appealing because it is very easy to see the complete list of stores for each trip. However, this information is rather redundant because we can get the same information from the Bridge Table (see Table 5.21).

The previous two snowflake schemas for the Truck Delivery System are actually a non-List Aggregate version. A non-List Aggregate version is simpler and cleaner. The implementation is simpler and more straightforward. Additionally, there is no redundant information on the stores within each trip.

However, in industry, the List Aggregate version is often a preferred option. This is because a group list is physically listed in the parent dimension, which may visually help the decision-makers to understand the completeness of the group list (e.g. StoreID for each Trip). Unfortunately, from an implementation point of view, creating a List Aggregate can be very complex. Nevertheless, the List Aggregate version is not uncommon. Additionally, when producing a report, if we would like to join the Trip and Store Dimensions, the List Aggregate attribute only needs two tables, the Trip and Store Dimension tables, without the need to join them with the Bridge Table.

```
select *
from TripDim3 T, StoreDim3 S
where T.StoreGroupList like '%'||S.StoreID||'%';
```

In this SQL, the joining is based on the StoreGroupList attribute in the Trip Dimension table and the StoreID in the Store Dimension table. However, the join condition is not a simple `T.StoreGroupList = S.StoreID` because the join

condition is not a simple match. Instead, we use a `like` operator to check if the StoreID value exists in the StoreGroupList.

In contrast, without the StoreGroupList attribute in the Trip Dimension, we need to join three tables, the Trip Dimension, the Bridge Table and the Store Dimension tables, using the following SQL:

```
select *
from TripDim3 T, BridgeTable3 B, StoreDim3 S
where T.TripID = B.TripID
and B.StoreID = S.StoreID;
```

5.3 Summary

In principle, a Bridge Table is used:

(a) When it is impossible to have a dimension connected directly to the Fact Table, because simply there is no relationship between this dimension and the Fact Table (e.g. in the Product Sales case study, it is impossible to have a direct link from SupplierDim to ProductSalesFact)
(b) When an entity (which will become a dimension) has a *many-many* relationship with another entity (dimension) in the E/R schema of the operational database (e.g. Supplier and Stock has a *many-many* relationship).
(c) When the temporality aspect (data history) is maintained in the operational database and the Bridge Table can be used to accommodate a dimension that has temporal attributes (e.g. product supply history is maintained in the second snowflake schema example).

When a Bridge Table is used in the schema, there are two additional options:

(a) A Weight Factor is used to estimate the contribution of a dimension in the calculation of the fact measure. Because this is only an estimate, a weight factor is an option.
(b) Every snowflake schema (whether it has Weight Factor or not) can be implemented in two ways: a List Aggregate version and a non-List Aggregate version.

5.4 Exercises

5.1 Figure 5.15 is an E/R diagram of an operational database. The system stores information about books, including the authors, publishers, book categories as well as the reviews that each book has received. The "stars" attribute in the Review entity records the star rating for each review (e.g. 5 stars for excellent to 1 star for poor, etc.). One book may receive many reviews. For simplicity, it is assumed, as also shown in the E/R diagram, that a book will only have one category.

The E/R diagram also includes entities related to the sale of books and the stores which sell the books. Each store has many sales transactions (i.e. the Sales

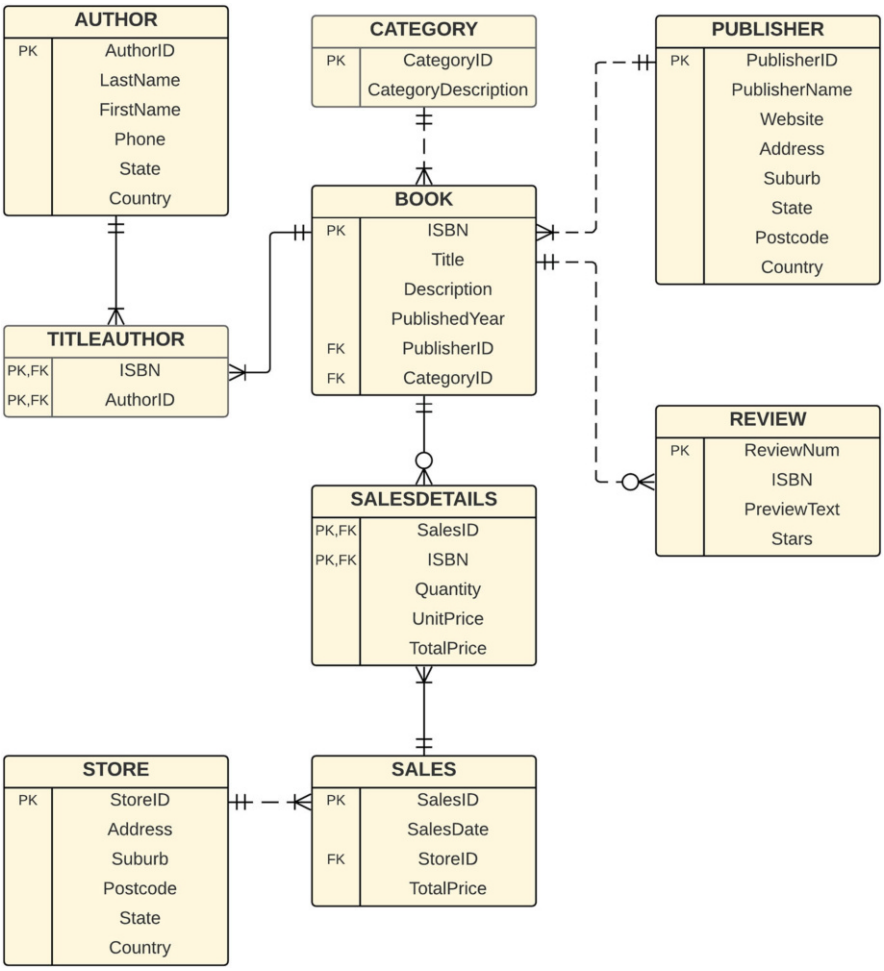

Fig. 5.15 A book sales system

entity), and each sales transaction may include several books (i.e. the SalesDetails entity). The TotalPrice attribute in the SalesDetails entity is quantity multiplied by UnitPrice, whereas the TotalPrice attribute in the Sales entity is the total price for each sales transaction.

You are required to design a small data warehouse for analysis purposes. The analysis is needed to answer at least the following questions:

- What are the total sales for each store in a month?
- How many books are sold in each category?
- Which book category has the highest sales?
- Which author has sold the highest number of books?

Write the SQL commands to create the dimensions and Fact Tables.

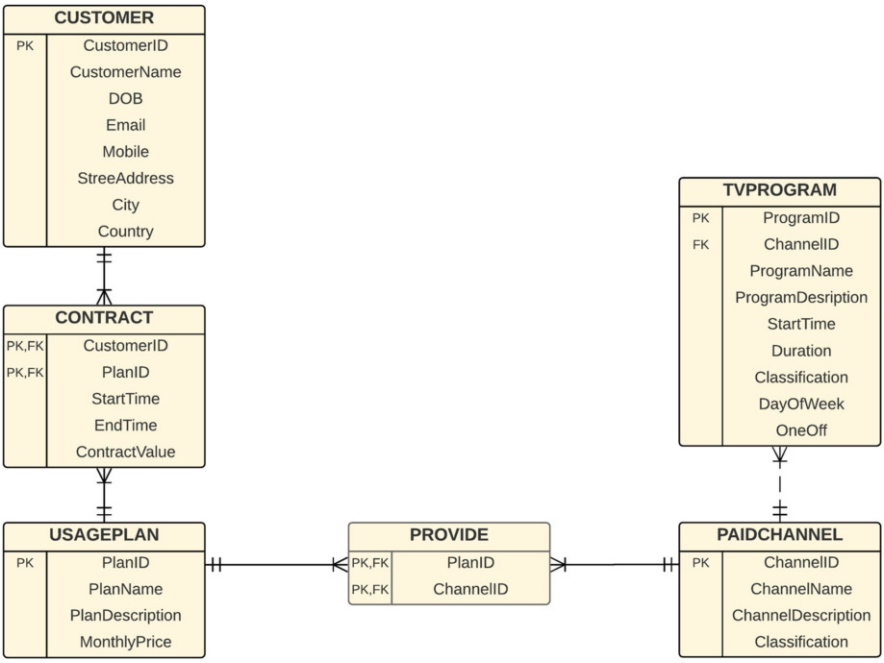

Fig. 5.16 A cable television system

5.2 The Program Manager of a Cable Television company is interested in analysing the statistics on the revenue it receives from paid channel subscriptions. The analysis is needed to identify which paid TV channels are more attractive to users than the others. The results of the analysis will be useful to determine which channels to purchase in the following years. You are required to design a small Data Warehouse to keep track of the statistics. The director is particularly interested in analysing the *total number of contracts* and *total revenue* (total contract value) by channels, location, year and month (as in contract start time). The E/R diagram of the operational database is shown in Fig. 5.16.

Based on the Program Manager's requirements, develop a snowflake schema. If you are using a Bridge Table, make sure you include a Weight Factor attribute and a List Aggregate attribute in your design. Write the queries to create (and populate) the dimension tables and the Fact Table.

5.3 The Rural University of Victoria (RUV) has a number of campuses in several cities and towns in the state of Victoria. Each campus has several departments. Staff from one campus may need to travel to a different campus for various reasons, for example to teach, to attend administrative meetings or to conduct collaborative research with staff on another campus.

To facilitate the travel between campuses, some departments have vehicles which staff may borrow. In addition to this, the university also has a fleet of vehicles which staff may borrow if the department's cars are booked.

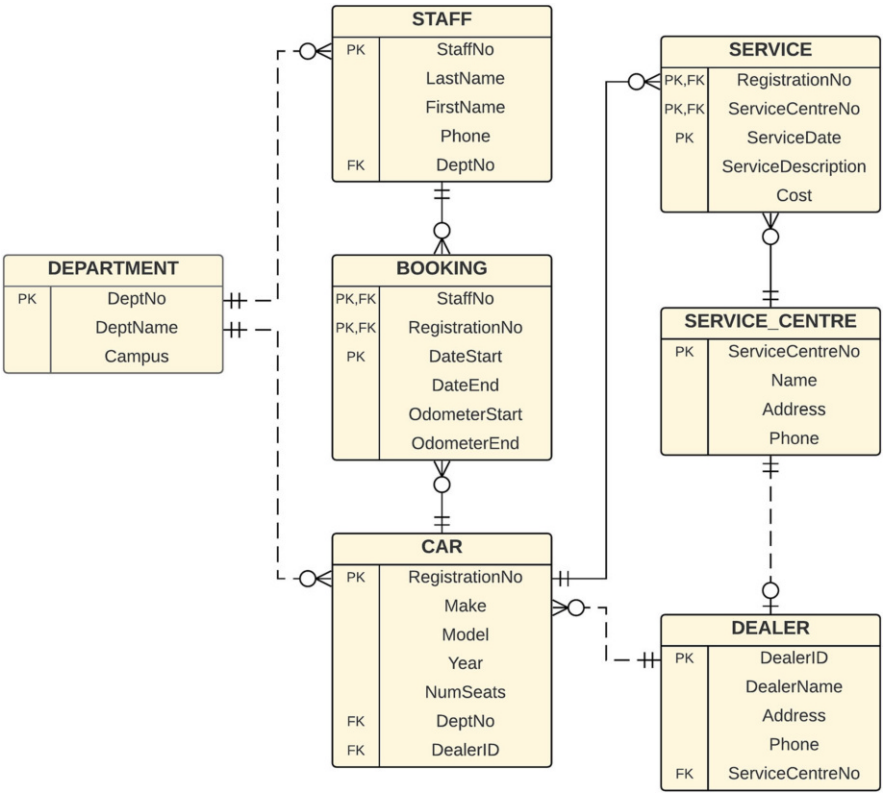

Fig. 5.17 A car booking system

These cars need a regular service, and from time to time, they also require non-regular maintenance, such as new tires, replacement batteries, etc.

The E/R diagram of the operational database is shown in Fig. 5.17.

The university now requires a data warehouse to analyse the cost of car services and service costs and to answer at least the following questions:

- How much is the total service cost in each month/year?
- How many times is a car serviced each month?
- How much is the total service cost for each department?
- Approximately, how much of the total service cost is incurred by each user?

Based on the above requirements, develop a snowflake schema with a Bridge Table. The snowflake schema must also include a Weight Factor attribute and a List Aggregate attribute. Write the queries to create (and populate) the dimension tables and the Fact Table.

5.4 This question is related to the Car Service System, shown in Fig. 5.18. Every time a service is conducted, a record is entered into the database. The information

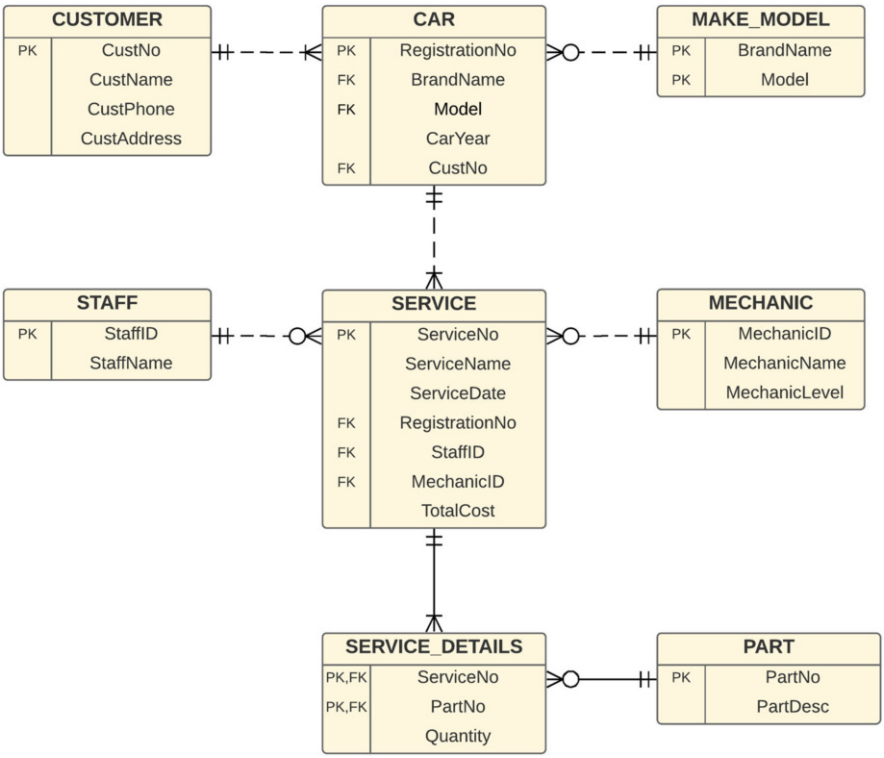

Fig. 5.18 A Car Service System

recorded includes service number, service name, car registration number, the name of the staff member who booked the service and liaised with the customer, the mechanic who performed the repair and the total cost of the repair.

Questions

(a) A simple star schema for the Car Service System has been designed and is shown in Fig. 5.19. It is a simple star schema with four dimensions: Time, Brand, Mechanic and Part Dimensions. The fact measure is the Number of Services. The Fact Table is created by a join operation of the Service, Service Details and Car tables in the operational database.

```
create table CarServiceFact as
select
  to_char(S.ServiceDate, 'YYYYMM') as TimeID,
  C.BrandName,
  S.MechanicID,
  D.PartNo,
  count(S.ServiceNo) as Number_of_Services
from Service S, Service_Details D, Car C
where S.ServiceNo = D.ServiceNo
and S.RegistrationNo = C.RegistrationNo
```

```
group by
  to_char(S.ServiceDate, 'YYYYMM'),
  C.BrandName,
  S.MechanicID,
  D.PartNo;
```

Your task is to create all the dimension tables. Is the Fact Table created correctly? Analyse the contents of the Fact Table by analysing the results of the following queries:

- Pick a Part, and then retrieve the Number of Services for this Part. Check this query result manually against the operational database.
- Pick a Mechanic, and then retrieve the Number of Services by this Mechanic. Again, check the query result against the operational database.
- Now use the Part and Mechanic, and retrieve the Number of Services for this Part and Mechanic. Check your query result against the operational database.
- Lastly, retrieve the Number of Services from the Fact Table. Check this manually against the number of services in the operational database

If the Fact Table is incorrect, explain why the star schema is incorrect, and then redesign the correct star schema.

(b) Build another star schema, but this time to analyse the Total Service Cost, also based on each month, car brand, mechanic and part. Note that every service may use several different parts. Write the SQL commands to create the dimension and Fact Tables.

(c) Looking at the above two star schemas, one for Number of Services (see Fig. 5.19) and the other for Total Service Cost (your star schema from question (*b*)), what is the main difference, and what/where was the mistake?

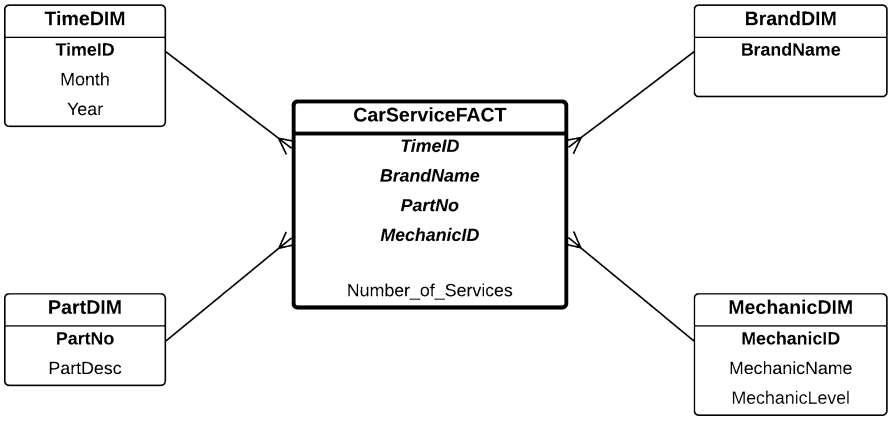

Fig. 5.19 A star schema for the Car Service System

5.5 Further Readings

This chapter focuses solely on the concept of *Bridge Tables* in star schemas. A Bridge Table is actually a *many-many* relationship between two dimension tables. The *many-many* relationships have been used extensively in relational database design and modelling. They can be found in typical database and data modelling textbooks, such as [1–21] and [22]

Aggregate functions, such as `listagg` and other aggregate functions, can be found in many SQL books, such as [23–25] and [26].

String subset join and other non-equi join queries are explained in various SQL books and database textbooks, such as [7–11, 13–15, 23–25] and [26].

Basic join algorithms, such as Nested-Loop, Sort-Merge and Hash Join algorithms, are explained in [27]. Non-equi join algorithms are less often discussed. Some earlier work on non-equi join and band join algorithms can be found in [28, 29] and [30].

References

1. T.A. Halpin, T. Morgan, *Information Modeling and Relational Databases*, 2nd edn. (Morgan Kaufmann, Los Altos, 2008)
2. S. Bagui, R. Earp, *Database Design Using Entity-Relationship Diagrams*. Foundations of Database Design (CRC Press, Boca Raton, 2003)
3. M.J. Hernandez, *Database Design for Mere Mortals: A Hands-on Guide to Relational Database Design*. For Mere Mortals (Pearson Education, 2013)
4. N.S. Umanath, R.W. Scamell, *Data Modeling and Database Design* (Cengage Learning, 2014)
5. T.J. Teorey, S.S. Lightstone, T. Nadeau, H.V. Jagadish, *Database Modeling and Design: Logical Design*. The Morgan Kaufmann Series in Data Management Systems (Elsevier, Amsterdam, 2011)
6. G. Simsion, G. Witt, *Data Modeling Essentials*. The Morgan Kaufmann Series in Data Management Systems (Elsevier, Amsterdam, 2004)
7. C. Coronel, S. Morris, *Database Systems: Design, Implementation, & Management* (Cengage Learning, 2018)
8. T. Connolly, C. Begg, *Database Systems: A Practical Approach to Design, Implementation, and Management* (Pearson Education, 2015)
9. J.A. Hoffer, F.R. McFadden, M.B. Prescott, *Modern Database Management* (Prentice Hall, 2002)
10. A. Silberschatz, H.F. Korth, S. Sudarshan, *Database System Concepts*, 7th edn. (McGraw-Hill, New York, 2020)
11. R. Ramakrishnan, J. Gehrke, *Database Management Systems*, 3rd edn. (McGraw-Hill, New York, 2003)
12. J.D. Ullman, J. Widom, *A First Course in Database Systems*, 2nd edn. (Prentice Hall, Englewood Cliffs, 2002)
13. H. Garcia-Molina, J.D. Ullman, J. Widom, *Database Systems: The Complete Book* (Pearson Education, 2011)
14. P.E. O'Neil, E.J. O'Neil, *Database: Principles, Programming, and Performance*, 2nd edn. (Morgan Kaufmann, Los Altos, 2000)

15. R. Elmasri, S.B. Navathe, *Fundamentals of Database Systems*, 3rd edn. (Addison-Wesley-Longman, 2000)
16. C.J. Date, *An Introduction to Database Systems*, 7th edn. (Addison-Wesley-Longman, 2000)
17. R.T. Watson, *Data Management—Databases and Organizations*, 5th edn. (Wiley, London, 2006)
18. P. Beynon-Davies, *Database Systems* (Springer, Berlin, 2004)
19. E.F. Codd, *The Relational Model for Database Management, Version 2* (Addison-Wesley, Reading, 1990).
20. J.L. Harrington, *Relational Database Design Clearly Explained*. Clearly Explained Series (Morgan Kaufmann, Los Altos, 2002)
21. M. Kifer, A.J. Bernstein, P.M. Lewis, *Database Systems: An Application-oriented Approach* (Pearson/Addison-Wesley, 2006)
22. W. Lemahieu, S. vanden Broucke, *Principles of Database Management: The Practical Guide to Storing, Managing and Analyzing Big and Small Data* (Cambridge University Press, Cambridge, 2018)
23. J. Melton, *Understanding the New SQL: A Complete Guide*, vol. I 2nd edn. (Morgan Kaufmann, Los Altos, 2000)
24. C.J. Date, *SQL and Relational Theory—How to Write Accurate SQL Code*, 2nd edn. Theory in Practice (O'Reilly, 2012)
25. A. Beaulieu, *Learning SQL: Master SQL Fundamentals* (O'Reilly Media, 2009)
26. M.J. Donahoo, G.D. Speegle, *SQL: Practical Guide for Developers*. The Practical Guides (Elsevier, Amsterdam, 2010)
27. D. Taniar, C.H.C. Leung, W. Rahayu, S. Goel, *High Performance Parallel Database Processing and Grid Databases* (Wiley, London, 2008)
28. J. Van den Bercken, B. Seeger, P. Widmayer, The bulk index join: a generic approach to processing non-equijoins, in *Proceedings of the 15th International Conference on Data Engineering, Sydney, Australia, March 23–26, 1999*, ed. by M. Kitsuregawa, M.P. Papazoglou, C. Pu (IEEE Computer Society, 1999), p. 257
29. P. Bouros, K. Lampropoulos, D. Tsitsigkos, N. Mamoulis, M. Terrovitis, Band joins for interval data, in *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, ed. by A. Bonifati, Y. Zhou, M.A.V Salles, A. Böhm, D. Olteanu, G.H.L. Fletcher, A. Khan, B. Yang (2020), pp. 443–446. OpenProceedings.org
30. H. Lu, K.L Tan, On sort-merge algorithm for band joins. IEEE Trans. Knowl. Data Eng. **7**(3), 508–510 (1995)