Topic 3.3: Star Schema Design

This module presents the star schema, an alternative to 3NF schemas intended for analytical databases.

Star Schema Overview

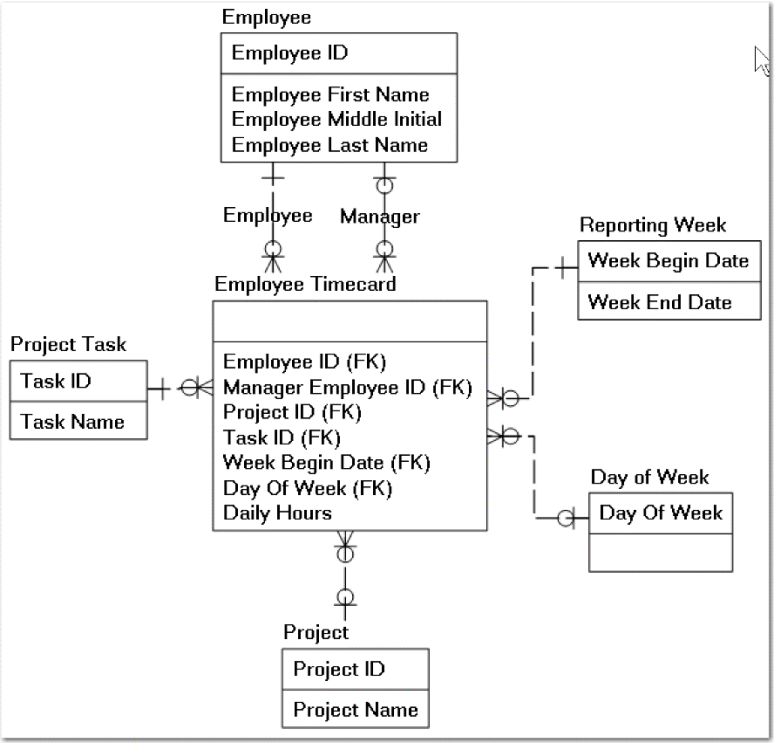
The star schema is a simple database architecture that is used extensively in analytical applications, particularly data marts. (A detailed presentation of data marts is included in Module 8.) In the late 1980s, Ralph Kimball popularized the star schema while he was the CEO of Redbrick Systems. Their flagship product, Redbrick, was the first commercial implementation of the star schema.

The star schema divides tables (relations) into fact tables and dimension tables. Each fact table is associated with one or more dimension tables using one-to-many relationships (each row in a dimension table can have many associated rows in a corresponding fact table). Dimension tables are associated only with fact tables – in a pure star schema, dimension tables do not participate in relationships with other dimension tables. (I will elaborate more on this point later in this topic.) Star schema diagrams are often drawn with a fact table in the center, surrounded by dimension tables, which forms a star pattern, and hence the name star schema.

- Fact tables contain *facts*, which are quantitative measures of a business process, such as Purchase Quantity or Amount Paid. Facts are generally numeric and cumulative, which means they can be summed without losing business meaning.
- Dimension tables contain *dimension attributes* (or simply *attributes*), which characterize or describe the facts in some way, such as Purchase Date, Product Code, and Product Description. Dimensions provide the business context for the facts, and are typically used for grouping, sorting and filtering the data contained in fact tables.

Normalization rules are generally not applied to star schemas because analytical applications work with historic data, which is usually not updated. In addition, data integrity is usually not enforced. The apparent lax design rules mortify database designers who are used to normalized data structures that support processing transactions. However, this is a different world where there are no business transactions. In fact, the data contained in analytical databases are typically loaded from the operational databases that support the business transactions, and once records are loaded, they are never updated – if changes must be made, the analytical data is merely reloaded with the corrected data.

Using the employee timecard example, here is a star schema model that could be used for data analytics.



Compared with the 3NF design for the Employee Timecard application that you saw in the last module, you may have noticed these differences (in addition to the placement of the tables on the diagram):

- The recursive relationship on the Employee table has been removed. Instead, a second relationship between Employee and Employee Timecard has been added that places the foreign key for the employee's manager in the Employee Timecard table. This was done to comply with the star schema rule that dimensions can only have relationships with fact tables.
- The Project table has been related directly with the Employee Timecard table (the fact table) instead of the Project Task table, which is another dimension. As before, this is to comply with the star schema rule that dimensions only have relationships with fact tables. This is one alternative solution when dimensions form a hierarchy, commonly known as *splitting* the hierarchy. We'll see another alternative shortly.
- Day of Week was implemented as a dimension table. As you will see shortly, this is not mandatory.
- The Employee Timecard table has no primary key. In star schema designs, fact tables normally do not have formally defined primary keys. The unique identifier is assumed to be the combination of all the foreign keys for all the dimensions connected to the fact table. However, with analytical applications, enforcing constraints is not important.

Fact Table Design

Recall that facts are measures taken from business processes that will be used as data analysis metrics. Another common term for a metric is a *key performance indicator* (KPI). Facts are nearly

always numeric, but that doesn't mean that all numeric attributes are facts. For example, an SSN is a numeric value, but it certainly is not a fact because it doesn't measure anything and it makes no sense to perform arithmetic operations on SSNs (for example, addition, multiplication, etc.) or statistical operations (for example, standard deviations). Facts can be divided into these categories:

- **Additive:** values can be summed without losing business meaning. For example, we can sum the hours worked across any or all of the dimensions in the employee timecard example (project, task, employee, etc.) without losing any meaning whatsoever. You should strive to include only additive facts in your fact tables.
- **NonAdditive:** values cannot be summed without losing business meaning. For example, if we added Employee Hourly Rate (the rate of pay per hour the employee earns) to the Employee Timecard fact table, it would be a nonadditive fact because you cannot add the hourly rate one employee earns with the hourly rate another employee earns and arrive at a meaningful data value. With nonadditive facts, especially prices, rates and percentages, you can usually break the fact down into components that are additive. In this case, we could multiply the hourly rate for the employee by the number of hours worked to arrive at the Gross Pay value (total amount earned in dollars and cents, or another currency), and Gross Pay is an additive fact.
- **Semi-Additive**: values can only be summed within some known context. For example, if the fact table holds financial account balances as of the end of each statement period (typically each month), you can add the account balances within one period to determine your total net worth. However, you cannot add the balance from the end of one period with the balance from the end of the next period because, in most cases, some of the funds that were in the account in one period are still there in the next period, and thus you would be double-counting those funds. For example, if I had $1,000 in a savings account at the end of November and I deposited $100 during December with no other activity, my balance at the end of December would be $1100. However, adding the two balances would yield $2100, and while it would be terrific if it worked this way, I never had $2100. About the only thing I can do in this situation that would provide value is to average the balances, or perhaps look at minimum and maximum values, or a percentage change. But the fact is still considered semi-additive because it cannot be meaningfully added in all situations (in this example, it cannot be added across statement periods).

Fact table design starts with analyzing all the numeric attributes to determine which ones are usable as facts. For each fact, you must determine the dimensions that describe it. Star schema designers use the term *grain* for the level of detail represented in a fact. Usually, the dimensions tell you the grain. In the employee timecard example, the grain of Daily Hours is Employee by Project by Project Task by Reporting Week by Day of Week.

Generally speaking, facts with the same grain are placed in the same fact table. For example, if we added Gross Pay along with Daily Hours, it would go in the Employee Timecard table. There is an important exception, however. Consider these user views that show orders and shipments by Product, Date and Customer:

Orders

Product	Date	Customer	Ordered Units
A	12/05/15	X	1
B	12/04/15	Y	2
B	12/05/15	Y	1

Shipments

Product	Date	Customer	Shipped Units
A	12/05/15	X	1
B	12/05/15	Y	1
B	12/06/15	Y	1

On initial examination, it is easy to see that there are two facts, Ordered Units and Shipped Units, and the facts have the same grain (Product by Date by Customer). However, the facts are not obtained at the same point in time because order creation and order shipment do not occur at the exact same time. Also, an order could have more than one shipment. If we tried to combine the two facts into one fact table, we would imply that shipments that occurred on a given day were for the order that was placed that same day. However, in reality, we cannot match the shipments to orders because we don't have a unique identifier for the order itself (i.e. there is no Order dimension), and without it, we are making a false assumption in combining the data. The real danger in terms of business analysis is that the data looks fine even though it is probably incorrect!

Order Shipments Fact Table

Product	Date	Customer	Ordered Units	Shipped Units
A	12/05/15	X	1	1
B	12/04/15	Y	2	
B	12/05/15	Y	1	1
B	12/06/15	Y		1

In summary, **always** remember that facts are combined into a common fact table **only** when they have the same grain **and** occur at the same point in time (that is, as part of the same business event or transaction).

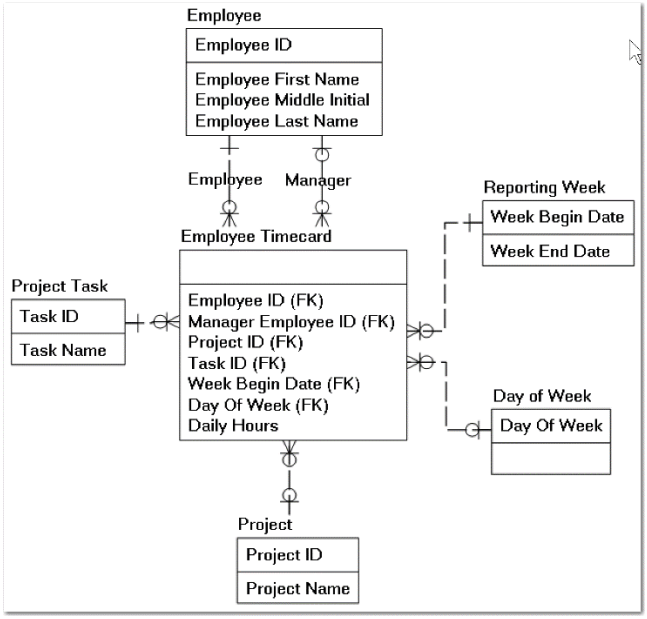
Dimension Table Design

Recall that attributes describe or characterize facts, and that facts can be summed (rolled up), filtered, and ordered by attributes during analysis. In SQL, it is the attributes that appear in WHERE, GROUP BY and ORDER BY clauses that you write while analyzing the data.

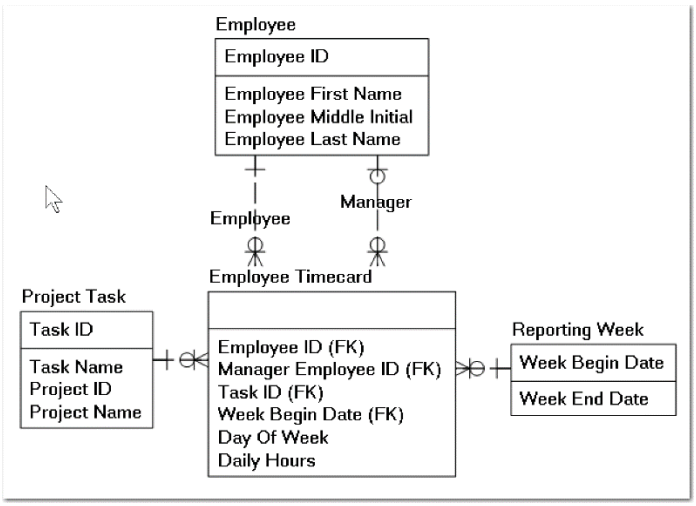
The identification of dimension attributes is straightforward – once you have identified the facts, everything left over is an attribute. Organize attributes in tables based on their unique identifiers.

Dimension Hierarchies: Splitting Versus Collapsing

When dimensions have relationships with other dimensions, we have the option of either *splitting* the hierarchy, with each dimension having an independent one-to-many relationship with the appropriate fact table(s), or *collapsing* the hierarchy into the lowest dimension. Here again is the Employee Timecard fact with its dimensions. The Project table was a parent of the Task table in the original 3NF schema, but it has been split to an independent dimension in this star schema:



Splitting hierarchies is recommended when the parent table in the hierarchy (the Project dimension in this example) is likely to be used in the grain of a fact table that does not include the child table (the Project Task dimension in this case). In this example, it is likely we would eventually have a Project Expenditure fact table with a grain that includes Project, but not Project Task. However, for the purpose of illustrating the alternative, here is the Employee Timecard star schema with Project collapsed into Project Task:



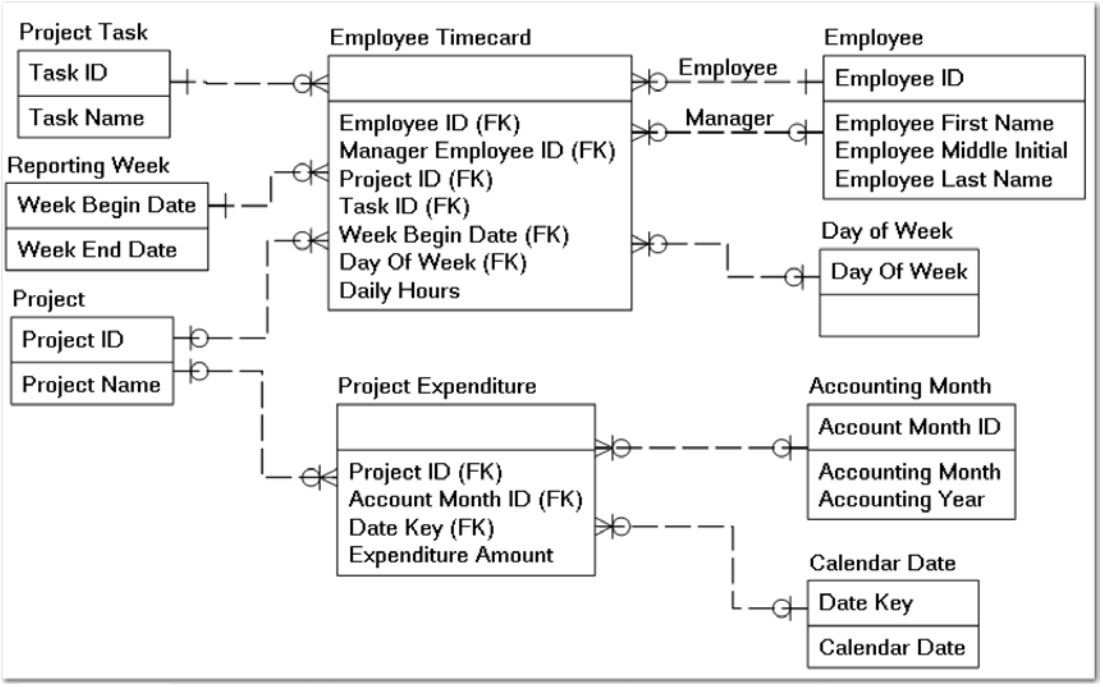
Degenerating Dimensions

You may have noticed one other change is the revised star schema we just looked at – the Day of Week dimension has been removed and the Day of Week attribute placed directly in the Employee Timecard fact table. This is known as *degenerating* the attribute. For dimensions containing just one or perhaps two attributes that are not expected to be used as part of the grain of other fact tables, this is a commonly used simplification. Data analysis is a bit simpler because there is one less dimension to join with, and we have one less dimension table to load with data.

Additional information: [Another Look at Degenerate Dimensions (Kimball Group)]

Conformed Dimensions

In most analytical applications, there are multiple fact tables, each having a distinct set of dimensions. When analytical queries need to use facts from multiple fact tables, at least one common dimension must be used in order to match (join) the facts in one fact table with facts in another fact table. Here is a preliminary star schema design which adds a Project Expenditure fact table and its dimensions to the Employee Timecard example we have been using in this module.



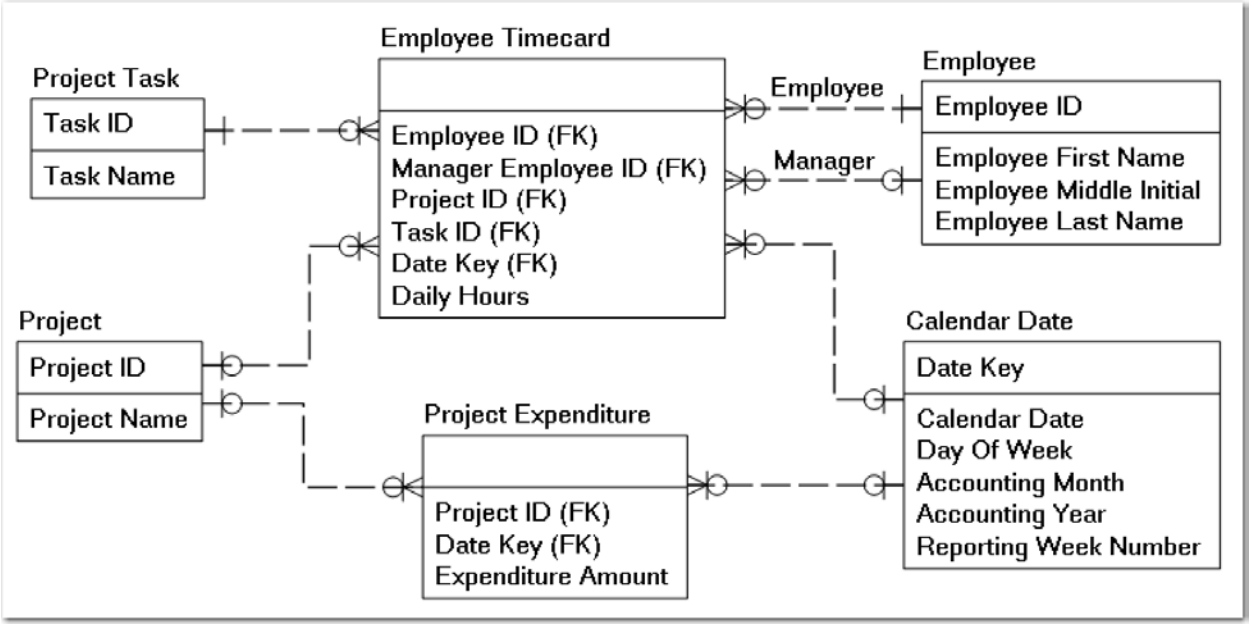
It is reasonable to expect a requirement to include labor costs along with project expenditures in order to determine the total cost of the project. However, this design has a huge flaw in that the only common dimension between Project Expenditure and Employee Timesheet is Project. That means an analytical query that needs to use the Daily Hours fact along with the Expenditure Amount fact has no choice but to summarize the Project Expenditure and Employee fact tables by Project ID (to establish a common grain) before joining the rows of data. Joining fact tables with different levels of detail (different grains) leads to partial Cartesian products, and thus to double-counting the facts. It is reasonable to assume a requirement where business users would want to break down costs by month or by week. However, with this design, they cannot.

Notice that the Reporting Week, Day of Week, Accounting Month and Calendar Date dimensions are similar in that they all track time in increments that can be broken down to numbers of days. The grain of Employee Timesheet is by day, but we have no dimension that clearly shows that – what we have is a begin date for the week and the day of the week. What we need is a conformed dimension for calendar days. Ralph Kimball developed the concept of conformed dimensions:

> **Conformed Dimensions**: two dimensions are conformed when either of these conditions is met:
>
> - The dimensions are exactly the same, including attribute definitions, primary keys, and all contents (essentially the same dimension is shared by multiple facts, or if separate dimension tables are used, one is an exact duplicate of the other).
> - One of the dimensions is a perfect subset of the other, meaning one dimension is a roll-up of the other.

Additional Information: Conformed Dimensions (Kimball Group)

In this example, assigning an account month, day of week, and week begin date to each row in the Calendar Date dimension would be simple to do as part of the process that loads the data into the dimension. We can then use it as a conformed dimension, shared by the two fact tables. A side benefit is that we can remove the other date-related dimensions, significantly simplifying the star schema. Here is the revised schema:



We now have at least two conformed (shared) dimensions between the fact tables, Project and Calendar Date, with considerably more flexibility in summarizing by calendar components (day, week, month, year) than we had with the previous design.

It takes considerable experience to recognize opportunities for conformed dimensions, and while normalization doesn't directly apply here, experience with normalization helps database designers with insights into the real structure and meaning of the data.

Additional Dimension Optimization

There are additional optimization techniques for dimensions that are more advanced, but the ones included here are the primary ones you will see in most star schema designs. If you have an interest in exploring these advanced optimizations, here are some articles to read:

[De-clutter Star Schemas With Junk Dimensions]

[Slowly Changing Dimensions (Part 1)]

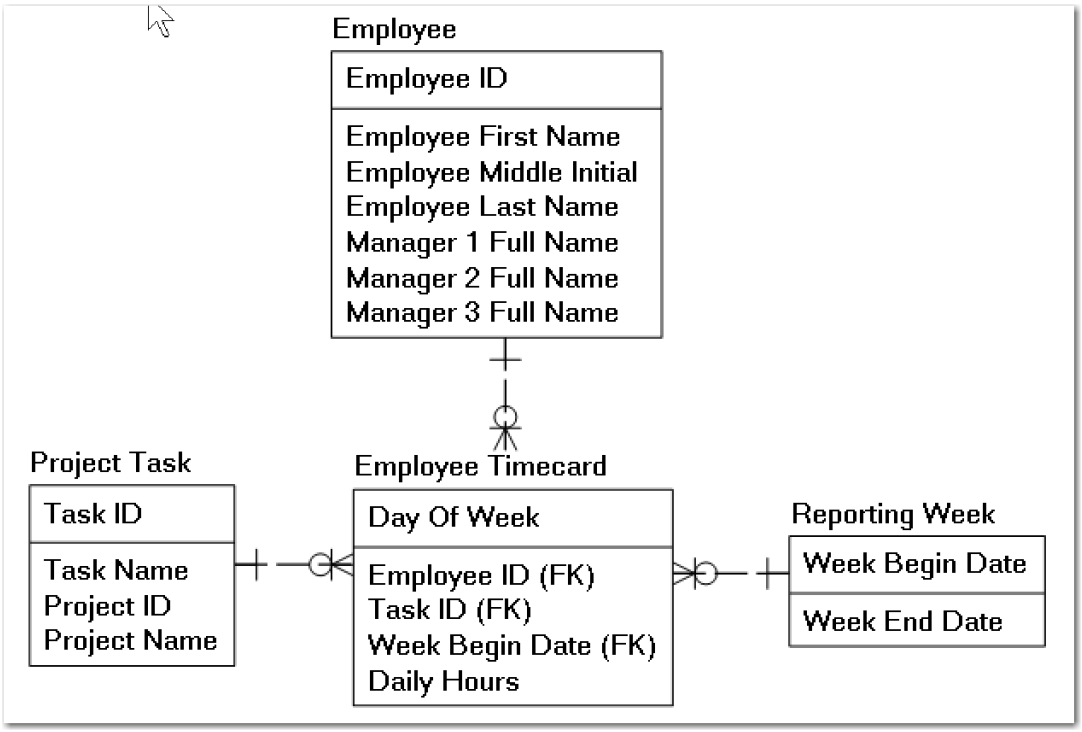
[Slowly Changing Dimensions (Part 2)]

[Creating and Managing Mini Dimensions]

Handling Many-to-Many Relationships

As with normalization, things get more complicated when many-to-many relationships come into play. Let's assume that another organization likes the Employee Timecard star schema and wants to use it for their own analytical purposes. However, there is a wrinkle – the new organization uses a matrix management system wherein employees have multiple managers. That makes the relationships between Manager and Employee many-to-many, which is not an issue in itself, but it makes the relationship between Employee and Employee Timecard many-to-many because we now have multiple managers for each employee.

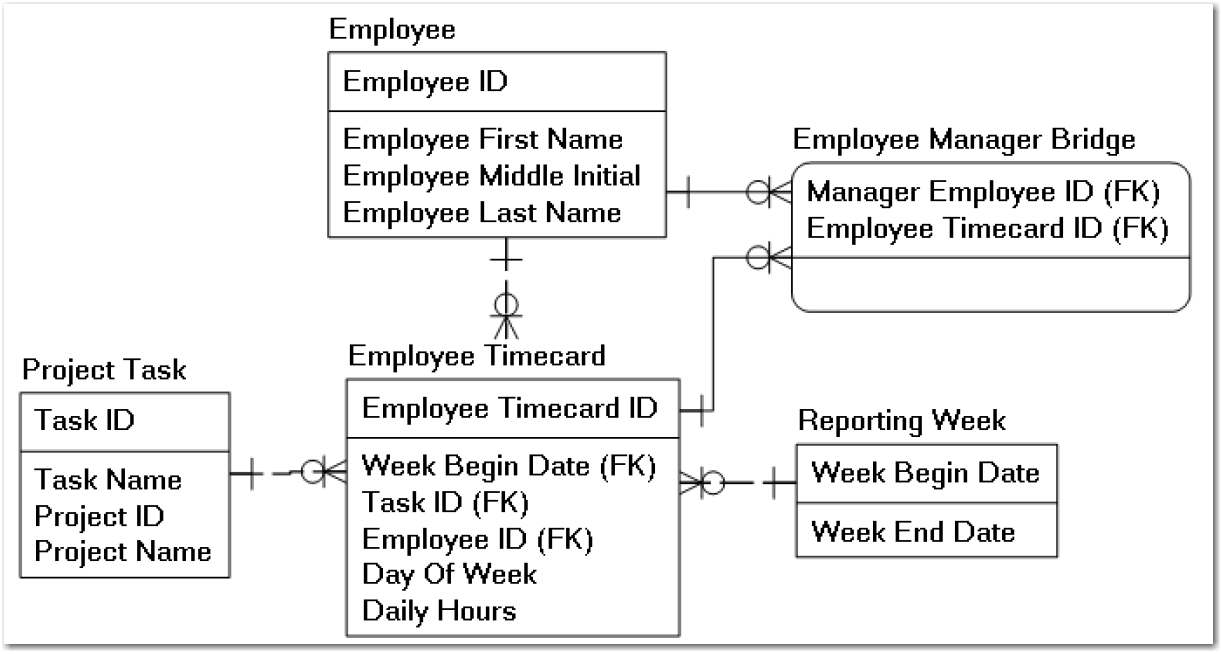
One way to handle a many-to-many situation like this one is using *bridge attributes*, which are essentially a repeating list of attributes. In this case, one attribute for each manager. Here is the Employee Timecard star schema with the bridge attributes added. In this case, I simplified the manger name attributes into the full name (first, middle, and last name combined into a single attribute).



The main advantage of the bridge attributes approach is that you don't have to add additional dimension tables and relationships. However, there are two significant disadvantages:

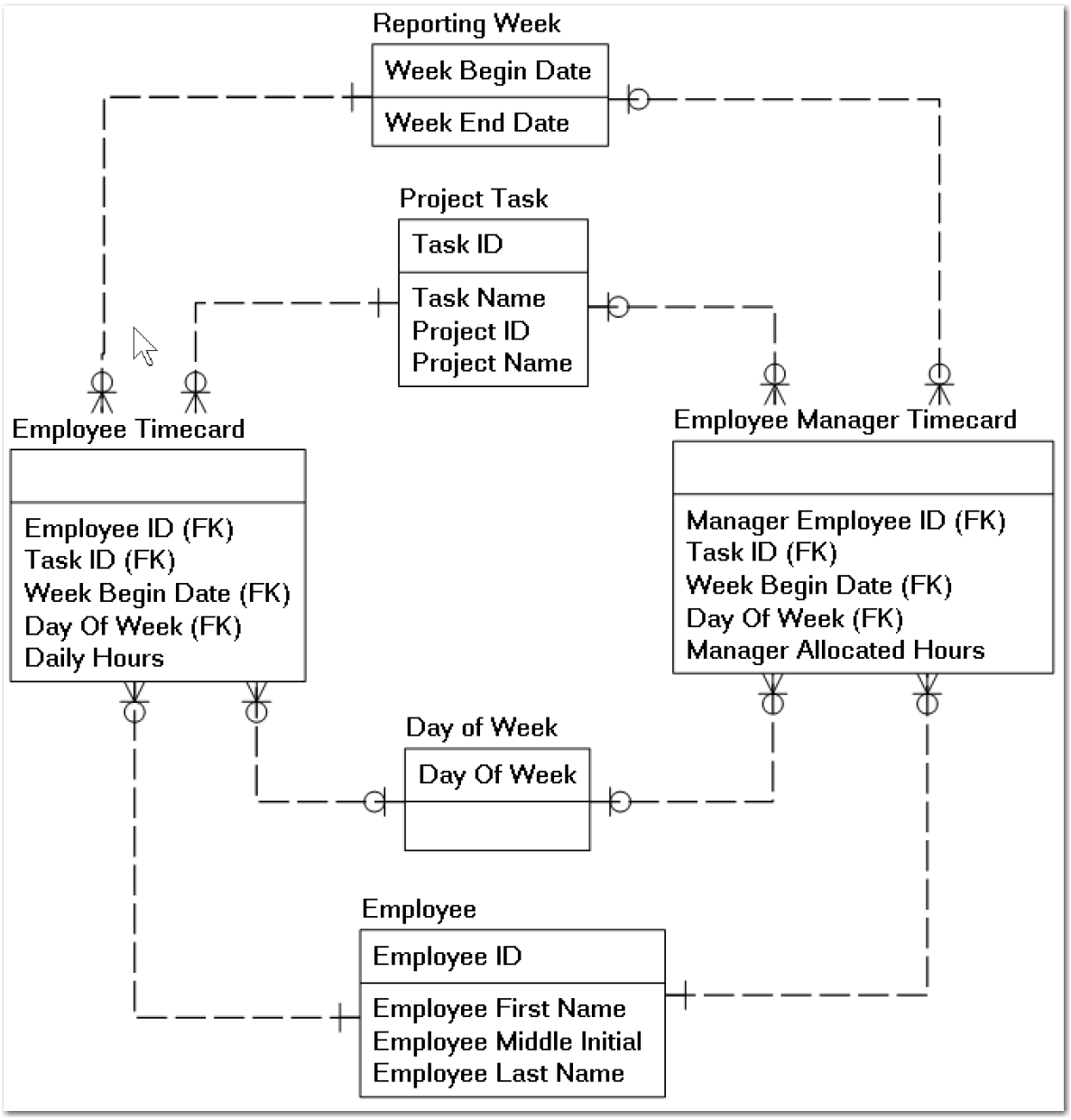
- We must settle on a finite number of repetitions. In this case, we chose a maximum of three managers. For managers with more than three managers, we must choose the three that will be represented in the Employee dimension.
- If we need to filter on a particular manager (for example, we only want to see timesheets where Sally M. Chan is one of the employee's managers), we have to remember to write a predicate that tests all three of the manager attributes (for example, WHERE MANAGER_1_FULL_NAME = 'Sally M. Chan' OR MANAGER_2_FULL_NAME = 'Sally M. Chan' OR MANAGER_3_FULL_NAME = 'Sally M. Chan'). Any simple oversight when writing analytical queries would lead to incorrect query results.

Another possible solution is to use what is known as a *bridge table*, which is a dimension that resolves a many-to-many relationship with another dimension. And, yes, technically it departs from the star schema rule against dimensions have relationships with other dimensions. However, in this case, you have to make compromises in order to have a robust solution. Also, some analysis tools do not handle bridge tables well. Here is the modified schema with Employee Manager Bridge table added:



You may have noticed that I added Employee Timecard ID as a surrogate key for the Employee Timecard fact table to simplify the relationship between Employee Timecard and the Employee Manager Bridge table.

Another possible solution is to create a fact table at the grain of Employee and Manager. This has the potential for double-counting of the Daily Hours fact if the new fact table is summed across managers for the same employee unless we have a method of allocating the daily hours by manager, dividing the hours for each Employee Timecard row across the applicable managers. In this solution, I have created the Allocated Daily Hours fact with that concept in mind. However, this solution only works when it is possible to allocate the facts from the original fact table to mitigate any possibility of double-counting.



Advantages of the Star Schema

Here are some of the primary advantages to using star schema database designs for analytical databases.

- Star schemas are easily understood by business users.
- Star schemas are very commonly used as a user interface for analytical applications. In fact, if you have ever used pivot tables in Microsoft Excel, you have a used an implementation of the star schema.

- Queries are easier to write compared with 3NF schemas because all the dimensions are just one layer (one join) away from the fact tables.
- Compared with 3NF schemas, star schemas are easier to change and expand as business requirements change.

Disadvantages of the Star Schema

Star schemas have the following disadvantages:

- Data integrity is not enforced. The source systems that provide the data for the analytical database must be responsible for data integrity.
- Star schemas do not handle many-to-many relationships as elegantly as 3NF schemas.