Aalborg Universitet

MIRABEL DW

Managing Complex Energy Data in a Smart Grid

Siksnys, Laurynas; Thomsen, Christian; Pedersen, Torben Bach

MIRABEL DW: Managing Complex Energy Data in a Smart Grid

Laurynas Siksnys, Christian Thomsen, Torben Bach Pedersen

Department of Computer Science, Aalborg University
{siksnys, chr, tbp}@cs.aau.dk

Abstract. In the MIRABEL project, a data management system for a *smart grid* is developed to enable smarter scheduling of energy consumption such that, e.g., charging of car batteries is done during night when there is an overcapacity of *green energy* from windmills etc. Energy can then be requested by means of *flex-offers* which define flexibility with respect to time, amount, and/or price. In this paper, we describe MIRABEL DW, a data warehouse (DW) for the management of the large amounts of complex energy data in MIRABEL. We present a unified schema that can manage data both at the level of the entire electricity network and at the level of individual nodes, such as a single consumer node. The schema has a number of complexities compared to typical DW schemas. These include *facts about facts* and *composed non-atomic facts* and unified handling of different kinds of flex-offers and time series. We also discuss alternative data modeling strategies and present typical queries from the energy domain and a performance study.

1 Introduction

More and more *green energy* is being produced by renewable energy sources (RES) such as windmills. It is, however, not possible to store larger amounts of energy and use it later. Therefore, there often is an unused capacity, e.g., during nights when most consumers sleep, but not enough green energy during day hours when most consumers are active. The EU FP7 project MIRABEL (Micro-Request-Based Aggregation, Forecasting, Scheduling of Energy Demand. Supply and Distribution) [11] addresses this challenge by proposing a "data-driven" solution for balancing supply and demand utilizing their flexibilities. Flexible demand such as for dishwashers and charging an electric vehicle can often be shifted to a time when green energy is available. Non-flexible demand such as lights, TV, or cooking stoves must still be satisfied at demand-time. In the MIRABEL-settings, a consumer offers a so-called *flex-offer* [2, 14] for every intent of flexible energy demand. The flex-offer must describe when and how much energy is needed and how flexible the demand is in time and amount. Likewise, a producer can offer a flex-offer for every intent of energy supply. The different flex-offers can then be accepted (or rejected if they cannot be fulfilled) and scheduled for execution at a given time. There will be extremely large quantities of such flex-offers and they cannot be scheduled individually. Instead flex-offers are *aggregated* into larger flex-offers which become scheduled and then *disaggregated* into the smaller flex-offers again [14]. To

enable this, there will be smart *nodes* at both consumer sites and producer sites in the electricity grid which we denote a *smart grid*.

There is a a strong need for efficient data management in these nodes. In this paper, we present *MIRABEL DW* which is a data warehouse (DW) for the management of large amounts of complex energy data in the MIRABEL project. This paper is the first to present a DW schema for the important domain of energy data. The schema can represent different "actors" in different "roles" as defined by the "Harmonised Electricity Market Role Model" [4] as well as (individual and aggregated) flex-offers, and time series. In the future, the managed data is to be distributed over millions of nodes [2] in non-traditional ways. In the paper, we focus on a DW on a single node, but present a unified schema that can manage data both at the level of the entire electricity network and at the level of individual nodes, such as a single consumer node. Compared to typical DW schemas, the schema has a number of complexities which we discuss in the paper. These include *facts about facts* and *composed non-atomic facts* and unified handling of different kinds of flex-offers and time series. We also discuss alternative data modeling strategies that use denormalization and arrays, respectively. Further, we present typical queries from the energy domain and a performance study that compares the described schemas with the denormalized and array-based alternatives.

The rest of the paper is organized as follows: Our representations of flex-offers, time series and actors are presented in Sections 2, 3, and 4, respectively. These parts together form the full schema which is presented in Section 5. Examples of analytical queries on the schema are given in Section 6. A performance study is given in Section 7. Previous work related to this is presented in Section 8 before the concluding remarks and pointers to future work which are given in Section 9.

2   Modeling of Flex-Offers

In this and the following two sections, we first present the data model we use in MIRABEL DW. Then we discuss the non-standard and advanced techniques that are applied in the modeling.

2.1   Data Model

To represent MIRABEL's flex-offers (both aggregated and non-aggregated) is an essential task for MIRABEL DW. This is done by means of the tables shown in Fig. 1. We first describe the dimensions (which are recognized by the prefix D_ in their table names) and then the fact tables (recognized by the prefix F_ in their names). All dimension tables have surrogate keys with names ending with Id. The possible states for a flex-offer (such as "offered", "accepted", and "rejected") are represented in the dimension D_flexEnergyState. A flex-offer has its state for a certain reason (for example, a flex-offer becomes rejected if the offered price is too high). The possible reasons are represented in the dimension D_flexEnergyStateReason. As we expect few generic reason categories (e.g, "Price too high") and many more specific reason descriptions (e.g. "Price (499.50 euros) too high") to exist, we have columns for both the generic categories and the specific reasons such that a hierarchy exists. In MIRABEL DW, we represent time by discretized time intervals. This is done by D_timeInterval which represents
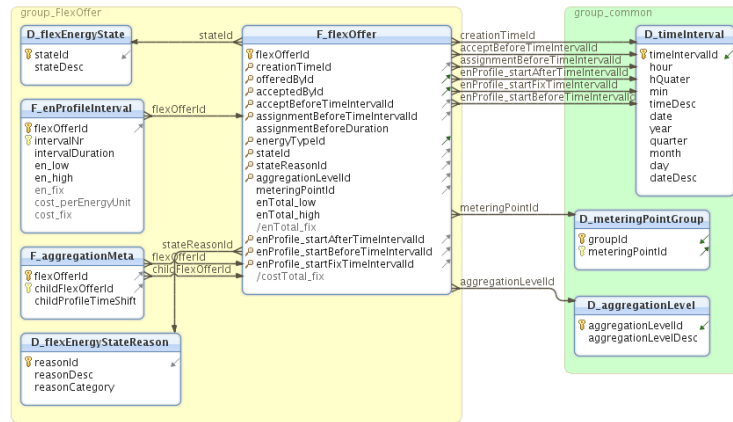
Fig. 1. Tables for representing flex-offers

15 minutes intervals (for now; other interval lengths can be chosen if needed). Flex-offers are always related to at least one metering point (at the location where the energy is to be consumed or produced), but if a flex-offer is aggregated, it will be associated with many metering points. To capture this, D_meteringPointGroup is used as bridge table [7] between the fact table and D_meteringPoint which represents the individual metering points. To represent the aggregation level of a flex-offer, D_aggregationLevel is used.

The fact table F_flexOffer holds flex-offer facts. It references all the previously described dimension tables. There are six foreign keys to D_timeInterval to represent different times such as when the flex-offer was created and when it at the latest has to be assigned etc. These foreign keys thus all represent an absolute time. There is also an attribute assignmentBeforeDuration which holds a time span telling how long before the actual execution time the assignment must take place.

Further, F_flexOffer references D_legalEntityRole (explained later) twice to represent who offered and accepted the flex-offer, respectively. Only the current information about a flex-offer is held; if a flex-offer is modified, the old fact is overwritten. There are measures to hold the lowest and highest amount of energy required by the flex-offer as well as a measure to hold the "fixed" amount of energy that becomes accepted. Further, a measure holds the total cost of the fix. Finally, each represented flex-offer is given a unique identifier in the attribute flexOfferId which technically is a degenerate dimension.

Information about the profile intervals of flex-offers is represented in the fact table F_enProfileInterval. This fact table only has a single foreign key which references the unique flexOfferId in F_flexOffer. The imported value together with a sequential intervalNr forms the primary key for F_enProfileInterval. The reason for this design is that a single flex-offer can have many profile intervals. For each represented profile interval, there is a duration specifying how many time units the profile interval spans over, and both the lowest and highest amount of energy needed in this interval. When the flex-offer becomes fixed, the actual amount of energy in the interval and the price for this

energy also becomes represented. An alternative to this design would be to represent the measures of F_enProfileInterval in *arrays* in F_flexOffer such that all data about a given flex-offer would be represented in a single fact. Yet another alternative would be to represent all attributes of F_enProfileInterval in F_flexOffer, i.e., denormalize the data and have one (wide) fact in F_flexOffer for each profile interval. (For space reasons, we do not show the alternative schemas in figures.)

As flex-offers can be aggregated into larger flex-offers, we also introduce the table F_aggregationMeta which references F_flexOffer twice to point to the aggregating "parent flex-offer" and the smaller "child flex-offer" which has been aggregated, respectively. Profiles of each child flex-offer can be shifted relatively to the profile start of the parent flex-offer when aggregating child flex-offers into the parent. Therefore, for every child flex-offer, the childProfileTimeShift attribute indicates the amount of time units the profiles of the child flex-offer has been shifted in the aggregated flex-offer. This information is used in the disaggregation.

2.2 Modeling Challenges

The fact table F_flexOffer is the central fact table for representation of flex-offers. It is, however, also used as a dimension table in the sense that each fact has a unique ID such that F_enProfileInterval and F_aggregationMeta can reference F_flexOffer and in effect store *facts about facts*. Considering F_flexOffer and F_enProfileInterval, it can even be discussed *what* a fact is. An energy profile interval (in this context) always belongs to a flex-offer and any meaningful flex-offer has an energy profile interval (a flex-offer for zero consumption/production at an undefined point in time is hardly interesting). It could be argued that a single fact is represented by a single row in F_flexOffer and many rows in F_enProfileInterval. Unlike traditional DW schemas, we thus have non-atomic *composed* facts. As pointed out above, we could alternatively have modeled this by using arrays in F_flexOffer to hold the measures that currently are represented in F_enProfileInterval. This would, however, make it more cumbersome to compare different measures (e.g., en_low with the minimum energy requirement to en_fix with the assigned energy) as the interval position currently represented by intervalNr only would be implicitly represented by the position in the array. The denormalized variant (with a fact in F_flexOffer for each profile interval) would increase redundancy dramatically.

Another interesting aspect of MIRABEL DW is how it represents facts for both non-aggregated and aggregated flex-offers in a unified way. The aggregation is unlike traditional aggregation since the parent flex-offer contains other flex-offers that can be shifted within the parent flex-offer. We call the contained flex-offers *shiftable child facts*.

3 Modeling of Time Series

3.1 Data Model

In MIRABEL DW, time series are represented by means of the tables shown in Fig. 2. It is necessary to be able to represent time series of various types, for now energy,
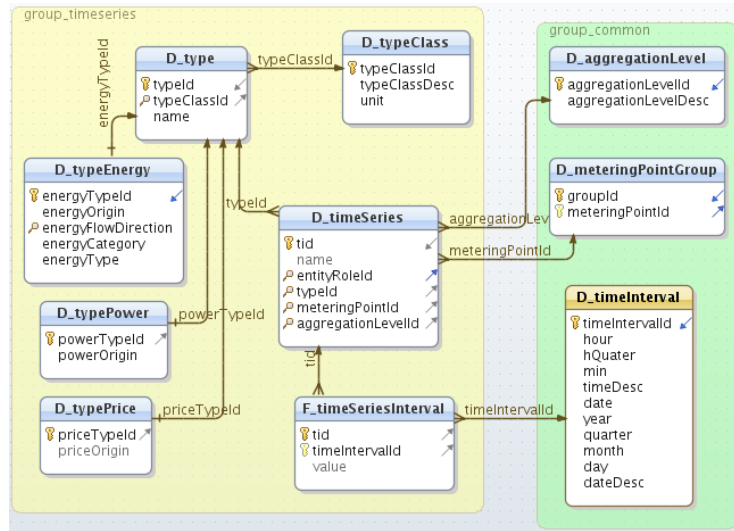
Fig. 2. Tables for representing time series

power, and price. To represent these general classes, we use the D_typeClass dimension table. Apart from its surrogate key, it has the attribute typeClassDesc which holds a textual description of the time series type (such as "Energy") and the attribute unit which holds the unit of measurements (such as "kWh"). Instances of the general types are represented in the table D_type. For example, an instance of the "Energy" class is "Energy-Metered-Production-RES-Wind". D_type references D_typeClass to represent the hierarchy between types and type classes. For different types of time series, it is, however, necessary to store different information. Therefore, we introduce the tables D_typeEnergy, D_typePower, and D_typePrice to hold the attributes that are relevant for the different types. These tables supplement, but cannot replace, D_type. The reason is that we need a single table to reference from D_timeSeries to represent the type of the time series in question. Thus D_type is referenced from D_timeSeries, but the special attributes for an energy time series are represented in D_typeEnergy. The latter table has columns to describe the origin of the time series (e.g. "Metered" or "Forecasted"), the flow direction (i.e., if it is production or consumption), the category (e.g., energy from renewable energy sources), and the type of energy (e.g. "Wind"). The design is likely to evolve in the future. For example, there is a traditional hierarchy where types roll up into categories that roll up into flow directions. A more advanced hierarchy is, however, needed to represent hybrid energy types like "At least 90% energy from renewable energy sources and the rest produced from coal".

D_timeSeries holds a single entry for an entire time series. For each represented time series, there is a unique ID tid and a name may be given. Further, D_timeSeries references D_type (as previously described), D_aggregationLevel to represent the level of aggregation of the time series, and D_meteringPointGroup to represent which meters the time series describes. Thus, D_timeSeries is mainly used to relate different di-

mension values that describe the represented time series. The values of the time series are, however, represented in the fact table F_timeSeriesInterval. This table references D_timeSeries to identify the time series a value belongs to and D_timeInterval to identify the time instant when the value occured. Finally, the table holds the value itself as the measure. A fact thus exists for each value in each time series. It can, however, also be argued that a fact consists of what it represented in F_timeSeriesInterval *and* what is represented in D_timeSeries which – apart from a possible name – only points out to other dimensions.

3.2 Modeling Challenges

Similarly to the represention of flex-offers, our representation of time series also leads to compound facts where one fact can be considered to be made up of parts in different tables (D_timeSeries and F_timeSeriesInterval). Actually, an alternative design is to merge F_timeSeriesInterval into D_timeSeries such that the values instead are represented in an array, meaning that a single time interval (and all its values) only would result in one fact. Yet another alternative is to merge D_timeSeries and F_timeSeriesInterval and have a row for each value in a time series. There are thus different possible ways to represent the complex sequence-facts arising from time series. We choose the model in Fig. 2 since it both reduces complexity (compared to the first alternative where two arrays must be processed to find the value for a given time instant) and redundancy (compared to the second alternative where there is very wide fact for each value in the time series).

In our modeling of time series, the schema is neither a traditional star schema nor a snowflake schema. One reason for this is of course the compound facts discussed above. Another reason is the support for different types of time series for which different attributes are needed. We have different tables that reference D_type which also is the dimension table referenced from the fact table. Consider for example D_typeEnergy which represents attributes that are relevant for energy time series. An alternative design would be to join all these D_type* tables into one dimension table, but for every dimension member many attribute values would then be NULL.

4 Modeling of Different Actors and Market Areas

4.1 Data Model

Many different entities are involved in different roles in energy trading and network operation. We represent the needed actors from the "Harmonised Electricity Market Role Model" [4] by means of the tables in Fig. 3

The table D_role represents roles such as "Producer" and "Consumer". A role can belong to another parent role and this is captured by a self-reference. For example, the parent role of both "Producer" and "Consumer" is "Party Connected To Grid". Legal entities are represented by D_legalEntity. To capture when a certain legal entity plays a certain role (a single legal entity can play several roles), we use D_legalEntityRole. This table references both D_role and D_legalEntity. Further, it has an attribute to hold a unique ID for a given legal entity playing a given role. We include this ID as it makes
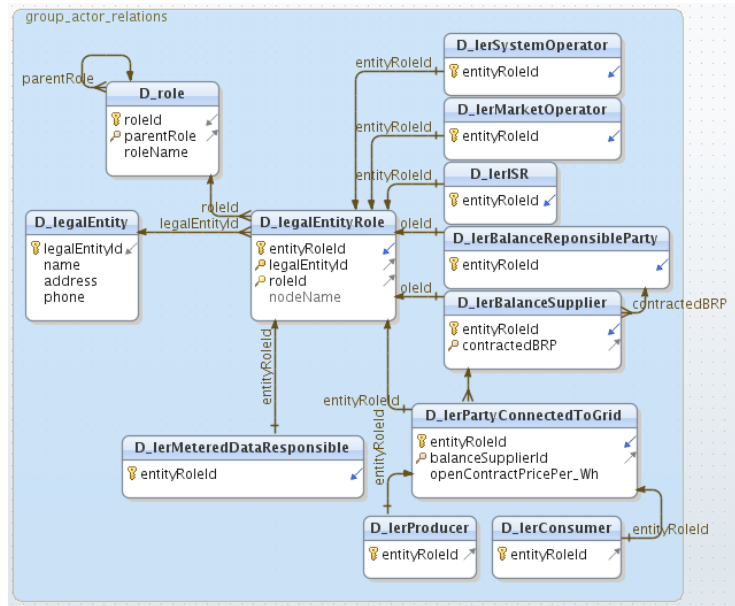
Fig. 3. Tables for representing different actors/roles

it easy to point to a legal entity in a certain role. We do exactly that from a number of tables as shown in Fig. 3. For each role, there is a specialized table that (directly or indirectly through another table) references D_legalEntityRole. Some of them, like D_lerSystemOperator, are simple and do only have one attribute which is a reference to this ID. The specialized table can be referenced and it is then explicit what kind of role is referenced. For example, the table D_lerSystemOperator is referenced from D_marketBalanceArea as shown in Fig. 5. A slightly more complex example is D_lerPartyConnectedToGrid which references D_legalEntityRole and also D_lerBalanceSupplier to represent that a party connected to the grid always is so through a balance supplier. Further, D_lerPartyConnectedToGrid is itself referenced from its specializations, D_lerProducer and D_lerConsumer.
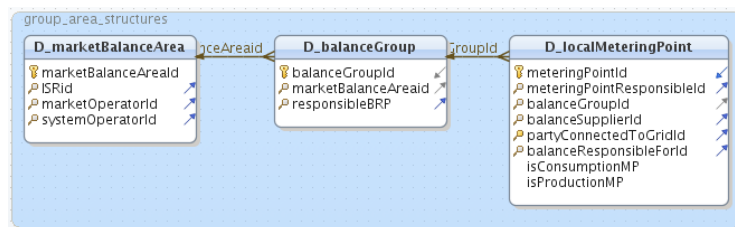

Fig. 4. Tables for representing market areas

Finally, we have tables to represent market areas as shown in Fig. 4. D_localMetering-Point represents the meters that are connected to the grid. Such meters are installed both at the producer and consumer sites. D_localMeteringPoint references four different specializations of D_legalEntityRole. Further, it references D_balanceGroup which in turn references D_marketBalanceArea which hierarchically groups metering points.

4.2 Modeling Challenges

To the best of our knowledge, this is the first paper to describe a DW for the complex concepts of actors and roles in the "Harmonised Electricity Market Role Model" [4]. Our model captures both how legal entities can play different roles and how roles can be parts of other roles. This is captured by the tables D_legalEntity, D_role, and D_legalEntityRole. In addition to these tables, a (narrow) table has been added for each role a legal entity can play (see the D_ler* tables). It is then possible to represent attributes that are only relevant for certain roles such as done for D_lerBalanceSupplier. Further, when foreign keys reference these tables (instead of just referencing D_legalEntityRole), it is explicit what kind of role playing is referenced and it helps to avoid mistakes where, e.g., a balance supplier is referenced where a balance responsible party actually should have been referenced. We note that if no special attributes must be stored for the different roles, then instead of storing the D_ler*'s as physical tables, they can be views selecting from D_legalEntityRole. This reduces the risk of mistakes further and makes maintenance of them automatic.

5 The Full Schema

To summarize the previous descriptions, the full schema for MIRABEL DW is shown in Fig. 5. The schema can capture the (needed) roles from the Harmonised Model [4] as well as the "actor configurations" where different actors play different roles. The schema also includes specializations of legal entities. Further, the schema can capture different kinds of time series as complex sequence facts. The schema is thus general enough to hold all the data that is needed in the MIRABEL project. It should, however, be noted that no single node is intended to hold all data. Instead, a node should only hold data that is relevant for the site where it is installed. For an end-consumer this would typically be her own non-aggregated flex-offers and time series about metered energy. For a balance responsible party buying electricity on the market and selling it to end-consumers, it would include both aggregated and non-aggregated flex-offers, forecasted and metered time series, and market areas. The data will thus be distributed accordingly to the roles played by the owners of the nodes. The data will also be at different aggregation levels such that some nodes have detailed data while others have more aggregated data. For example, will a consumer know the details of her flex-offers, i.e., when she has requested energy and how much. For a balance responsible party, the individual non-aggregated flex-offers and end-users generating may not be known, but the aggregated information will be known, e.g., that x MWhs must be produced in a given time interval. Note that the different nodes can use the same schema.
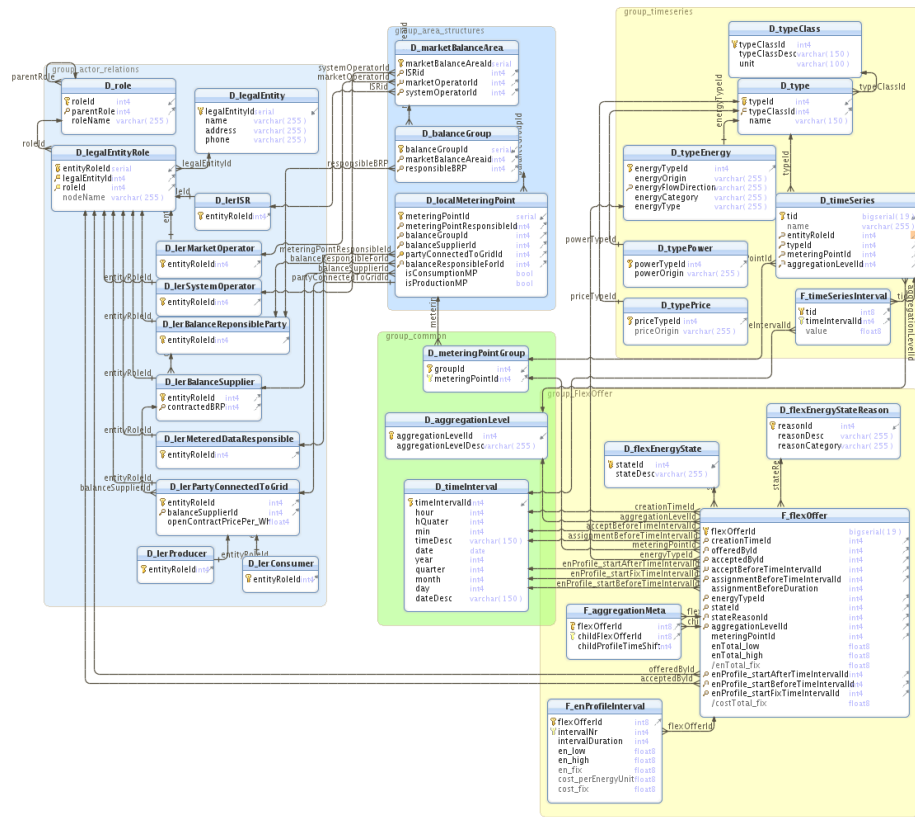
Fig. 5. The full schema for MIRABEL DW

6 Queries

In this section, we give examples of interesting queries on data in MIRABEL DW. We first focus on queries on flex-offers and then on time series.

6.1 Queries on Flex-Offers

The first example, Q1, considers the flexibility in flex-offers, both with respect to time and amount of energy.

```
Q1:   SELECT AVG((enProfile_startBeforeTimeIntervalId –
                 enProfile_startAfterTimeIntervalId) *
                (SELECT SUM((en_high – en_low) * intervalDuration)
                 FROM F_enProfileInterval i
                 WHERE i.flexOfferId = f.flexOfferId)
                )
      FROM F_flexOffer f;
```

The query uses the flexibility with respect to time, i.e., the difference between when the flex-offer at the latest *has* to be executed and when it at the earliest *can* be scheduled.

We assume that time interval IDs are assigned sequentially and thus use the difference between the IDs of the time intervals to find the flexibility. This flexibility is multiplied with the SUM of the energy flexibility in each profile interval. The energy flexibility in a profile interval is found as the length of the profile interval multiplied with the difference between the maximally required amount of energy and the minimally required amount of energy. Finally, the shown query considers the average of the combined flexibility for all flex-offers. The query is an example of a non-traditional kind of aggregation. If we consider a graph showing the relative start and end times for profile intervals on the X axis and the minimal and maximal energy amounts on the Y axis, the query Q1 finds the *area of energy flexibility* for all flex-offers and multiplies these with the length of their time flexibilities before the entire average is found. This number is primarily of interest *before* the scheduling gets done and a high number indicates much freedom in the scheduling while a low number shows that the considered flex-offers are not very flexible.

The next example, Q2, is of interest *after* the scheduling and gives the total amount of scheduled energy. This is a simple query which, however, must read data from many rows in a realistic setting (the DBMS we use does currently not support materialized views).

```
Q2: SELECT SUM(en_fix)
    FROM F_enProfileInterval;
```

Q3 is a more complex query to apply after scheduling has taken place. It builds a time series that for each time interval ID shows the amount of fixed energy.

```
Q3: SELECT timeIntervalId, SUM(en_fix_part)
    FROM (SELECT en_fix_part, ROW_NUMBER() OVER (PARTITION BY i.flexOfferId
                ORDER BY intervalNr) - 1 + f.enProfile_startFixTimeIntervalId
                AS timeIntervalId
          FROM (SELECT flexOfferId, intervalNr, en_fix / intervalDuration
                    AS en_fix_part, generate_series(1, intervalDuration)
                FROM F_enProfileInterval
                WHERE en_fix IS NOT NULL
              ) i, F_flexOffer f, D_flexEnergyState s
          WHERE i.flexOfferId = f.flexOfferId AND f.stateId = s.stateId
                AND s.stateDesc = 'Assigned'
         ) AS subquery
    GROUP BY timeIntervalId
    ORDER BY timeIntervalId;
```

The query computes the IDs of the time intervals where a flex-offer's profile intervals are executed. But a profile interval has a duration (in intervalDuration) which defines how many time intervals the profile interval spans. Therefore, it is necessary to (evenly) distribute the profile intervals' energy amounts over one or more time intervals. To do this, one "part" row is generated for each time interval a profile interval covers by means of generate_series. This happens in the innermost SELECT. The result of this is used by the second SELECT which also uses the SQL window function ROW_NUMBER to enumerate the rows in each partition where a partition consists of the part rows for a given flex offer and is ordered by the interval numbers. Thus, the resulting row number corresponds to the number of time intervals between the assigned start time for the entire flex offer and the part represented by the row (we subtract 1 since ROW_NUMBER counts from 1). When we add enProfile_startFixTimeInterval for

the flex-offer, we get the ID of tje absolute time interval when the part executes. Finally, the outermost SELECT aggregates the sums of fixed energy amounts over all parts belonging to a given time interval.

6.2 Queries on Time Series

`Q4` is query that finds the balance, i.e., the difference between produced and consumed energy, for a 24 hours period.

```
Q4: SELECT date, timeDesc,
           SUM(CASE energyFlowDirection WHEN 'Production' THEN value
                                        ELSE 0 END) AS production,
           SUM(CASE energyFlowDirection WHEN 'Consumption' THEN value
                                        ELSE 0 END) AS consumption
           SUM(CASE energyFlowDirection WHEN 'Production' THEN value
                                        WHEN 'Consumption' THEN -1 * value
                                        ELSE 0 END) AS balance
    FROM F_timeSeriesInterval f, D_timeSeries ts, D_type ty,
         D_typeEnergy te, D_timeInterval ti
    WHERE f.tid = ts.tid AND ts.typeId = ty.typeId AND te.energyTypeId =
         ty.typeId  AND ti.timeIntervalId = f.timeIntervalId AND
         te.energyOrigin = 'Metered' AND ti.date = '2011-06-01'
    GROUP BY ti.timeIntervalId
    ORDER BY ti.timeIntervalId;
```

The query `Q4` is an example where we use the special attributes that only apply to some time series. In this example, we consider consumed and produced energy and we thus use energyFlowDirection and energyOrigin which only exist for energy time series. The query sums the production values, consumption values, and the difference between them for each time interval that belongs to a given date.

Our last example, `Q5`, is a query to find those time series where the average energy usage grouped on hours exceeds the average energy usage for the hour with 25% or more at least 10 times.

```
Q5: WITH indavguse AS (
        SELECT tid, hour, COUNT(value) AS indcnt, AVG(value) AS indavg
        FROM F_timeSeriesInterval NATURAL JOIN D_timeInterval
        GROUP BY tid, hour
    ),
    totavguse AS (
        SELECT hour, SUM(indcnt * indavg) / SUM(indcnt) AS totavg
        FROM indavguse
        GROUP BY hour
    ),
    overuse AS (
       SELECT tid, t.hour, indavg, totavg,
              COUNT(*) OVER (PARTITION BY tid) AS cnt
       FROM totavguse t, indavguse i
       WHERE t.hour = i.hour AND indavg >= 1.25 * totavg
    )
    SELECT tid, cnt, hour, indavg, totavg
    FROM overuse
    WHERE cnt > 10
    ORDER BY tid, hour;
```

The query has Common Table Expressions (CTEs) in the WITH part. In the first CTE, `indavguse`, we compute a (temporary) table with the average hourly energy usage for each time series. The result is used again to compute the second CTE, `totavguse`, where we find the average energy use per hour among all time series (we

could join F_timeSeriesInterval and D_timeInterval again, but it is faster to reuse the result of the previously computed CTE). In the third CTE, `overuse`, we join the the results of the two previous CTEs to find the IDs of time series and the hours from `indavguse` where the consumption is at least 25% higher than the general hourly average consumption found in `totavguse`. Further, we use COUNT as a window function to count how many such hours we find for a given time series. Finally, we select the ID of the time series, the count of hours with an average energy usage at least 25% higher than the average, and the consumption in the last SELECT clause.

7 Performance Study

In this section, we compare the performance of the queries from the previous section. We consider them as they are on the described schema (called "MDW") and in addition, we consider alternative queries on the described schema alternatives with denormalization and arrays, respectively. In the denormalized variant, F_flexOffer and F_enProfileInterval are joined and so are F_timeSeriesInterval and D_timeSeries (however, with the name varchar attribute replaced by an integer to make it a typical fact table). In the array variant, the same tables are joined, but now grouped on all dimension references and with measures aggregated into arrays. For the tests, we use a real life data set with consumption data from 963 customers (the data originates from the MeRegio project [10]) and we synthetically generate flex-offers based on this data set. This gives rise to 963 (energy consumption) time series with 32.1 million time series values, and 3,1 million flex-offers. We test the performance on a Linux server with two Quad Core 1.86GHz Intel Xeon CPUs, 16 GB RAM, 4 SATA 7200RPM disks (with one dedicated to the DBMS). The DBMS is PostgreSQL 9.1 [12] and has the parameter shared_buffers set to 4GB, temp_buffers to 128MB, and work_mem to 96MB. All tables are "fully vacuumed" such that their disk representations only take up the needed space and do not occupy unused space. Further, the tables are "analyzed" such that their statistics are up-to-date. Each query is executed once in a warm-up round and then the queries are executed in a round-robin fashion such that each query gets executed five times. We report the average execution times. The results are shown in Figure 6.

For `Q1`, it can be seen that the MDW variant is the fastest followed by the array variant (38.3 seconds and 49.1 seconds, respectively). These two query variants have similar plans, but with arrays there are fewer rows to process. On the other hand, these rows need to have their arrays "unnested" to produce as many values as there are rows to consider in the MDW variant. When the denormalized variant is considered, there are also many rows and these rows are wide. Further, the plan is not similar to the plans for the other variants as GROUP BY is necessary with this variant. This makes the denormalized variant the slowest (123.4 seconds).

For `Q2`, the MDW variant is again the fastest (8.9 seconds) to use. Again, the array variant is the second fastest (11.1 seconds). With this variant the arrays must again be unnested to produce the values that are available in the rows in the MDW variant. The denormalized variant uses wider rows and is the slowest (16.8 seconds).

For `Q3`, the MDW variant remains the fastest (172.1 seconds) while the array variant now is the slowest (237.2 seconds) even though it avoids a join. On the other hand, the array variant requires a SELECT clause to unnest the array and an extra use of
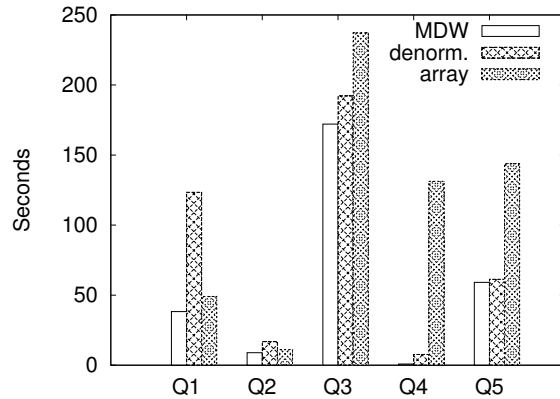
Fig. 6. Results of performance study

ROW_NUMBER to recreate the values from intervalNr which only are implicitly available from the array positions. The denormalized variant (192.2 seconds) is bit slower than the MDW variant even though it avoids a join.

For Q4, the MDW variant is significantly faster (0.8 seconds) than the others. The denormalized variant which avoids a join, uses an order of magnitude more time (7.7 seconds). The array variant is by far the slowest (131.9 seconds) as there is no index on timeIntervalId which is an array. Thus all rows must be processed and have their rows unnested to perform a join with D_timeInterval.

For Q5, the MDW and denormalized variants perform similarly (59.1 and 61.3 seconds, respectively). The queries involve the same number of rows and are identical apart from that the denormalized variant uses a wider table. For the array variant, the first CTE has to unnest two arrays and the query takes longer time (143.8 seconds).

To summarize, the MDW variant performs the best for all queries. Another interesting thing to consider, is the disk space usage. The tables F_flexOffer, F_enProfileInterval, F_timeSeriesInterval, and D_timeSeries take up 4.1 GB in the MDW variant (not counting indexes). Their alternative representations take up 7.0 GB in the denormalized variant and 1.9 GB in the array variant, respectively. It notable how little space the array variant uses compared to the other variants due to its fewer number of rows (and thus fewer space-consuming row headers). Overall, the MDW variant is a good choice considering both its performance and space requirements.

8 Related Work

In the energy sector, there is number of standardized data models used to represent the major objects in an electric utility enterprise [6] as well as to define administrative data internally interchanged between European electricity markets [4, 5]. These models focus on various aspects of energy trading and physical electricity delivery, and specify 1) components of a power system at the electrical level, 2) actors and roles involved in the

energy trading, 3) relationships and data exchange between those entities. These models are used as a basis for the MIRACLE data model [8], which further enriches them with the concept of shiftable consumption and production. All these models, however, focus on a semantic rather than the storage or the management of energy-related entities. By focusing on two most important entities in MIRABEL, i.e., time series and flex-offers, this paper, on the other hand, presents data representation models for these two types of entities offering a convenient storage and a good performance of analytical queries.

This paper is the first to dealt with the storage of flex-offers, but there are previous works which focus on time series and warehousing, e.g. UML-based modeling of time-series in DWs [15], and temporal aggregation of multidimensional data [3], and temporal DWs exploiting research results from the field of temporal databases [9]. Our modeling of different time-series types have similarities with Bauer et al.'s work [1]. They discuss "locally valid dimensional attributes" whose existence depends on values of dimensional elements. This is the case, e.g., for our attribute energyType which only exists if the D_type value represents an energy time-series. The problem of representing all these attributes in a single dimension table (as in a typical star schema) is that there will be many NULLs in the held data. Bauer et al. propose to have separate tables with the specific attributes and then create views "on top" of these with common attributes as well as textual values showing the name of the relation the data comes from which can be used for hierarchical classification. In contrast, we use tables (and not views) for the common attributes of a dimension and then represent special attributes that only exist for some dimensional values in other tables that reference the table with the common attributes. This makes it possible to declare foreign keys to the dimension table with the common attributes and also declare indexes and constraints on these tables. Bauer et al. also propose to use table inheritance to represent such cases. This would also be possible in our DBMS [12], but constraints cannot be enforced on child tables then.

In the current paper, we consider different representations of profile intervals and time series intervals which can be considered as facts with multi-valued measures. The latter case also has a many-many relationship between the time series facts and the time interval dimension. Previous work [13] has considered many-many relationships between fact tables and dimension tables. Our denormalized representation is similar to one of the methods of [13] whereas our other approaches with fact tables referencing other fact tables and measure values in arrays, respectively, are different.

9 Conclusion

In this paper, we have presented a DW schema for managing the complex energy data in a smart grid, including actors playing roles, flex-offers, and different types of time series. The schema has a number of interesting complexities such as facts about facts and composed non-atomic facts. The different nodes will hold different parts of the data accordingly to the roles of the node owners and the data will be at different aggregation levels at different nodes. The same schema can, however, be used for all kinds of nodes. We have considered different alternatives for the schema modeling using denormalization and arrays, respectively, but based on the performance and space usage, the chosen design is favourable. In the near future, we are going to perform large-scale simulations

with realistic data amounts from different types of nodes. We will also address the challenges with distribution of the data on many nodes such propagation of data through the hierarchy, caching, etc. Further, we plan to investigate the possibilities for having specialized versions of the schema for different types of nodes, but such that queries can be formulated on the generic schema and automatically be translated to the specialized schemas to make the results combinable.

References

1. A. Bauer, W. Hümmer, and W. Lehner. "An Alternative Relational OLAP Modeling Approach". In *Proc. of DaWaK*, 2000.
2. M. Boehm et al. "Data Management in the MIRABEL Smart Grid System". In *Proc. of EDBT/ICDT Workshops*, 2012.
3. M. Böhlen, J. Gamper, and C.S. Jensen. "Multi-dimensional Aggregation for Temporal Data". In *Proc. of EDBT*, 2006.
4. European Network of Transmission System Operators for Electricity. *The Harmonised Electricity Market Role Model*, version 2011-01. Available at `www.ebix.org/Documents/role_model_v2011_01.pdf` as of 2012-06-12.
5. Introduction to Business Requirements and Information Models. Available at `www.ebix.org/documents/Introduction\%20to\%20ebIX\%20Models\%200.0.D.pdf` as of 2012-06-12.
6. IEC61970-301 Ed. 2, *Energy management system application program interface (EMS-API) - Part 301: Common information model (CIM) base*, International Electrotechnical Commission, 2009.
7. C.S. Jensen, T.B. Pedersen, and C. Thomsen. *Multidimensional Databases and Data Warehousing*. Morgan & Claypool, 2010.
8. M.J. Konsman and F.J. Rumph. *MIRABEL Deliverable 2.3: Final data model, specification of request and negotiation messages and contracts*. Available at `wwwdb.inf.tu-dresden.de/miracle/files/deliverables/M18/D2.3_final.pdf` as of 2012-06-12.
9. E. Malinowski and E. Zimányi. *Advanced Data Warehouse Design From Conventional to Spatial and Temporal Applications*. Springer, 2009.
10. `www.meregio.de/en/` as of 2012-06-12.
11. `www.mirabel-project.eu/` as of 2012-06-12.
12. `postgresql.org` as of 2012-06-12.
13. I-Y. Song et al. "An Analysis of Many-to-Many Relationships Between Fact and Dimension Tables in Dimensional Modeling". In *Proc. of DMDW*, 2001.
14. L. Siksnys, M.E. Khalefa, and T.B. Pedersen. "Aggregating and Disaggregating Flexibility Objects". In *Proc. of SSDBM*, 2012.
15. J. Zubcoff, J. Pardillo, and J. Trujillo. "A UML profile for the conceptual modelling of data-mining with time-series in data warehouses". *Information and Software Technology*, 51(6):977–992, 2008.