David Evans • 10601514 • devans • davidjsevans@gmail.com

Database Programming Assignment 2

Forecasting Electricity Demand

Contents

The problem

Electricity traders must bid into the market daily for the amount of electricity that their customers will need. If they buy more energy than is consumed, they do not get any money back. If they buy less energy than is consumed, they will also lose money. Thus, they need an accurate energy demand forecast before they bid, in order to operate efficiently.

The task

The task is to forecast energy demand:

- for each TNI-LR-FRMP combination at half-hour intervals
- calculated two weeks in advance
- flagged as STATEMENT_TYPE = 'FORECAST'
- based on averaged historical demand by days of the week or holidays i.e. "the average demand of all previous Tuesdays" or "the average demand of all previous holidays". For TNIs with no past data for holidays, Sundays must be used
- based on actual values i.e. STATEMENT_TYPE != 'FORECAST'
- time-stamped by updating the CHANGE_DATE field

The solution should be:

- well-designed
- professionally documented
- modular
- maintainable

How to effectively run the program

The program is stored the database in a single package named Assignment 2. The package specification contains the names of all procedures and functions with commentary describing their purpose, inputs and outputs. It also contains a comment section with scripts required to initialise the forecast and parameter tables required to run the program.

The package body contains the code for all functions and procedures, with the same commentary describing their purpose, inputs and outputs. There are also comments highlighting business rules contained in the code. The code has been written to be self-evident, with intuitive names and functionality.

The package can be viewed easily in a development environment such as SQL developer, or queried in any environment by executing the following SQL statement:

```
SELECT *
FROM USER_SOURCE
WHERE NAME = 'ASSIGNMENT2'
ORDER BY TYPE, LINE;
```

The program can be run by executing the following PLSQL anonymous block:

```
BEGIN
```

```
ASSIGNMENT2.RM16_FORECAST;
END;
```

Functional Requirements

Based on the specification, functional requirements were defined as the following:

ID	Description
FR1	**Forecast electricity demand**
FR1.1	The forecast should be split by TNI-FRMP-LR-DAY-HH combination
FR1.2	The forecast should be based on the average of prior combinations, with all fields matched except for day, which should be matched as per FR1.3
FR1.3	Day should be matched to past data by the day of the week or holiday.
FR1.3.1	Regular days (non-holidays) should be matched by the day of the week e.g. Mondays to Mondays. When there is no past data for a given day of the week, no forecast will be produced.
FR1.3.2	Holidays should be matched to past holidays (holiday dates are defined as those occurring in the DBP_HOLIDAY table). Where past holidays are not available, holidays should be matched to past Sundays. When past holidays and Sundays are not available, no forecast will be produced.
FR1.4	The forecast should be flagged as such with STATEMENT_TYPE = 'FORECAST'
FR1.5	The forecast should be time-stamped with CHANGE_DATE = sysdate
FR1.6	The program's progress should be logged into a table using the common.log() procedure

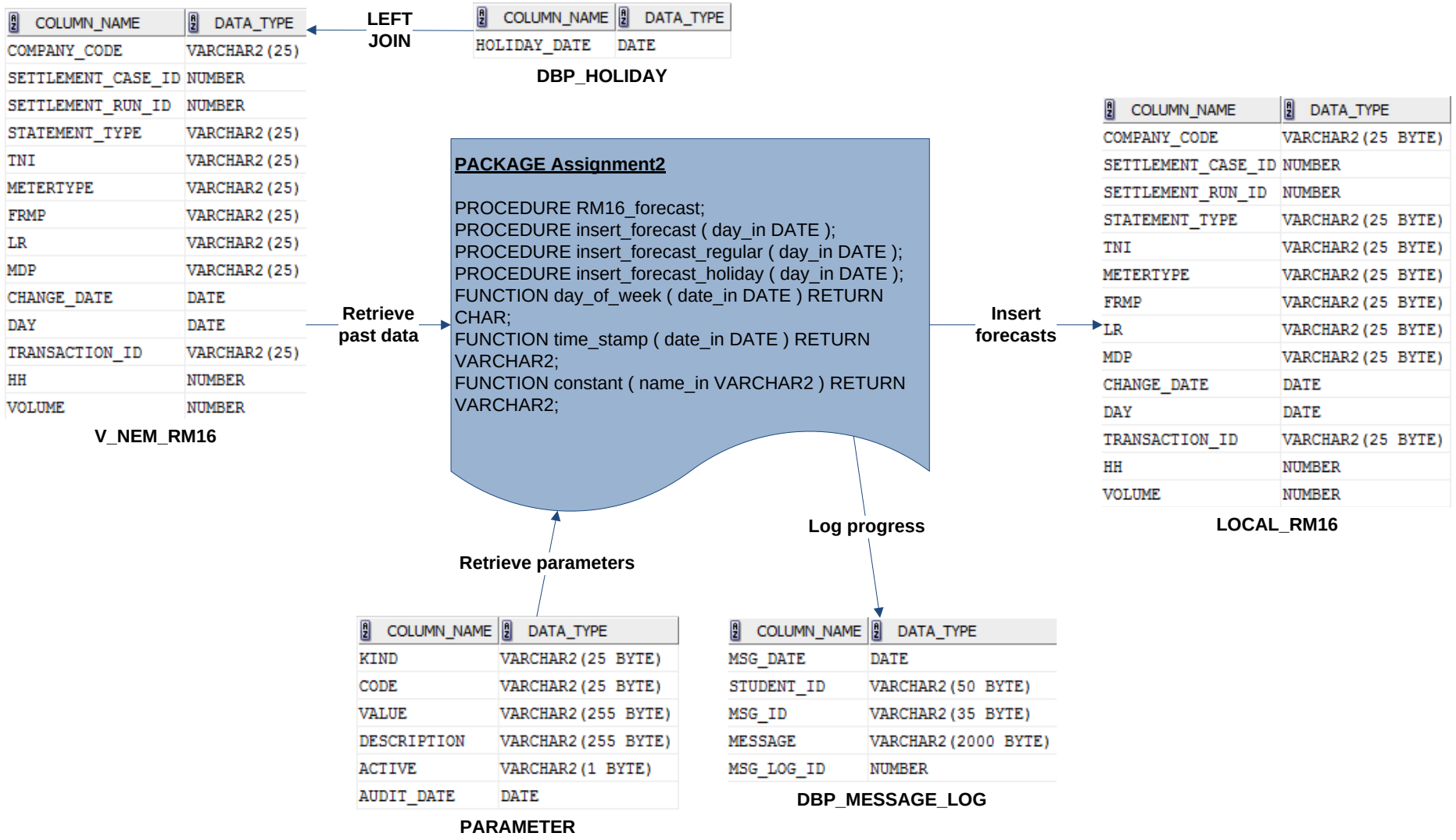
Non-Functional Requirements

Based on the specification, non-functional requirements were defined as the following:

ID	Description
NFR1	**Maintainability**
NFR1.1	Document the approach to the solution
NFR1.2	Document the code by using logical variable naming and comments where necessary
NFR1.3	Build the code in distinct modules which can be easily changed
NFR1.4	Expose appropriate program constants in a parameter table so that they can be updated without needing to change the program
NFR2	**Testability**
NFR2.1	Name the package and main procedure Assignment2.RM16_forecast so that it can be accessed by the standard automatic testing
NFR3	**Completeness**
NFR3.1	Provide the fullest set of forecasts achievable from the past data available i.e. FR1 is achieved whenever possible
NFR3.2	All data in LOCAL_RM16 table should be valid, correct records
NFR4	**Reliability**
NFR4.1	Include exception handling to capture errors
NFR5	**Usability**
NFR5.1	Ensure the code is easy to run for a novice user

Design of the Solution

The table design and data flow is specified by the client as follows:



Build

The solution was originally built using functions to determine details such as the day of the week and whether a day was a holiday. This approach proved to be too slow, with the functions hitting the DBP_HOLIDAY table for every date analysed – the run time was about 100 seconds.

The solution was rewritten, with the DBP_HOLIDAY data being accessed through a left join on the V_NEM_RM16 table view. The SQL query did the same work with a single hit to the database.

Appropriate parameters were also removed to the parameter table and made accessible via the *constant (name_in VARCHAR2)* function. They include the number of days to forecast and the date format of logging timestamps.

Unit Testing

Pieces of code were tested in various ways. It usually involved checking forecast results against manually calculated forecasts. Sometimes, sections of control structures were commented out in order to see if they or others were being reached.

Acceptance Testing

Acceptance testing was performed by verification of the functional and non-functional requirements:

Functional Requirements

ID	Description	Results
FR1	**Forecast electricity demand**	
FR1.1	The forecast should be split by TNI-FRMP-LR-DAY-HH combination	All forecasts are split appropriately.
FR1.2	The forecast should be based on the average of prior combinations, with all fields matched except for day, which should be matched as per FR1.3	All combinations are matched appropriately – see testing for FR1.3.
FR1.3	Day should be matched to past data by the day of the week or holiday.	
FR1.3.1	Regular days (non-holidays) should be matched by the day of the week e.g. Mondays to Mondays. When there is no past data for a given day of the week, no forecast will be produced.	• Average of Saturdays for TNI 'NALB' is 0.007245 – the same figure inserted for forecasts Saturday 9 June and Saturday 16 June. • TNI 'NCTB' only has one Tuesday of past data – only Tuesdays are forecasted for.
FR1.3.2	Holidays should be matched to past holidays (holiday dates are defined as those occurring in the DBP_HOLIDAY table). Where past holidays are not available, holidays should be matched to past Sundays. When past holidays and Sundays are not available, no forecast will be produced.	• Average of holidays for TNI 'NALB' is 0.008648 – the same figure inserted for the forecast of Monday 11 June (an upcoming holiday). • Average of Sundays for TNI 'VTBT' is 0.001806 – the same figure inserted for the forecast of Monday 11 June. • TNI 'NCTB' only has one Tuesday of past data – there is no holiday data or

		Sunday data, so Monday 11 June has no forecast.
FR1.4	The forecast should be flagged as such with STATEMENT_TYPE = 'FORECAST'	All inserted forecasts are flagged STATEMENT_TYPE = 'FORECAST'
FR1.5	The forecast should be time-stamped with CHANGE_DATE = sysdate	All inserted forecasts are time-stamped CHANGE_DATE = sysdate
FR1.6	The program's progress should be logged into a table using the common.log() procedure	Progress logged appropriately, indicating forecasted period and with time-stamps reflecting the run time: [06/JUN/12 16:10:28] BEGIN: Forecasting for 06/JUN/12 to 20/JUN/12 started. [06/JUN/12 16:10:43] END: Forecasting for 06/JUN/12 to 20/JUN/12 complete.

Non-Functional Requirements

ID	Description	Results
NFR1	**Maintainability**	
NFR1.1	Document the approach to the solution	User and technical documentation are provided (this document)
NFR1.2	Document the code by using logical variable naming and comments where necessary	Logical, consistent variable names are used. Comments explain: the purpose of the program, how to execute the program, how to modify the program constants, the purpose of the modules and notable aspects of the modules such as inputs and outputs
NFR1.3	Build the code in distinct modules which can be easily changed	The code is built in four procedures and three function modules. • **Constants** are retrieved from the parameter table using the *constant (name_in VARCHAR2)* function. This allows them to be changed without changing the program. • **Private variables** ensure that they have low coupling and high cohesion, to allow them to be modified without breaking the program. Sections of control structures are broken down into modular procedures to make the code easier to understand and test. • **Business rules** are highlighted where they are implemented through the code, in case they need to be changed.
NFR1.4	Expose appropriate program constants in a parameter table so that they can be updated without needing to change the program	Program constants are exposed in the "parameter" table, with the initialisation script included in the Assignment2 package specification comments.

NFR2	**Testability**	
NFR2.1	Name the package and main procedure Assignment2.RM16_forecast so that it can be accessed by the standard automatic testing	Package/procedure is named Assignment2.RM16_forecast as required

NFR3	**Completeness**	
NFR3.1	Provide the fullest set of forecasts achievable from the past data available i.e. FR1 is achieved whenever possible	• If all TNIs had sufficient data, this would result in 72 * 14 * 48 = 48 384 forecasts. As this is not the case, the program inserts the maximum possible forecasts at 35 520 with the current data set. • Missing forecasts are confirmed to be due to TNIs without data for particular days of the week, holidays, or Sundays in their absence. **All TNIs with sufficient historical data are forecasted for.**
NFR3.2	All data in LOCAL_RM16 table should be valid, correct records	Spot-checking has confirmed that all records are valid and correct, but the development of an automated test could provide 100% confirmation.

NFR4	**Reliability**	
NFR4.1	Include exception handling to capture errors	Exception handling is present throughout the program. In procedures and functions where an error would be fatal, the state of the database is rolled back and appropriate error messages are printed to the console. In the *time_stamp(date_in DATE)* function, an exception would not be fatal and the input date is simply returned unformatted, with the program allowed to continue execution. This will avoid an error caused by an invalid date format in the parameter table.

NFR5	**Usability**	
NFR5.1	Ensure the code is easy to run for a novice user	The code is easy to run: the user just needs access via a user interface described in "How to effectively run the program" and to follow the documented steps, selecting the relevant statements and executing them.