It is necessary to establish a limit on the amount of workload a reactor can sustain while properties, such as coupling, are maintained within reactors. Given the access frequency, the table coupling, and the maximum workload limit, our technique aims to divide workload among clusters by properly allocating tables and interfaces to reactors. It is noteworthy that this work only considers interfaces that enable GET and POST operations. The parameters and the decision variables are defined as follows.

Parameters

$access_i$	access frequency for interface i;
$coup_{i,j}$	coupling degree between tables i and j;
Q	the maximum access frequency load a reactor can sustain.

$$post_{i,j} = \begin{cases} 1, & \text{if POST for table } j \text{ is fulfilled through interface } i \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I, \ \forall j \in T$$

$$get_{i,j} = \begin{cases} 1, & \text{if GET for table } j \text{ is fulfilled through interface } i \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in I, \ \forall j \in T$$

Decision variables

$tb_{k,i}$	equals 1 if table *i* is allocated to reactor type *k*, and 0 otherwise;
$int_{k,i}$	equals 1 if interface *i* is allocated to reactor type *k*, and 0 otherwise.

Based on the parameters and decision variables, we introduce the model designed to optimize the definition of reactors.

$$\max \quad \sum_{k=1}^{n} \omega_k \tag{1}$$

The objective function (1) maximizes the coupling level among tables allocated within each cluster. In order words, we aim to keep associated tables together as maximum as possible in order to reduce communication costs in case of JOIN operations.

$$\sum_{k=1}^{n} tb_{k,i} = 1 \qquad \forall i \in T \tag{2}$$

$$\sum_{k=1}^{n} int_{k,i} = 1 \qquad \forall i \in I \tag{3}$$

Constraints (2) and (3) limit the maximum number of clusters each table and interface are allowed to be allocated to, respectively. Our objective is to allocate a given table (or interface) to only one cluster.

$$int_{k,i} - tb_{k,j} = 0 \qquad where \quad post_{i,j} = 1 \quad \forall k = 1, ..., n \tag{4}$$
$$int_{k,i} - tb_{k,j} = 0 \qquad where \quad get_{i,j} = 1 \quad \forall k = 1, ..., n \tag{5}$$
$$\sum_{k=1}^{n} \sum_{i \in I} \sum_{j \in T} int_{k,i} + tb_{k,j} = 2 \qquad where \quad post_{i,j} = 1 \tag{6}$$
$$\sum_{k=1}^{n} \sum_{i \in I} \sum_{j \in T} int_{k,i} + tb_{k,j} = 2 \qquad where \quad get_{i,j} = 1 \tag{7}$$

Constraints (4)-(7) force the table that represents a resource to be allocated in the same cluster as its respective GET and POST operations are allocated to (through respective interfaces). In addition, constraints (4)-(7) force GET and POST interfaces to be allocated in the same cluster. In summary, if a table is allocated to a cluster, we force its POST and GET interface to be allocated in the same cluster.

$$\alpha_k = \sum_{i \in T} tb_{k,i} \qquad \forall k = 1, ..., n \tag{8}$$
$$\omega_k = \sum_{i \in T} \sum_{j \in T/i} tb_{k,i}.tb_{k,j}.coup_{i,j} \qquad \forall k = 1, ..., n \tag{9}$$
$$\alpha_k \leq \omega_k + 1 \qquad \forall k = 1, ..., n \tag{10}$$
$$\sum_{i \in I} int_{k,i}.access_i \leq Q \qquad \forall k = 1, ..., n \tag{11}$$
$$\sum_{i \in I} int_{k,i} \geq \sum_{j \in T} tb_{k,j} \qquad \forall k = 1, ..., n \tag{12}$$

Constraints (8) represent the total number of tables a given cluster holds. Constraints (9) represent the level of coupling a given cluster holds. Constraints (10) force tables in a cluster to have a positive coupling level. In other words, if there is more than a table in the cluster, the tables must be associated through FK. Constraints (11) limit the maximum access frequency a cluster can sustain. Constraints (12) restrict the existence of a cluster with tables and no interfaces.

$$int_{k,i} \in \{0, 1\} \tag{13}$$
$$tb_{k,i} \in \{0, 1\} \tag{14}$$

Constraints (13-14) are integrity constraints.

Reactor function extraction. In order to identify application logic that would be more efficiently executed by the DBMS, Cheung et al. [Cheung et al. 2012] assert that "programmers must identify sections of code that make multiple (or large) database accesses and can be parameterized by relatively small amounts of input". Based on this observation, this step aims at identifying source code lines with: (*i*) high degree of data access and manipulation, and (*ii*) low complexity. Thus, we seek for the identification of application logic in business layer to be migrated to a reactor function (*RAF*). Equation