**OBJECTIVE:** This is a worked example of the type of design information that I am looking for when you migrate from a relational database management system (RDBMS) design to a denormalized document-based database such as MongoDB.

**INTRODUCTION:** A good reference for how to make design decisions regarding migrating associations from a relational design to a document-based database can be found [here](https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design). For this example, we are going to concentrate on these design patterns. Other design patterns can be useful as well, but in the interests of time, I want to focus your attention on just these six approaches.

**PROCEDURE:** The following table is the results of Professor Terrell's analysis of our Practice SQL database structure, formatted in the way that I want to see phase one of the term project done.

Please note that all the surrogate keys from the Practice SQL will be implemented using the \_id attribute that MongoDB supplies. For instance, officeCode from offices goes into \_id in the offices collection, employeeNumber goes into the \_id attribute in the employees collection and so forth. Instead of letting MongoDB assign values to these surrogates, the JSON files for the Practice SQL supply an integer value for each \_id, and that is the same value that was in the original relational table surrogate key.

| **Parent Table Name** | **Child Table Name** | **Multiplicity** | **Direction** | **Design Pattern** | **Comments** | **Rationale** |
| --- | --- | --- | --- | --- | --- | --- |
| students | courserequirements | 1..1 to 0..\* | Parent to child | One to few | Each courserequirement document has an array of courses. Courses disappears as a collection because the payments are embedded into customers. | The courserequirements for each student tend to be few. Each is small. The size of the customers document should stay relatively small. |
| customers | orders | 1..1 to 0..\* |  | One to many | Create an array of order numbers in customers, much like a bi-directional relationship in SQLAlchemy. | There appears to be no single set of attributes in orders that we need more than the others, so there is no compelling drive to add other columns from orders into the list of orders. |
| customers | orders | 1..1 to 0..\* | Child to parent | Two-way referencing | In the **orders** collection, embed both the customer ID as well as the customer name. | The customer name frequently comes up in queries originating in the order, so redundantly storing the name with the customer surrogate will save time, at the expense of redundant data. |
| offices | employees | 1..1 to 0..\* | Parent to child | Denormalization from many to one | Offices has an array called "employees" that contains the employeenumber, lastname, firstname, and jobtitle columns. | Those are the columns most often required of an employee, so we accept some redundancy in the data structure in exchange for removing the need for a join between offices and employees in most cases. |
| offices | employees | 1..1 to 0..\* | Child to parent | Denormalization from one to many | employees stores the officeCode, city, state, and country of the office. | These are the columns most often called for from the office, so we buy better performance with redundant data. |
| employees | employees | 1..1 to 0..\* | Parent to child | One to squillions | The parent has no connection to the child at all. | The prediction is that recursive queries to report out reporting relationships will be generally rather rare. There is therefore little to be gained by repeating any of the data. |
| employees | employees | 1..1 to 0..\* | Child to Parent | One to squillions | The child (subordinate employee) has a reference to their manager (the superior employee) | Sticking to the way that this relationship is implemented in the relational database (i.e. with a migrating foreign key) seems to be the most direct approach. |
| orders | orderdetails | 1..1 to 1..\* | Parent to child | One to few | The entire contents of orderdetails is embedded into orders. | The amount of data in each orderdetails row is relatively small, and there are, at most, a few dozen orderdetails rows for each order. |