Database Designs

Section 5: Database Design Theory

5.1 Practical Tips on Database Design

A good database design is when a database can withstand the test of time. If the data structure does not change that much even when the data grows and expands into new scenarios/areas of information then this can signify a good good database design. This is a hard topic and so much can be said about this topic alone. Below are a few important principles of designing a good schema for a database.

Designing the tables of a database is called designing the schema of a database.The list below is not a comprehensive list but will help when setting up a database to withstand the test of time.

- Identify the problem you are trying to model/capture from the real worlds
- Have a good idea of the types of queries (i.e. information to drive business decisions) that you want to run on the database
- Break up the information into logical units where each logical unit is one table in your database
- Each logical component will probably connect with other logical components. Capture the relationships either via a constraint or other tables devoted to holding relationship specific data

Tables typically hold information about an entity or a relationship. Think of whether you need a way to uniquely identify each row in a table. If the answer to this question is yes (in most circumstances this will be true) you need a Primary Key.

- A Primary Key can never have a NULL value
- A Primary Key will be used to represent that row and other tables can reference that row by specifying the Primary Key value
- Do not use data that can change as a Primary Key value
- If there are no such values in the fields you want to store then you can generate a value which will be used as a unique identifier i.e. DBMS have auto-increment to generate unique identifiers for you
- Numeric values tend to be more efficient in representation and lookup and should be preferred for Primary Key values
- Having an identifier whose job is to be a key value is standard practice
- Remember, most DBMS will automatically create an index on the Primary Key for faster lookup and retrieval

Do columns have other constraints which they need to satisfy?

- Are NULL values allowed for any column? If the answer is no then the column should be specified as NOT NULL
- Should columns have a default value? If the answer is yes then the column should be setup with the DEFAULT keyword followed by what the default value for that column should be

- Ensure the column datatype are setup correctly e.g. don't use a string for storing date information (even though it is allowed) as this will limit the features applicable to date and time values for example comparing date difference between two date values
- Think about the number of characters/numbers a column will hold e.g. if a data is a single character don't use a CHAR(30) to store the string. Database occupy space and space that are not used up are a waste of resource and storage resource is expensive

5.2 Further Tips on Database Design (Table Relationships)

How many tables should we use to store information? To answer this question we should always think of the relationship between the different pieces of information being represented.

If the information is a one-to-one compulsory information e.g. students and their email address there is no need to store the information is separate tables. Every student has just one email address and therefore there is no need for the email address to be stored in a separate table from the rest of the students information.

When the relationship between the various pieces of information are one-to-one and the information is not optional (i.e. it is always specified) then we can put that information into the same table. Remember this as a rule of thumb. You may want to separate it for some reason but typically if the information has a one-to-one relationship there is no reason to separate the data.

StudentID	Name	Email
1	John Doe	j.doe@email.com
2	Martha Jones	m.jones@email.com

If the information is a one-to-one but one of the information is optional e.g. students and siblings information you may decide to hold these information in separate tables. A student may or may not have siblings and if the main students table included sibling information then there would be many empty cells. In this case the students table will hold only the mandatory information while the sibling information should be held in a separate table.

When the relationship between the various pieces of information are one-to-one but some information are optional (i.e. may or may not exist) then it may be best to separate the information keeping all the mandatory information in one table and the optional information in another. The main table will hold a Primary Key as an identifier for each row. The second optional data table would reference the main table's Primary Key value as a Foreign Key value, thereby linking both the table information together.

Optional data are represented by the lack of rows in the second table and not by empty cell value in a row. This is a much cleaner setup.

StudentID	Name	Email
1	John Doe	j.doe@email.com
2	Martha Jones	m.jones@email.com

StudentID	Sibling Name
1	Joanne Doe
1	Jonathon Doe
2	Katey Jones

If the information is a one-to-many or a many-to-one relationship e.g. employees and managers information then at a minimum we would need two tables to represent this information. One table would be used to store all information relating to employees and another to hold the relationship between an employee and his/her manager.

Notice in the example table below, the second table uses only the unique identifiers to specify the relationship. Once again, the Primary Key is used on the main table to represent the entire row details of the employee. The second table merely makes reference to these keys only.

This is not the only way to represent this information and there are different ways of representing this data but typically speaking a one-to-many or many-to-one relationship is best represented using two tables at a minimum and what the two tables contain can be designed differently.

EmployeeID	Name	Email
1	John Doe	j.doe@email.com
2	Martha Jones	m.jones@email.com
3	Katey Jones	k.jones@email.com

EmployeeID	Manager_EmployeeID
1	2
3	2

Finally if the information is a many-to-many relationship e.g. stores, products and revenue information then we would need 3 or more tables to represent the information. The revenue data is specified for every product sold in every store and therefore links the stores and products data in a many-to-many relationship.

In such a case we would need at a minimum 3 tables, the first table representing the first entity's information, the second table representing the second entity's informations and the third table linking the two entities tables in a many-to-many relationship.

StoreID	StoreLocation
1	Birmingham
2	Coventry

ProductID	ProductName
1	Bread
2	Milk
3	Nutella

StoreID	ProductID	Date	Revenue
1	1	30 November 2020	453.22
1	3	30 November 2020	225.55
2	2	30 November 2020	1860.98

It is best to keep the entities information in separate tables. If we were to add more details about the product e.g. product details, if we did not have a separate table for stores and products, we would end up adding more and more information to the single table making the table fatter even if the information has nothing to do with the store itself. Separate tables allows us to expand the information logically while still maintain tall and thin structures. The linking table itself will have a tall and fat structure.

The linking table would use Foreign Keys to link the the Primary Key values of the other entity tables which combines all three tables together in a many-to-many relationship. This provides a clean structure for specifying a many-to-many relationship.

Understanding the relationships of the information you are trying to represent is vital in setting up the tables and its column in the best way possible to stand the test of time.

5.3 Normal Forms in Database Design

Database Normalisation i.e. Normal Forms are basically rules that tell you how a good relational database should be designed. Database theory provides us the standard rules to test if a table is well designed. These rules are specified in the form of Normal Forms (and there are many available to us). Normal Forms are very inaccessible, but if you can understand them they are very good rules that are well thought through and can help improve the design of the database table. Below are some examples for Normal Forms translated to normal understandable english.

1NF - “First Normal Form”:
“A relation is in first Normal Form if the domain of each attribute contains only atomic (indivisible) values and the value of each attribute contains only a single value from that domain.”

This means that a table should not have any column with concatenated/compound values. If this is the case, the data should be split into multiple columns which are atomic/ indivisible values. This will allow greater flexibility with the way you specify attributes of a particular row.

For example the table below would be best split into separate columns to make comparisons, ordering, grouping, etc. much easier to perform. The second table below follows the First Normal Form rule.

StudentID	Sudent_FirstName_LastName	Subject_GraduatingClass
1	John Doe	CS_2020
2	Martha Jones	ECE_2019

StudentID	FirstName	LastName	Subject	GraduatingClass
1	John	Doe	CS	2020
2	Martha	Jones	ECE	2019

2NF - “Second Normal Form”:
“A table is in 2NF if it is in 1NF and no non-prime attribute is dependant on any proper subset of any candidate key of the table. (A non-prime attribute of a table is is an attribute that is not a part of any candidate key of the table).”

This means a table should have every non-key column fully defined by or dependant on a Primary Key.

Below is an example of a table that violates the 2NF rule. The Primary Key for the table is the combination of StoreID, ProductID and Date columns which provides the unique information of the revenue of a particular product sold by a particular store on a particular date. There are two additional non-key columns.

The Revenue column is clearly defined i.e. “identified” by the Primary Key(s). The Revenue column as a value makes absolute sense for that particular logical unit that the Primary Keys specifies (i.e. for a particular store, product and date the revenue was X amount).

The CategoryManager column is not clearly identified by the Primary Key(s) i.e. while the CategoryManager can be for a product or a product in a store; however, the date for the CategoryManager does not make any sense here.

StoreID	ProductID	Date	Revenue	CategoryManager
Birmingham	Bananas	30 November 2020	2245.60	John
Coventry	Bananas	30 November 2020	4455.00	Martha

In this scenario the CategoryManager violates the 2NF rule and so we would extract the CategoryManager from this table and create a separate products or products_store table where it would make sense for the data to be present in the new table.

3NF - “Third Normal Form”:
“A table is in 3NF if
(1) The Entity is in 2NF, and
(2) All the attributes in a table are determined only by the candidate keys of that table and not by any non-prime attributes.”

This means a table should have all non-key columns independent of each other and should not be related in any way.

In the below example table the StoreID is a unique identifier for the store and therefore is the Primary Key for this table. All other columns are non-key columns. If we were to combine all of the non-key columns data together we would notice they are closely related to each other and not completely independent of one another. In fact using only the storeAddress we could figure out the StoreCity and the StoreCountry. Therefore, the city and country follow from the store location and are not independent columns and should not be part of the below table. Instead the StoreCity and StoreCountry should move into a separate tabled Keyed by the Store Location. You should split up the tables so that the data follows 3NF as seen in the second table example below

StoreID	StoreAddress	StoreCity	StoreCountry
1	84 Boroughbridge Road	Birmingham	England
2	57 Whatlington Road	Coventry	England

StoreID	StoreAddress
1	84 Boroughbridge Road
2	57 Whatlington Road

StoreID	StoreCity	StoreCountry
1	Birmingham	England
2	Coventry	England

(Note: this is one possible way of splitting the store location data into multiple tables).

There are other Normal Forms rules such as Boyce-Codd Normal Form (BCNF), Forth Normal Form (4NF) and Fifth Normal Form (5NF) rules.