Database Designing Guiding Principal for GEA 2.0

Table of contents

1. Introduction

1.1. Purpose of this Document

This guideline in ensuring consistent data, elimination of data redundancy, efficient execution of queries and high performance application. Taking the time to design a database saves time and frustration during development, and a well-designed database ensures ease of access and retrieval of information.

Standardize design process is required to the data exchange framework and Master data management across the enterprise.

1.2. Structure of this Document

This document has been split into three distinct part.

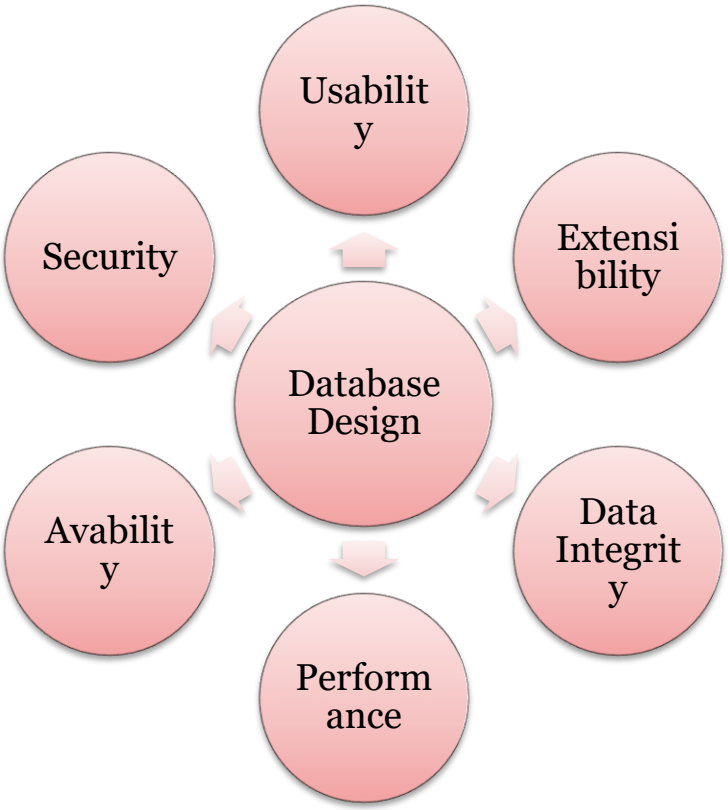
1. Database Design Guiding Principal
2. Database Design process
3. Outcome – Detail data model

2. Database Designing Guiding Principal for GEA 2.0

Database must be organized, stored, secured, and should be readily available in a usable format for daily operations and analysis by individuals, groups, and processes, both for current and future use.

There are six main objectives, which must be fulfilled effectively by a standard database.

1. Usability
2. Extensibility
3. Data Integrity
4. Performance
5. Availability
6. Security



2.1. Usability

Any information which are storing in any organization should be meaningful for that organization. If storing those factors which are actually not fit with organization's requirement then this is just waste of resources.

Primary objective of any information system should be to meet organization requirements. Following are few points to consider while going to start an architecture.

1. Properly get details about requirements.
2. See how information can be fit with requirement.
3. Trace requirement matrix to capture mapping of information architecture and requirements.
4. Organize it simple.
5. Decide upon format of data so that could be easily converted to meaningful representation.

2.2. Extensibility

Everyday new business requirements come up and every day there is a need to change or enhance information system to capture new requirements. So information design should be extensible so that it can adopt new requirements without much efforts or without major breaking changes.

If your initial design is too much complex or unorganized then it may create trouble for you to adopt new things effectively.

Following are few points to consider when thinking of extensibility.

- Normalization and correct handling of optional data.
- Generalization of entities when designing the schema.
- Data-driven designs that not only model the data but also enable the organization to store the behavioral patterns or flow which can be hooked up in different stages of information processing.
- A ll-defined abstraction layer that decouples the database from all client access, including client apps, middle tiers, ETL, and reports.
- Extensibility is also closely related to simplicity. Complexity breeds complexity. A simple solution is easy to understand and adopt, and ultimately, easy to adjust later.

2.3. Data Integrity

Now at this point understand that information is very much important for any organization. Based on the historic information, every organization makes different strategies, decisions for growth. One small mistake in data can lead to major issues with any organization's key decision and hence a big risk for growth.

When designing a good information system then must keep in mind about integrity, correctness of data. Our system should be smart enough to handle incorrect, missing data attributes and based on that it should either take corrective actions or straightaway reject the data. Incorrect data should not be present in system or at least should not exposed to individuals creating misunderstanding.

Data integrity can be of many types.

2.3.1. Entity Integrity

Involves the structure (primary key and its attributes) of the entity. If the primary key is unique and all attributes are scalar and fully dependent on the primary key, then the integrity of the entity is good. In the physical schema, the table's primary key enforces entity integrity.

2.3.2. Domain Integrity

It defines that data should be of correct type and should handle optional data in correct way. Should apply Null ability to those attributes, which are optional for organization. can define proper data types for different attributes based on organization's requirement so that correct format data should present in system.

2.3.3. Referential Integrity

This defines if any entity is dependent on another one then parent entity should be there in the system and should be uniquely identifiable. can do this by implementing foreign keys.

2.3.4. Transactional Integrity

This defines that transaction should have its ACID properties. Any transaction should be atomic, consistent, durable and isolated. The quality of a database product is measured by its transactions' adherence to the ACID properties:

Atomic — all or nothing

Consistent — the database begins and ends the transaction in a consistent state

Isolated — one transaction does not affect another transaction

Durable — once committed always committed

2.3.5. User defined integrity

There are few business rules which cannot validate just by primary keys, foreign keys etc. There has to be some mechanism so that can validate complex rules for integrity. can implement these rules in following ways:

- Check Constraints (Specify range of values of a colum)
- Triggers & Stored Procedures
- Queries to identify incorrect data and handle in correct way.

2.4. Performance

Information should be readily available as requested. Performance of the system should be up to the mark. As data in increasing day by day so at some time there will be impact on performance if database design is poor or 'll not take any actions to improve performance.

Following could be few strategies which can implement when there is need as data increases.

- All-designed schema with normalization and generalization
- A sound indexing strategy, including careful selection of clustered and no clustered
- Tight, fast transactions that reduce locking and blocking

- Partitioning, which is useful for advanced scalability

2.5. Availability

The availability of information refers to the information's accessibility when required regarding uptime, locations, and the availability of the data for future analysis. Disaster recovery, redundancy, archiving, and network delivery all affect availability.

Following are the factors, which impact availability:

- Quality, redundant hardware
- Server's high-availability features
- Proper DBA procedures regarding data backup and backup storage
- Disaster recovery planning

2.6. Security

For any organizational asset, the level of security must be secured depending on its value and sensitivity. Sometime organizations has suffered a lot because of data leaks which results in loss of faith and tends to business risk. So security is one of the most important aspect of good database design.

Can enhance Security by the following:

- Physical security and restricted access of the data center
- Defensively coding against SQL injection
- Appropriate operating system security
- Reducing the surface area of SQL Server to only those services and features required
- Identifying and documenting ownership of the data
- Granting access according to the principle of least privilege, which is the concept that users should have only the minimum access rights required to perform necessary functions within the database
- Cryptography — data encryption of live databases, backups, and data warehouses
- Metadata and data audit trails documenting the source and veracity of the data, including updates

Based on above principles, one should start designing databases and architectures.

3. The Design Process

The design process consists of the following steps:

3.1. Determine the purpose of your database

Its purpose, how expect to use it, and who will use it. For a small database for an internal application, for example, you might write something simple like "The citizen database keeps a list of citizen information for the purpose of producing mailings and reports." If the database is more complex or is used by many people, as often occurs in a corporate setting, the purpose could easily be a paragraph or more and should include when and how each person will use the database. The idea is to have a developed mission statement that can be referred to throughout the design process. Having such a statement helps you focus on your goals when you make decisions.

3.2. Finding and organizing the required information

To find and organize the information required, start with your existing information. For example, you might record purchase orders in a ledger or keep citizen information on paper forms in a file cabinet. Gather those documents and list each type of information shown (for example, each box that you fill in on a form). If you don't have any existing forms, imagine instead that you have to design a form to record the citizen information. What information would you put on the form? What fill-in boxes would you create? Identify and list each of these items. For example, suppose you currently keep the citizen list on index cards. Examining these cards might show that each card holds a citizens name, address, city, state, postal code and telephone number. Each of these items represents a potential column in a table.

As you prepare this list, don’t worry about getting it perfect at first. Instead, list each item that comes to mind. If someone else will be using the database, ask for their ideas, too. You can fine-tune the list later.

Next, consider the types of reports or mailings you might want to produce from the database. For instance, you might want a product sales report to show sales by region, or an inventory summary report that shows product inventory levels. You might also want to generate form letters to send to citizens that announces a sale event or offers a premium. Design the report in your mind, and imagine what it would look like. What information would you place on the report? List each item. Do the same for the form letter and for any other report you anticipate creating.

Giving thought to the reports and mailings you might want to create helps you identify items you will need in your database. For example, suppose you give citizens the opportunity to opt in to (or out of) periodic e-mail updates, and you want to print a listing of those who have opted in. To record that information, you add a “Send e-mail” column to the citizen table. For each citizen, you can set the field to Yes or No.

The requirement to send e-mail messages to citizens suggests another item to record. Once you know that a citizen wants to receive e-mail messages, you will also need to know the e-mail address to which to send them. Therefore you need to record an e-mail address for each citizen.

It makes good sense to construct a prototype of each report or output listing and consider what items you will need to produce the report. For instance, when you examine a form letter, a few things might come to mind. If you want to include a proper salutation — for example, the "Mr.", "Mrs." or "Ms." string that starts a greeting, you will have to create a salutation item. Also, you might typically start a letter with “Dear Mr. Smith”, rather than “Dear. Mr. Sylvester Smith”. This suggests you would typically want to store the last name separate from the first name.

A key point to remember is that you should break each piece of information into its smallest useful parts. In the case of a name, to make the last name readily available, you will break the name into two parts — First Name and Last Name. To sort a report by last name, for example, it helps to have the citizen's last name stored separately. In general, if you want to sort, search, calculate, or report based on an item of information, you should put that item in its own field.

Think about the questions you might want the database to ansr. For instance, how many sales of your featured product did you close last month? Where do your best citizens live? Who is the supplier for your best-selling product? Anticipating these questions helps you zero in on additional items to record.

3.3. Dividing the information into tables

To divide the information into tables, choose the major entities, or subjects. For example, after finding and organizing information for a product sales database, the preliminary list might look like this:

The major entities shown here are the products, the suppliers, the citizens, and the orders. Therefore, it makes sense to start out with these four tables: one for facts about products, one for facts about suppliers, one for facts about citizens, and one for facts about orders. Although this doesn't complete the list, it is a good starting point. You can continue to refine this list until you have a design that works ll.

When you first review the preliminary list of items, you might be tempted to place them all in a single table, instead of the four shown in the preceding illustration. You will learn here why that is a bad idea. Consider for a moment, the table shown here:

In this case, each row contains information about both the product and its supplier. Because you can have many products from the same supplier, the supplier name and address information has to be repeated many times. This wastes disk space. Recording the supplier information only once in a separate Suppliers table, and then linking that table to the Products table, is a much better solution.

A second problem with this design comes about when you need to modify information about the supplier. For example, suppose you need to change a supplier's address. Because it appears in many places, you might accidentally change the address in one place but forget to change it in the others. Recording the supplier's address in only one place solves the problem.

When you design your database, always try to record each fact just once. If you find yourself repeating the same information in more than one place, such as the address for a particular supplier, place that information in a separate table.

Finally, suppose there is only one product supplied by Coho Winery, and you want to delete the product, but retain the supplier name and address information. How would you delete the product record without also losing the supplier information? You can't. Because each record contains facts about a product, as ll as facts about a supplier, you cannot delete one without deleting the other. To keep these facts separate, you must split the one table into two: one table for product information, and another table for supplier information. Deleting a product record should delete only the facts about the product, not the facts about the supplier.

Once you have chosen the subject that is represented by a table, columns in that table should store facts only about the subject. For instance, the product table should store facts only about products. Because the supplier address is a fact about the supplier, and not a fact about the product, it belongs in the supplier table.

3.4. Turning information items into columns

To determine the columns in a table, decide what information you need to track about the subject recorded in the table. For example, for the Citizens table, Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail

address comprise a good starting list of columns. Each record in the table contains the same set of columns, so you can store Name, Address, City-State-Zip, Send e-mail, Salutation and E-mail address information for each record. For example, the address column contains citizens' addresses. Each record contains data about one citizen, and the address field contains the address for that citizen.

Once you have determined the initial set of columns for each table, you can further refine the columns. For example, it makes sense to store the citizen name as two separate columns: first name and last name, so that you can sort, search, and index on just those columns. Similarly, the address actually consists of five separate components, address, city, state, postal code, and country/region, and it also makes sense to store them in separate columns. If you want to perform a search, filter or sort operation by state, for example, you need the state information stored in a separate column.

You should also consider whether the database will hold information that is of domestic origin only, or international, as ll. For instance, if you plan to store international addresses, it is better to have a Region column instead of State, because such a column can accommodate both domestic states and the regions of other countries/regions. Similarly, Postal Code makes more sense than Zip Code if you are going to store international addresses.

The following list shows a few tips for determining your columns.

- **Don't include calculated data**

  In most cases, you should not store the result of calculations in tables. Instead, you can have Access perform the calculations when you want to see the result. For example, suppose there is a Products On Order report that displays the subtotal of units on order for each category of product in the database. Hover, there is no Units On Order subtotal column in any table. Instead, the Products table includes a Units On Order column that stores the units on order for each product. Using that data, Access calculates the subtotal each time you print the report. The subtotal itself should not be stored in a table.

- **Store information in its smallest logical parts**

  You may be tempted to have a single field for full names, or for product names along with product descriptions. If you combine more than one kind of information in a field, it is difficult to retrieve individual facts later. Try to break down information into logical parts; for example, create separate fields for first and last name, or for product name, category, and description.

Once you have refined the data columns in each table, you are ready to choose each table's primary key.

3.5. Specify Primary Key

Each table should include a column or set of columns that uniquely identifies each row stored in the table. This is often a unique identification number, such as an employee ID number or a serial number. In database terminology, this information is called the **primary key** of the table. Access uses primary key fields to quickly associate data from multiple tables and bring the data together for you.

If you already have a unique identifier for a table, such as a product number that uniquely identifies each product in your catalog, you can use that identifier as the table's primary key — but only if the values in this column will always be different for each record. You cannot have duplicate values in a primary key. For example, don't use people's names as a primary key, because names are not unique. You could easily have two people with the same name in the same table.

A primary key must always have a value. If a column's value can become unassigned or unknown (a missing value) at some point, it can't be used as a component in a primary key.

You should always choose a primary key whose value will not change. In a database that uses more than one table, a table's primary key can be used as a reference in other tables. If the primary key changes, the change must also

be applied everywhere the key is referenced. Using a primary key that will not change reduces the chance that the primary key might become out of sync with other tables that reference it.

Often, an arbitrary unique number is used as the primary key. For example, you might assign each order a unique order number. The order number's only purpose is to identify an order. Once assigned, it never changes.

If you don't have in mind a column or set of columns that might make a good primary key, consider using a column that has the AutoNumber data type. When you use the AutoNumber data type, Access automatically assigns a value for you. Such an identifier is factless; it contains no factual information describing the row that it represents. Factless identifiers are ideal for use as a primary key because they do not change. A primary key that contains facts about a row — a telephone number or a citizen name, for example — is more likely to change, because the factual information itself might change.

1. A column set to the AutoNumber data type often makes a good primary key. No two product IDs are the same.

In some cases, you may want to use two or more fields that, together, provide the primary key of a table. For example, an Order Details table that stores line items for orders would use two columns in its primary key: Order ID and Product ID. When a primary key employs more than one column, it is also called a composite key.

For the product sales database, you can create an AutoNumber column for each of the tables to serve as primary key: ProductID for the Products table, OrderID for the Orders table, CitizenID for the Citizens table, and SupplierID for the Suppliers table.

3.6. Creating the table relationships

Now that you have divided your information into tables, you need a way to bring the information together again in meaningful ways. For example, the following form includes information from several tables.

1. Information in this form comes from the Citizens table...

2. ...the Employees table...

3. ...the Orders table...

4. ...the Products table...

5. ...and the Order Details table.

Access is a relational database management system. In a relational database, you divide your information into separate, subject-based tables. You then use table relationships to bring the information together as needed.

Determining the relationships between tables helps you ensure that you have the right tables and columns. When a one-to-one or one-to-many relationship exists, the tables involved need to share a common column or columns. When a many-to-many relationship exists, a third table is needed to represent the relationship.

3.6.1. One-to-Many relationship

To represent a one-to-many relationship in your database design, take the primary key on the "one" side of the relationship and add it as an additional column or columns to the table on the "many" side of the relationship.

3.6.2. Many to Many relationship

Consider the relationship between the Products table and Orders table.

A single order can include more than one product. On the other hand, a single product can appear on many orders. Therefore, for each record in the Orders table, there can be many records in the Products table. And for each record in the Products table, there can be many records in the Orders table. This type of relationship is called

a many-to-many relationship because for any product, there can be many orders; and for any order, there can be many products. Note that to detect many-to-many relationships between your tables, it is important that you consider both sides of the relationship.

3.6.3. One to One relationship

Detect the need for a one-to-one relationship in your database, consider whether can be put the information from the two tables together in one table. If don't want to do that for some reason, perhaps because it would result in a lot of empty space, the following list shows how you would represent the relationship in your design:

If the two tables have the same subject, can probably set up the relationship by using the same primary key in both tables.

If the two tables have different subjects with different primary keys, choose one of the tables (either one) and insert its primary key in the other table as a foreign key.

3.7. Refining the design

Once you have the tables, fields, and relationships you need, you should create and populate your tables with sample data and try working with the information: creating queries, adding new records, and so on. Doing this helps highlight potential problems — for example, you might need to add a column that you forgot to insert during your design phase, or you may have a table that you should split into two tables to remove duplication.

See if you can use the database to get the answers you want. Create rough drafts of your forms and reports and see if they show the data you expect. Look for unnecessary duplication of data and, when you find any, alter your design to eliminate it.

As you try out your initial database, you will probably discover room for improvement. Here are a few things to check for:

Did you forget any columns? If so, does the information belong in the existing tables? If it is information about something else, you may need to create another table. Create a column for every information item you need to track. If the information can't be calculated from other columns, it is likely that you will need a new column for it.

Are any columns unnecessary because they can be calculated from existing fields? If an information item can be calculated from other existing columns — a discounted price calculated from the retail price, for example — it is usually better to do just that, and avoid creating new column.

Are you repeatedly entering duplicate information in one of your tables? If so, you probably need to divide the table into two tables that have a one-to-many relationship.

Do you have tables with many fields, a limited number of records, and many empty fields in individual records? If so, think about redesigning the table so it has fer fields and more records.

Has each information item been broken into its smallest useful parts? If you need to report, sort, search, or calculate on an item of information, put that item in its own column.

Does each column contain a fact about the table's subject? If a column does not contain information about the table's subject, it belongs in a different table.

Are all relationships between tables represented, either by common fields or by a third table? One-to-one and one-to- many relationships require common columns. Many-to-many relationships require a third table.

3.8. Applying the normalization rules

You can apply the data normalization rules (sometimes just called normalization rules) as the next step in your design. You use these rules to see if your tables are structured correctly. The process of applying the rules to your database design is called normalizing the database, or just normalization.

Normalization is most useful after you have represented all of the information items and have arrived at a preliminary design. The idea is to help you ensure that you have divided your information items into the appropriate tables. What normalization cannot do is ensure that you have all the correct data items to begin with.

You apply the rules in succession, at each step ensuring that your design arrives at one of what is known as the "normal forms." Five normal forms are widely accepted — the first normal form through the fifth normal form. This article expands on the first three, because they are all that is required for the majority of database designs.

First normal form

First normal form states that at every row and column intersection in the table there, exists a single value, and never a list of values. For example, you cannot have a field named Price in which you place more than one Price. If you think of each intersection of rows and columns as a cell, each cell can hold only one value.

Second normal form

Second normal form requires that each non-key column be fully dependent on the entire primary key, not on just part of the key. This rule applies when you have a primary key that consists of more than one column. For example, suppose you have a table containing the following columns, where Order ID and Product ID form the primary key:

Order ID (primary key)

Product ID (primary key)

Product Name

This design violates second normal form, because Product Name is dependent on Product ID, but not on Order ID, so it is not dependent on the entire primary key. You must remove Product Name from the table. It belongs in a different table (Products).

Third normal form

Third normal form requires that not only every non-key column be dependent on the entire primary key, but that non-key columns be independent of each other.

Another way of saying this is that each non-key column must be dependent on the primary key and nothing but the primary key. For example, suppose you have a table containing the following columns:

ProductID (primary key)

Name

SRP

Discount

Assume that Discount depends on the suggested retail price (SRP). This table violates third normal form because a non-key column, Discount, depends on another non-key column, SRP. Column independence means that you should be able to change any non-key column without affecting any other column. If you change a value in the SRP field, the Discount would change accordingly, thus violating that rule. In this case Discount should be moved to another table that is keyed on SRP.

4. Outcome -- Detailed data model

4.1. Component-1 (Duplicate this section for each data component)

<Provide a brief overview of the data component (database), what its data domain is purpose in the architecture, scope, etc. Include in the description what the relationship of this component has with others. Use cases or other techniques to describe the concept of operation should be used in the discussion.>

4.1.1. Entity-relationship diagram

<Place an ER diagram here.>

4.1.2. Tables

4.1.2.1. Table 1 (Duplicate this Section for Each Table)

<For each table in the database there needs to be a detailed design describing its contents, columns, and relationship to other tables. Should information be migrated from another application, the Source column identifies the source. For databases that are part of an installed application, only those tables related to data that will be used in the solution should be listed, and only columns containing information used in the solution need to be included.>

Table 1:Description

Column. Name	Description	Data Type	Constraints	Mandatory	Unique	Source
		Number (10), Varchar2 (20) etc.	PK, FK	yes/no	yes/no	

4.1.3. Indexes

<This section lists the indexes defined in the database. If databases are part of an installed application, only those indexes related to share data as part of an integration should be listed.>

Indexes

Index name	Table	Column	Unique
			yes/no

4.1.4. Triggers

<This section describes defined triggers in the database. If databases are part of an installed application, only those triggers handling data as part of the integration should be listed.>

Triggers

Trigger Name	Table	Column	Event	Description

4.1.5. Stored procedures

<This section describes the stored procedures in the database that are used in the solution. If databases are part of an installed application, only those stored procedures handling data as part of the integration should be listed.>

Stored Procedures

Store Procedure Name	Arguments	Description

4.1.6. Security

<This section should describe any impacts to security or vulnerabilities that may exist. It should list any roles defined in the database and their permissions.>

4.1.7. Performance

<This section describes any impacts on performance that are anticipated. It includes a description of either design characteristics intended to improve, or minimise the impact on performance.>

4.1.8. Capacity

<This section describes any impacts on capacity. It includes a description of either design characteristics intended to improve, or minimise the impact on capacity.>

4.1.9. Data access

Describe the method, approach, data abstraction, object hierarchy, or class structure that will be developed/used for programmatic data access. If the details of this are in a detailed design document refer the reader to the appropriate document.>

4.1.10. Error handling

<This section needs to describe any error handling built into the component. If this component generates any error messages they should be listed with a description of their likely cause and a potential remedy if appropriate. The error message specification will be needed by testers as ll as during the development of operational/user documentation.>

4.1.11. Installation and deployment strategy

<This section describes how this database needs to be installed including required software and configuration or setting of environment variables, etc. This section should be included only for new databases developed for

the solution. Installation instructions for databases that are part of an already embedded application do not need to be included in this section unless the DB has been modified.>

4.1.12. Data initialisation

<This section describes how the data will be initially populated. For example, for new security tables user IDs need to be added to roles and, if a column was added to an existing table it will need to be populated. Many projects involve migration of data, in which cases the sections that follow come into play. If there is a migration planned this section should describe the strategy for doing so as an introduction to the sections that follow.>

4.1.12.1. Cleansing

<Describe any data cleansing that is planned or if none is planned say so>

4.1.12.2. Conversion

<Describe any data conversion that will be necessary to migrate the data – may include data types, mapping to completely new values, or adjusting other characteristics (i.e. length, etc.). Describe the tools to be used, processes, etc. If there is no conversion planned state there is none.>

4.1.12.3. Migration

<Describe the data migration that will be necessary – if data will be manually entered or typed in prior to going into production describe that as ll. Describe the tools to be used, processes, etc.>

4.1.13. Data management

<Describe here what data management activities are recommended – backup, defrags, log maintenance, support of disaster recovery, etc.>

4.1.14. Documentation and training impacts

<This section should describe what should be included in the documentation and training as a result of this design.>

4.1.15. Unit test cases

<In many cases this section may be abbreviated or not applicable as the unit test plan for the DB is performed in conjunction with other functional parts of the solution. Hover any tests that will be performed to verify that the data store has been accurately configured and functions as designed should be described here.>

<If there are a large number of cases or if this table is to be reused to track test results, consider putting it in Excel or other tool that will allow it to be more productively managed/reused (i.e. sorted, filtered, etc.). See the Testing work stream for applicable templates.>

Unit Test Cases

Test case Number	Feature	Test Description	Inputs	Expected Results	Additional Notes
Use the projects numbering system	Short name for feature or component	Describe the test that will done	Data that will be input	Data that will result	Any additional criteria – response time etc.

4.2. Component-2…

5. Database Dictionary Example

Workflow Master.xlsx