The database development life cycle

Introduction

Relational database systems underpin the majority of the managed data storage in computer systems. This unit presents an overview of the development life cycle for a database system and highlights how the database development differs from traditional software development.

This unit is an adapted extract from the Open University course *Relational databases: theory and practice* (M359).

1 The database development life cycle

1.1 Introduction

In this unit we look at an outline of the stages involved in the development of a database. We consider the broader issue of how to decide what should be in a database and how to structure the tables that should be included. Our aim is to give you a basic development method so that you can see how a basic database system is developed. We don't argue that this specific method is the most applicable to any given situation – however, we do consider that this method is straightforward and will allow you to contextualize or, by comparison, consider a range of database development techniques.

Before we consider the development method in more detail let's discuss why we need to take a formal approach to database development. After all, it is quite simple to use structured query language (SQL) **CREATE TABLE** statements to define tables, or to use the facilities of a database tool to define them for you. Once developed, the tables can be manipulated and displayed in many different ways, again using SQL statements, a database tool or an application development tool. However, uncontrolled ad hoc creation of tables by end users leads to an unmanageable and unusable database environment, and can result in the inclusion of multiple copies of potentially inconsistent data. In effect, this can create islands of data within which the end users cannot find the data that they require.

SQL is a special kind of computer language used for relational databases. These initials originated from 'structured query language'. Although this phrase is no longer used the initials SQL still are. SQL is an essential part of the practical understanding of relational databases, but we are only concerned that you appreciate its role in defining and accessing a database.

To recognize why methodical development is an issue, let's look at a very simple example. A hotel provides its clients with accommodation, food and drink and wants to record what each client spends for each cost category so that, as they leave, each client is presented with an itemized bill for all they have spent.

The problem is that there is not just one way in which we can choose tables for this purpose. We can suggest three alternative ways of satisfying the basic requirement of being able to record the data that the hotel has specified. Occurrences of data for

two example clients (arbitrarily identified by a code) for each method are shown in Figures 1, 2 and 3. The billing data for both the clients are the same in all three figures, but represented differently.

Bill

Client	Accommodation	Food	Drink
C35	162	75	
C41		38	7

Figure 1 First hotel example table

In Figure 1 there is one table, Bill, which has a row for each client and a column for each cost category. When a client does not spend any money for a category, there is simply no entry.

Accommodation

Client	Cost
C35	162

Food

Client	Cost
C35	75
C41	38

Drink

Client	Cost
C41	7

Figure 2 Second hotel example tables

In Figure 2 there are three tables, Accommodation, Food and Drink, corresponding to each of the three cost categories, and each table has a row for a client only if they have spent money in that billing category. Notice that with this new design the empty entries for unused billable items are not necessary. A record is created only when needed.

Cost

Client	Category	Cost
C35	Accommodation	162
C35	Food	75
C41	Food	38
C41	Drink	7

Figure 3 Third hotel example table

In Figure 3 there is one table, Cost, which has a row for each item of cost, with an associated column describing the category of that cost.

We are not going to say which is the best option to choose, mainly because this decision really involves a lot more understanding of the user's requirements than we have presented here. In particular, it is important to know whether the data may be used for some other purpose (such as monitoring regular clients) and whether

there may be a requirement to include additional data at some time (such as the cost of telephone calls).

Exercise 1

For the three options given for the hotel example, describe how each one would allow for the inclusion of data about the cost of telephone calls.

Answer

To include data about the cost of telephone calls, the first method will need a new column added to the table Bill. The second method of recording the data will need a new table, which could be named Telephone, with the same columns as the other tables for this method. The third method does not need any change: rows can just be added for this data with an appropriate category for the cost.

You can see from Solution 1 that any additional data can have a different impact on each alternative data-recording technique. Databases may be expected to change, so you need to appreciate that making the right choice of tables is important for the long-term success of the database implementation.

The message from this simple example then is that a relational database is not just a collection of tables created at the whim of a user but should be seen as a coordinated set of tables designed to satisfy some specified requirements. What is needed is a way of developing and designing a database to allow the requirements to be identified clearly and taken into account.

1.1.1 Desirable properties of a database

As you will see, there are many possible choices that can be made during the design and many rules to guide this work. When trying to decide if some choices are better than others, you need to consider the key desirable properties of a database. The table here outlines some of them:

Completeness	Ensures that users can access the data they want. Note that this includes **ad hoc queries**, which would not be explicitly given as part of a statement of data requirements.
Integrity	Ensures that data is both consistent (no contradictory data) and correct (no invalid data), and ensures that users trust the database.
Flexibility	Ensures that a database can evolve (without requiring excessive effort) to satisfy changing user requirements.
Efficiency	Ensures that users do not have unduly long response times when accessing data.
Usability (ease of use)	Ensures that data can be accessed and manipulated in ways which match user requirements.

Developing a ‘good’ database with these desirable properties isn't easy. Indeed, it is quite possible to develop a database that appears to contain all the relevant data but does not have these properties: as you will see, this has unfortunate consequences.

Exercise 2

Using the given desirable database properties above, very briefly summarize the characteristics that may be expected with a ‘bad’ database.

Answer

A bad database would: not satisfy all user requirements; would contain inconsistent and invalid data; would require excessive effort to change; would be slow and clumsy to use for achieving a desired outcome.

The importance of good database development is found simply in terms of preventing the problems outlined in Solution 2. Database development is not just a matter of creating tables that seem to match the way you see data on forms or reports, but requires a detailed understanding of the meaning of the data and their relationships to ensure that a database has the right properties. This understanding comes from data analysis, which is concerned with representing the meaning of data as a **conceptual data model**. Getting a conceptual data model right is crucial to the development of a good database.

1.2 A development life cycle

Database development is just one part of the much wider field of **software engineering**, the process of developing and maintaining software. A core aspect of software engineering is the subdivision of the development process into a series of phases, or steps, each of which focuses on one aspect of the development. The collection of these steps is sometimes referred to as a **development life cycle**. The software product moves through this life cycle (sometimes repeatedly as it is refined or redeveloped) until it is finally retired from use. Ideally, each phase in the life cycle can be checked for correctness before moving on to the next phase. However, software engineering is a very rich discipline with many different methods for the subdivision of the development process and a detailed exploration of the many different ways in which development can be structured is beyond the scope of this unit.

Here, we start with an overview of the **waterfall model** such as you will find in most software engineering text books. (Do note that in this unit we aim to present database development principles and techniques that are common to many development methods, not just the waterfall model.) Figure 4 illustrates a general waterfall model which could apply to any computer system development. It shows the process as a strict sequence of steps where the output of one step is the input to the next and all of one step has to be completed before moving onto the next.

However, in reality there is usually some degree of refinement and feedback as the product proceeds through the development stages (it would be rare to find that each task is performed perfectly and never needs revisiting – although that is one possibility!).

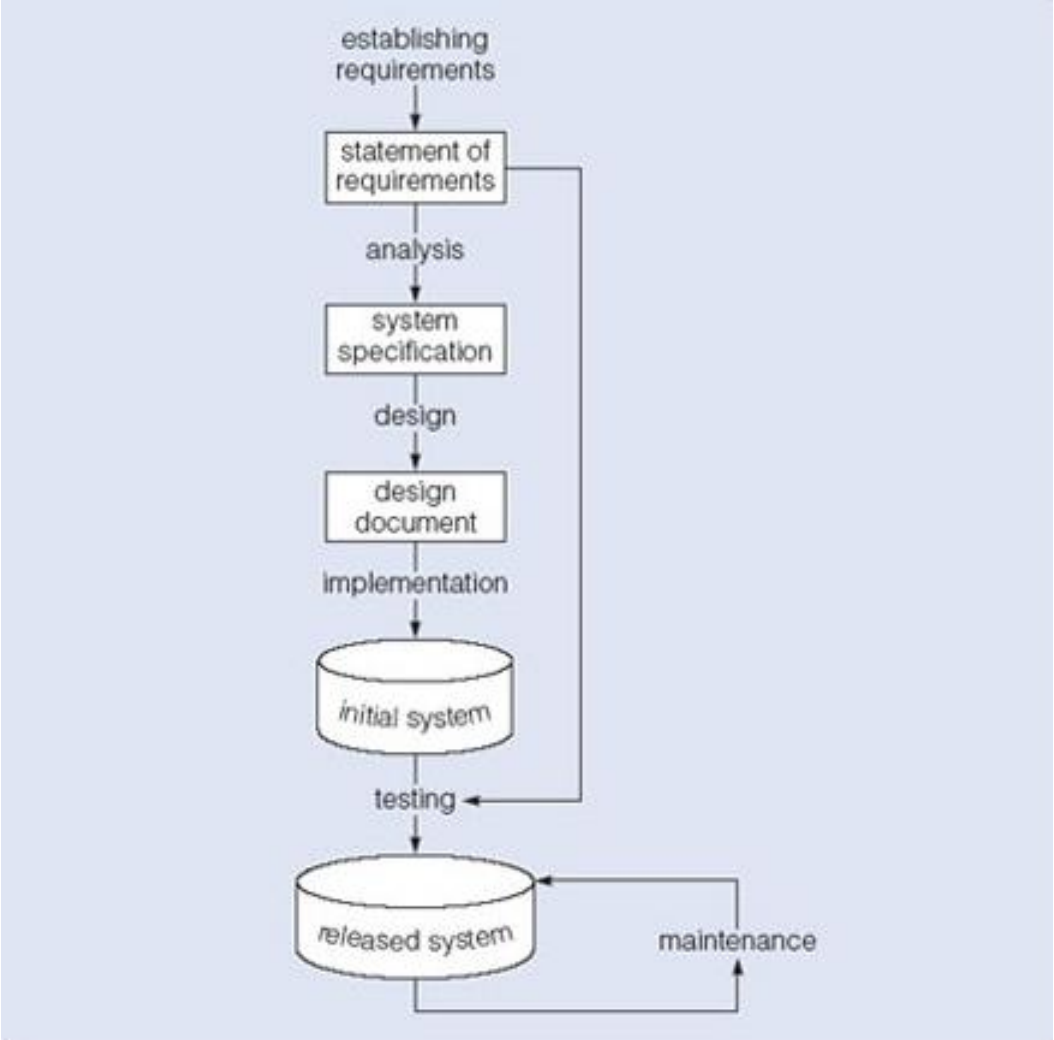

Figure 4 A general model of system development: the waterfall model

We can use Figure 4 as a means of identifying the tasks that are required, together with the input and output for each activity. What is important is the scope of the activities, which can be summarized as follows:

- **Establishing requirements** involves consultation with, and agreement among **stakeholders** as to what they want of a system, expressed as a statement of requirements.
- **Analysis** starts by considering the statement of requirements and finishes by producing a system specification. The specification is a formal representation of what a system should do, expressed in terms that are independent of how it may be realized.
- **Design** begins with a system specification and produces design documents, and provides a detailed description of how a system should be constructed.

- **Implementation** is the construction of a computer system according to a given design document and taking account of the environment in which the system will be operating (for example specific hardware or software available for the development). Implementation may be staged, usually with an initial system than can be validated and tested before a final system is released for use.
- **Testing** compares the implemented system against the design documents and requirements specification and produces an acceptance report or, more usually, a list of errors and bugs that require a review of the analysis, design and implementation processes to correct (testing is usually the task that leads to the waterfall model iterating through the life cycle).
- **Maintenance** involves dealing with changes in the requirements, or the implementation environment, bug fixing or porting of the system to new environments (for example migrating a system from a standalone PC to a UNIX workstation or a networked environment). Since maintenance involves the analysis of the changes required, design of a solution, implementation and testing of that solution over the lifetime of a maintained software system, the waterfall life cycle will be repeatedly revisited.

Exercise 3

The following are problems that have been identified during the testing process in the development of a new system. In which part of the life cycle do you think these problem could have originated and been identified by a thorough review following that stage in the development life cycle?

1. The performance of the system is poor – failing to respond quickly enough to meet the stated user requirement of interactive, screen-based use.
2. No backup facilities were included to meet the users' requirement of long-term archival of their data.
3. No user manuals were provided!

Answer

1. The user requirement for interactive, screen-based use forms part of the system specification, but actual performance can only be identified at implementation time, when the system has been built and is being evaluated. In the development of some systems, attempts at performance prediction can occur during the design stage; however, with database development it is normal to validate such design predictions at the testing stage.
2. The missing feature was in the requirements, so during analysis, design or implementation someone has overlooked this requirement. Without further information it is impossible to say when the feature fell out of the development life cycle.
3. No user manuals – were they asked for as part of the requirements? If they were then see the answer to (2). If they weren't part of the requirements, then such a basic omission should have been identified at the requirements gathering – user documentation should be considered as a standard requirement for any new system.

There are many issues outstanding from this brief description, but we do not want to elaborate on the general model any further, simply because it is general and does not address the development of a database system explicitly. However, we can use this framework as the basis for a model of database development which incorporates three assumptions:

- We can separate the development of a database – that is, specification and creation of a schema to define data in a database – from the user processes that make use of the database.
- We can use the three-schema architecture as a basis for distinguishing the activities associated with a schema.
- We can represent the constraints to enforce the semantics of the data once, within a database, rather than within every user process that uses the data.

Using these assumptions, Figure 5 represents a model of the activities and their outputs for database development. It is applicable to any class of **DBMS** (database management system), not just a relational approach.

In the rest of this unit we will look at each of these tasks in more detail.

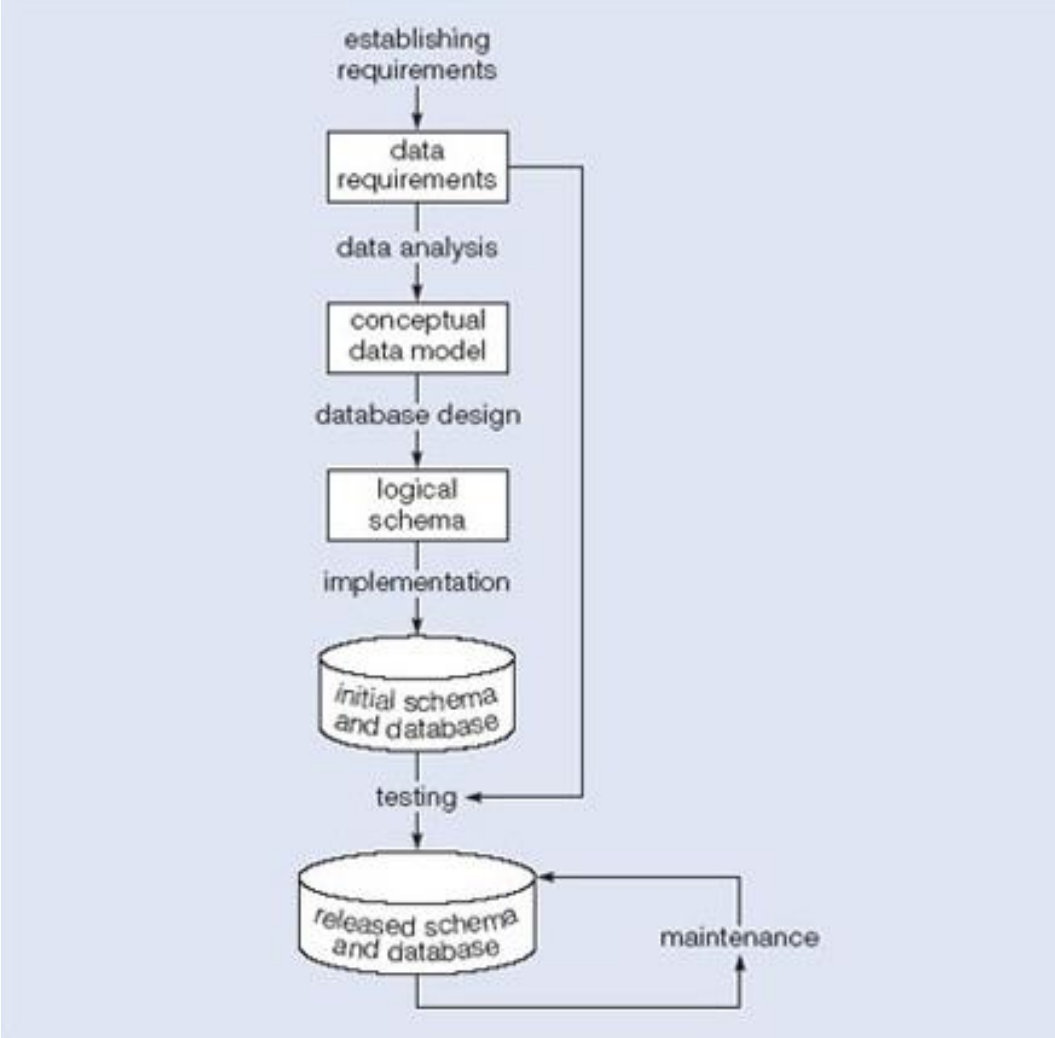

Figure 5 Model of database development

1.3 Requirements gathering

Here we are concerned only with the requirements that relate specifically to the data. Establishing requirements involves consultation with, and agreement among, all the users as to what persistent data they want to store along with an agreement as to the meaning and interpretation of the data elements. The data administrator plays a key role in this process as they overview the business, legal and ethical issues within the organization that impact on the data requirements.

The data requirements document is used to agree requirements with users. To make sure that it is easily understood, it should not be overly formal or highly encoded. The document should give a concise summary of all users' requirements – not just a collection of individuals' requirements – as the intention is to develop a single shared database.

Below is an example of a statement from an Open University summary database card:

> Each course which is available for study is given a course code, a title and a value for credit points – either a 30-point course or a 60-point course. A course may have a quota – the maximum number of students that can be enrolled on the course in any one year; each quota is reviewed regularly and the last date of review is recorded with the quota. A course need not (yet) have any students enrolled on it. Students may not enroll for more than 180 points' worth of courses at any one time.

You should note that the above says nothing about how the data are processed, but it does state what the data items are, what attributes they have, what constraints apply and the relationships that hold between the data items.

1.4 Analysis

Data analysis begins with the statement of data requirements and then produces a **conceptual data model**. The aim of analysis is to obtain a detailed description of the data that will suit user requirements so that both high and low level properties of data and their use are dealt with. These include properties such as the possible range of values that can be permitted for attributes such as, in the Open University example for instance, the course code, course title and credit points.

The conceptual data model provides a shared, formal representation of what is being communicated between clients and developers during database development – it is focused on the data in a database, irrespective of the eventual use of that data in user processes or implementation of the data in specific computer environments. Therefore, a conceptual data model is concerned with the meaning and structure of data, but not with the details affecting how they are implemented.

The conceptual data model then is a formal representation of what data a database should contain and the constraints the data must satisfy. This should be expressed in terms that are independent of how the model may be implemented. As a result, analysis focuses on 'What is required?' not 'How is it achieved?' Data analysis is a highly skilled task and the analyst has a specialized role that is beyond the scope of this unit where our focus is on design. Consequently, we will not consider the analysis task in detail but we will assume that the conceptual data model is the starting point for our database development.

Analyze the data requirements, not the implementation
One of the hardest issues facing an analyst is to perform the analysis without prejudging decisions about implementation. The analysis is purely focused on the data requirements and not about how those requirements are to be met, or the limitations that might be enforced by the DBMS chosen to host the database. Compromises and enforced limitations resulting from a particular DBMS or computer system should be dealt with during the implementation phase. The requirements gathering and analysis tasks should be performed as if the implementation environment will do everything that needs to be done to satisfy the requirements being specified. Any compromises made at the analysis stage will

affect the usefulness of the database and may lead to it failing to meet the user requirements.

You may think of a conceptual data model as being a formal description of the eventual database semantics used to produce a logical schema for a database. Everything in the conceptual data model will appear in the logical schema and everything in the logical schema will be in the conceptual data model. However, a conceptual data model is **not** necessarily expressed in terms of relations or tables, because it will not necessarily depend on the use of a relational DBMS for implementation. You should also note that a conceptual data model is a specification used by people for the database design activity; it is not used by any DBMS, nor is it a programming language.

Exercise 4

For each of the following statements decide which processes – requirements gathering or data analysis – would generate the statement as part of the documented output.

1. A customer record will allow for the storage of a name, UK address, evening and daytime phone numbers, one mobile phone number and as many email addresses as the customer wants to include.
2. We need to relate customer orders to their credit card details. If the credit card is invalid we need to know before any orders are accepted.
3. An order must have the opportunity to include a delivery address that is different from the customer's credit card billing address.

Answer

1. This is quite a detailed description of what data will be recorded about a customer so it is likely to be data analysis output – it might be included in the requirements analysis (a lot of information about data items usually is), but it will form part of the more formal conceptual data model.
2. This statement is about the requirements for recording a valid order, so it is part of the requirements specification. In effect, it is saying that an order can be recorded only if the credit card details are valid. If this appeared in the data analysis output we would need to include answers to several more questions: What is a valid credit card? What is an order? Can a customer use more than one credit card per order, or none? And so on.
3. This is another requirements specification statement – it says nothing about the data requirements in any detail. As a result, a lot more questions would need to be asked about the statement and what it references before the data analyst would be able to document this statement.

1.5 Design

Database design starts with a conceptual data model and produces a specification of a logical schema; this will usually determine the specific type of database system (network, relational, object-oriented) that is required, but not the detailed implementation of that design (which will depend on the operating environment for the database such as the specific DBMS available). The relational representation is still independent of any specific DBMS; it is another conceptual data model.

Our approach here is to use a relational database environment. We can use a relational representation of the conceptual data model as input to the design process. The output of the design stage is a detailed relational specification, the logical schema, or Entity Relationship Diagram (ERD) of all the tables and constraints needed to satisfy the description of the data in the conceptual data model. It is during the design activity that choices are made as to which tables are most appropriate for representing the data in a database, such as for the sample hotel example in section 1. These choices must take into account various design criteria including, for example, flexibility for change, control of duplication and how best to represent the constraints. It is the tables defined by the logical schema that determine what data are stored and how they may be manipulated in the database.

Database designers familiar with relational databases and SQL might be tempted to go directly to implementation after they have produced a conceptual data model. However, such a direct transformation of the relational representation to SQL tables does not necessarily result in a database that has all the desirable properties: completeness, integrity, flexibility, efficiency and usability. A good conceptual data model is an essential first step *towards* a database with these properties, but that does not mean that the direct transformation to SQL tables automatically produces a good database. This first step (sometimes called a **first-cut design**) will accurately represent the tables and constraints needed to satisfy the conceptual data model description, and so satisfies the completeness and integrity requirements, but it may be inflexible or offer poor usability. The first-cut design is then **flexed** to improve the quality of the database design. Flexing is a term that is intended to capture the simultaneous ideas of bending something for a different purpose and weakening aspects of it as it is bent.

Using relational theory for a formal design
There will be occasions when it is necessary to prove formally that a database satisfies given requirements. Using relational theory can allow a relational representation of a conceptual data model to be analyzed rigorously. This stage, which is usually omitted in all but the most exacting development environments (such as safety-critical systems), involves using the formal properties of the relational theory to mathematically prove properties of the conceptual data model that would then be realized in the database design.

Figure 6 summarizes the iterative (repeated) steps involved in database design, based on the overview given. Its main purpose is to distinguish the general issue of

what tables should be used from the detailed definition of the constituent parts of each table – these tables are considered one at a time, although they are not independent of each other. Each iteration that involves a revision of the tables would lead to a new design; collectively they are usually referred to as **second-cut designs**, even if the process iterates for more than a single loop.

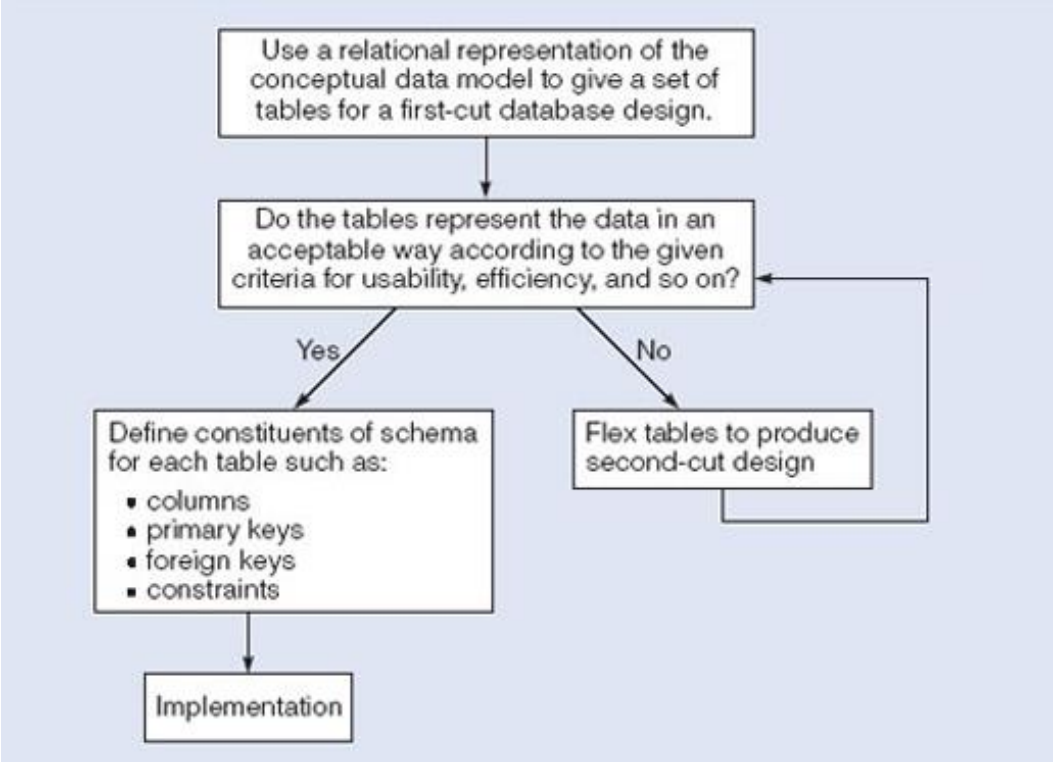

Figure 6 Summary of steps for database design

Before we turn to consider implementation, you should note three general points that form the basis of our design approach.

First, for a given conceptual data model it is not necessary that all the user requirements it represents have to be satisfied by a single database. There can be various reasons for the development of more than one database, such as the need for independent operation in different locations or departmental control over 'their' data. However, if the collection of databases contains duplicated data and users need to access data in more than one database, then there are of course further issues related to data replication and distribution.

second, remember that one of the assumptions about our database development is that we can separate the development of a database from the development of user processes that make use of it. This is based on the expectation that, once a database has been implemented, all data required by currently identified user processes have been defined and can be accessed; but we also require flexibility to allow us to meet future requirements changes. In developing a database for some applications it may be possible to predict the common requests that will be presented to the database and so we can optimize our design for the most common requests.

Third, at a detailed level, many aspects of database design and implementation depend on the particular DBMS being used. If the choice of DBMS is fixed or made prior to the design task, that choice can be used to determine design criteria rather than waiting until implementation. That is, it is possible to incorporate design decisions for a specific DBMS rather than produce a generic design and then tailor it to the DBMS during implementation.

It is not uncommon to find that a single design cannot simultaneously satisfy all the properties of a good database. So it is important that the designer has prioritized these properties (usually using information from the requirements specification), for example, to decide if integrity is more important than efficiency and whether usability is more important than flexibility in a given development.

At the end of our design stage the logical schema will be specified by SQL data definition language (DDL) statements, which describe the database that needs to be implemented to meet the user requirements.

1.6 Implementation

Implementation involves the construction of a database according to the specification of a logical schema. This will include the specification of an appropriate storage schema, security enforcement, external schema, and so on. Implementation is heavily influenced by the choice of available DBMS, database tools and operating environment. There are additional tasks beyond simply creating a database schema and implementing the constraints – data must be entered into the tables, issues relating to the users and user processes need to be addressed and the management activities associated with wider aspects of corporate data management need to be supported. In keeping with the DBMS approach we want as many of these concerns as possible to be addressed within the DBMS. We look at some of these concerns briefly now.

In practice, implementation of the logical schema in a given DBMS requires a very detailed knowledge of the specific features and facilities that the DBMS has to offer. In an ideal world, and in keeping with good software engineering practice, the first stage of implementation would involve matching the design requirements with the best available implementing tools and then using those tools for the implementation. In database terms, this might involve choosing vendor products whose DBMS and SQL variants are most suited to the database we need to implement. However, we don't live in an ideal world and more often that not, hardware choice and decisions regarding the DBMS will have been made well in advance of consideration of the database design. Consequently, implementation can involve additional flexing of the design to overcome any software or hardware limitations.

Exercise 5

Summarize the reason why we want as much of the data management as possible to be addressed within the DBMS.

Answer

In a typical database system, different users and user processes share data – the alternative to putting the data management in the DBMS with the data is to duplicate the data management requirements in every application that uses the data. The more we can put in the DBMS, the less we have to implement in our user processes.

1.6.1 Realizing the design

So far we have been concerned only with the specification of a logical schema. We now need our database to be created according to the definitions we have produced. For an implementation with a relational DBMS, this will involve the use of SQL to create tables and constraints that satisfy the logical schema description and the choice of appropriate storage schema (if the DBMS permits that level of control).

One way to achieve this is to write the appropriate SQL DDL statements into a file that can be executed by a DBMS so that there is an independent record, a text file, of the SQL statements defining the database. Another method is to work interactively using a database tool like Sybase Central (or Microsoft Access), where the forms provided for defining tables help avoid the need to remember the syntactic detail of the SQL language. While this may seem to make it easier to realize a database, it can lead to maintenance problems. In this case, there can be a problem keeping track of exactly how tables are defined and the ability to make changes to them, so it is not recommended for large development projects.

Whatever mechanism is used to implement the logical schema, the result is that a database, with tables and constraints, is defined but will contain no data for the user processes.

1.6.2 Populating the database

After a database has been created, there are two ways of populating the tables – either from existing data, or through the use of the user applications developed for the database.

For some tables, there may be existing data from another database or data files. For example, in establishing a database for a hospital you would expect that there are already some records of all the staff that have to be included in the database. Data might also be bought in from an outside agency (address lists are frequently bought in from external companies) or produced during a large data entry task (converting hard-copy manual records into computer files can be done by a data entry agency). In such situations the simplest approach to populate the database is to use the import and export facilities found in the DBMS. Facilities to import and export data in various standard formats are usually available (these functions are also known in some systems as loading and unloading data). Importing enables a file of data to be copied directly into a table. When data are held in a file format that is not

appropriate for using the import function then it is necessary to prepare an application program that reads in the old data, transforms them as necessary and then inserts them into the database using SQL code specifically produced for that purpose. The transfer of large quantities of existing data into a database is referred to as a **bulk load**. Bulk loading of data may involve very large quantities of data being loaded, one table at a time so you may find that there are DBMS facilities to postpone constraint checking until the end of the bulk loading.

Exercise 6

Why would you postpone constraint checking during bulk loading operations?

Answer

Some constraints will involve comparisons between data stored in several tables. Since the importing of data is usually performed one table at a time, these constraints will fail and the import would be prevented. Only when all the bulk data have been loaded should the constraints be checked because, at that time, the data in the database should be consistent.

Those tables for which there are no existing data will be populated using the application processes supporting the normal activities of the organization, such as enrolling new students on courses or assigning new patients to wards in a hospital.

1.6.3 Supporting users and user processes

Use of a database involves user processes (either application programs or database tools) which must be developed outside of the database development. In terms of the three-schema architecture we now need to address the development of the external schema. This will define the data accessible to each user process or group of user processes. In reality, most DBMSs, and SQL itself, do not have many facilities to support the explicit definition of the external schema. However, by using built-in queries and procedures, and with appropriate security management, it is possible to ensure access to data by a user process is limited to a tailored subset of the entire database content.

In addition to ensuring that appropriate data access for each user process is achieved, the database developer needs to address other user-related issues. Examples of such issues include: reporting of error conditions, constraint enforcement, automated processing using triggers, grouping of activities into transactions, defining database procedures and functions and data security (in addition to the general database and user process access control).

1.6.4 Supporting data management strategies

Most of the development we've covered so far in this unit has focused on meeting specific user requirements – that is, ensuring the right data are constrained correctly

and made available to the right user processes. However, other questions must also be addressed in order to support a data management strategy: How frequently should data be backed-up? What auditing mechanisms are required? Which users will be permitted to perform which functions? Which database tools and user processes will be available to users to access data? What level of legal reporting is required? And so on. The data administrator will be involved in setting policy, but the implementer needs to ensure that the right data are being accessed in the right ways by the right users, with appropriate security, record keeping and reporting taking place.

Efficiency: the interaction between design and implementation

When using the three-schema architecture we would like to separate the logical schema, that is, the description of the tables in the database, from the storage schema required for its efficient implementation. This separation represents an ideal that is rarely found in a commercial DBMS. This is most evident when we need to take account of efficiency. When DBMSs lack the ability to separate these concerns it forces efficiency issues to be considered during the database design (by choosing efficient table structures) rather than leaving such decisions until the implementation stage. An initial design for a logical schema may be produced, but its efficiency can only be evaluated fully during implementation. If the resulting implemented database is not efficient enough to meet the processing requirements, it is necessary to return to the database design and consider how the logical schema may be changed to be more efficient. If separation of logical and storage schema is possible, and if another storage schema can efficiently implement the logical design, then the logical design may not need revision.

For this reason, some database design methods refer to two separate design stages: *logical* database design and *physical* database design. However, this physical stage is mainly concerned with producing a revised logical schema – that is, specifying SQL tables – so this terminology can be confusing. The tables of a logical schema may differ from the relational representation of the conceptual data model because of concerns with efficiency. To make design decisions that address specific efficiency concerns requires detailed knowledge of the specific DBMS facilities and hardware systems that will host the database.

1.7 Testing

The aim of testing is to uncover errors in the design and implementation of the database, its structure, constraints and associated user and management support. Testing is usually considered to involve two main tasks – validation and verification. Without adequate testing users will have little confidence in their data processing.

Validation answers the question: has the right database been developed to meet the requirements? It attempts to confirm that the right database has been constructed, with the right characteristics to meet the specified requirements.

Verification answers the question: has the database design been implemented correctly? Verification ensures that the processing steps, constraints and other 'programmed' components of the database (security, backup, recovery, audit trails, etc.) have been correctly implemented and contain no errors in program logic or execution sequences.

Of course, testing does not just take place only after all the above development steps are complete. It is usually applied throughout the stages in the development processes and includes appropriate reviews to scrutinize the outputs of the development activities. The aim is to identify errors as soon as possible in the development life cycle.

1.8 Maintenance

Databases are one of the more enduring **software engineering artefacts**; it is not uncommon to find database implementations whose use can be traced back for 15 years or more. Consequently, maintenance of the database is a key issue.

Maintenance can take three main forms:

- **Operational maintenance**, where the performance of the database is monitored. If it falls below some acceptable standard, then reorganization of the database, usually in the form of adjustments to the storage schema, takes place to ensure that performance is maintained at an acceptable level.
- **Porting** and implementation maintenance, in which the DBMS, the user processes, the underlying computer system or some other aspect undergoes changes that require the database implementation to be revised. Of course, the extreme form of porting occurs when the database is to be entirely replaced – in which case the entire development life cycle is usually followed using the existing database and its limitations as the starting point for requirements analysis. Adjustments to the storage schema are almost inevitable as new data storage capabilities become available. This can involve both restructuring and reorganization, depending on the scope of the changes taking place.
- **Requirements change**, where the original requirement specification changes, usually because databases are frequently used for purposes for which they were not originally designed. This involves restructuring and typically involves a 'mini life cycle' related to the development of changes to meet the new requirements.

If we follow the three-schema architecture approach we would hope to minimize the impact of change and build systems which are easily maintained.