University of Bristol Information Services

Designing your database

Practical workbook

Aims and Learning Objectives

By the end of this document, you will be able to:

- explain the three types of logical relationship that can be identified in database design and point out the only relationship that should be used in logical design;
- explain how to resolve many-to-many relationships;
- explain where primary and foreign keys must be located in any logical design;
- capture data requirements from paper forms or spreadsheets;
- draw simple data structure models with correctly-placed primary and foreign keys.

Document information

Course files

This document and any associated practice files (if needed) are available on the web. To find these, go to www.bristol.ac.uk/it-services/learning/resources and in the Keyword box, type the document code given in brackets at the top of this page.

Related documentation

Other related documents are available from the web at:

http://www.bristol.ac.uk/it-services/learning/resources

Introduction

This document provides a framework for planning and designing a simple database. Once you have taken on board the few simple rules introduced here, you will find that these rules apply to the logical design of any relational database, whatever its size. However, you will have to apply and practice your new skills.

Why do I need to make plans before I start to build my database?

Too many databases fail to do what was planned because simple questions were not asked and simple tasks were not done beforehand. Poor planning results in time-wasting throughout development and often leads to an unusable database. Yes, parts might work, but it could take longer to do the job using the database than it did to do it manually - and the database might provide less accurate results to boot.

Prerequisites

A clear understanding of what you want to get out of your database once it is completed. If you know what outputs you want, then you are more than half way toward knowing what inputs you need: and if you know what inputs you need, then you should have little trouble working out what tables your database needs. At which point, this document will fill in the gaps to help you design your database correctly.

Contents

Task 1 Understanding relationships

Objectives To understand the three main types of relationship that can be identified when designing database tables; to know how to resolve a many-to-many relationship.

Comments Databases are made up of tables (also called entities) and each of these tables is related to one or more of the other tables in the database.

Database tables

Before trying to design your database, you should know how to go about designing the database on paper first. The most common model is called a logical data structure (LDS) and is also known as an entity-relationship model (ER) because it shows the entities (tables) and the relationships between them.

In a database, each table should hold data about one specific item. For example, a Person table would hold data about people but would not hold data about jobs, because data about jobs would be held in a Job table. The two tables would then be linked by a relationship so that you can tell which person has which job (or which people have which jobs).

Once we have worked out what tables the database needs to hold we need to work out what the relationships between these tables are.

Types of relationship

One-to-one relationship

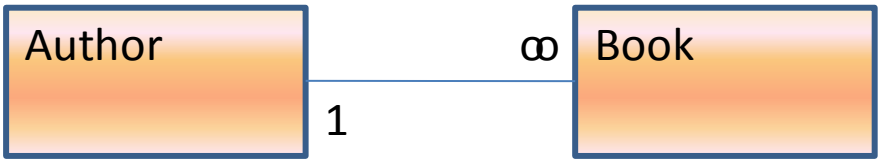
This is where there is only one occurrence of something for each individual occurrence of the related item, e.g. each person has only one National Insurance number.

One-to-one relationships, generally speaking, are not very useful and the one-to-one related tables can generally be merged into one table, e.g. in the above example, there is no need for both a Person table and a National Insurance number table – the person's National Insurance number can easily be held in the Person table.

(In the logical design of a database there is no place for one-to-one relationships. However, when it comes to building the database, there may be reasons for putting some of the data fields (i.e. those not often used) into a separate table and linking with the same primary key in both tables. But my advice is don't worry about this unless you are needing to build a massive corporate database – in which case you probably won't be reading this cheap and cheerful guide to relational design issues!)

One-to-many relationship

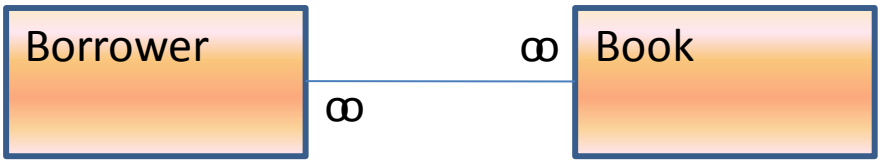
Relational databases are made up of one-to-many related tables, meaning that one occurrence of something can have more than one occurrence of the related item, e.g. each author in an Author table can have written more than one book in a Book table (we are assuming here that no books are written by more than one author – if so then this would be a many-to-many relationship – for which see the next subsection).



The infinity sign at the Book end of the relationship above denotes the **many** end of the relationship – there are other techniques but they all do pretty much the same thing.

Many-to-many relationship

For example, a single book can be loaned to many people in a Borrower table. At the same time, each person in the Borrower table can borrow many books in the Book table.



You *cannot* have tables with many-to-many relationships in a relational database. You will identify at least one many-to-many relationship during the following tasks – don't worry, once you have identified them, they can be resolved using a standard database design method called *many-to-many table resolution*. It's easier than it sounds!

Many-to-many relationship resolution

To resolve a many-to-many relationship, you do the following:

The table added in the middle resolves the many-to-many relationship and is called a *many-to-many resolution table*, or *resolution table* for short. You do the same thing every time you identify a many-to-many relationship – it's that simple. The hardest thing about resolving many-to-many relationships is in coming up with a sensible name for the resolution table!

1.1 Think about the following entities and work out what the relationships are likely to be between them.

Entities	Relationship
Consultant and patient	
Mother and child	
Staff member and job title	
Artist and song title	
Book and ISBN number	
Patient and hospital admission	
Builder and material supplier	
Person and disease	
Student and exam	
Car and registration number	

Ah, it's not quite as easy as it appears at first, is it? Many of the answers depend on your assumptions about exactly what you intend to *do* with the data you want to hold. Even book and ISBN number could be a one-to-many if you take into account that the same book may appear on the same ISBN number but with multiple jacket designs or minor typo revisions. Even car and registration number could be a one-to-many in certain circumstances. It is basically down to the context of what you want to do with your data ... and that is your decision.

Task 2 Understanding keys

Objectives To know where primary and foreign keys should be put.

Comments Every table must have a primary key, and the related tables must have foreign keys: the joins between the tables are made with the primary and foreign keys so it is imperative that you get this right.

Primary and foreign keys

Every table must have a Primary Key: this is a field that uniquely identifies each individual record. At the University, student number is a good primary key because no two students will have the same student number. In fact, in the one-to-one relationship in the previous task, National Insurance Number would make a very good primary key for the Person table.

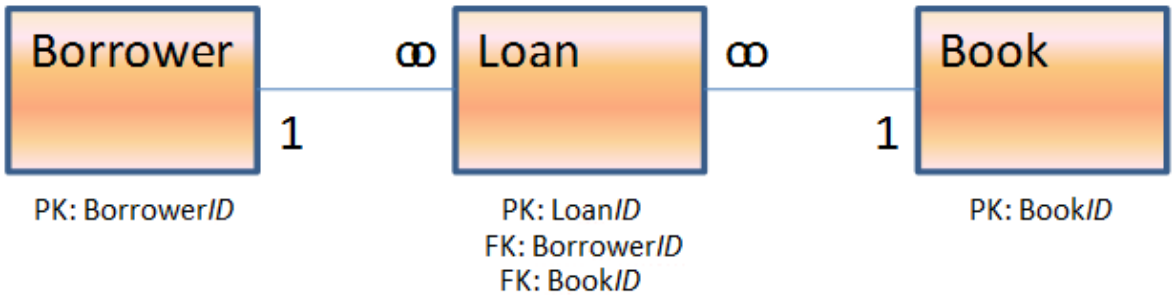
If there is no field in a table that can be used as a Primary Key, then one can be assigned arbitrarily, such as an autonumber field, which increments by one as each new record is added.

Once the relationships between tables have been worked out it is an easy task to know where Foreign Keys should go. Basically, for every one-to-many relationship you must put the Primary Key from the one end of the relationship into the table at the many end of the relationship as well, where it becomes a Foreign Key.

You do not have to understand the logic behind this, you just have to do it!

Worked example

In the example in the previous task showing Borrower, Loan and Book tables, the primary and foreign keys should be located as follows (the keys have been names as *BorrowerID*, *LoanID* and *BookID*. PK denotes primary key and FK denotes foreign key location).



Task 3 Identifying tables through using paper forms (or spreadsheets)

Objectives To identify tables through analysing repeated data on a paper form.

Comments This task provides a simple method for identifying required database tables. The same can be achieved by looking for repeating data on Excel spreadsheets.

Where repeating data appears on a paper form or in an Excel spreadsheet or similar, this indicates where you are likely to need another table.

3.1 Identify the repeating data on the following paper form and circle each column header where repeating data exists.

List of records bought during.....OCTOBER 2008.....

Format	Artist	Title	Label	Cat. no.	Condition	Cost
LP	Beatles	Abbey Road	Apple	PCS 9102	VG/EX	£2
7"	Kevin Ayers	Caribbean Moon	Harvest	HAR 5081	EX	50p
7"	Supercharge	Local Boy Chops Wood	Virgin	VS 1771	EX	50p
7"	Beatles	Hello Goodbye	Parlophone	R 6129	VG	£1
LP	Kevin Ayers	Falling Up	Virgin	V 2376	VG/EX	£3
LP	Roy Harper	H.Q	Harvest	SHSM 501	VG/EX	£2.50
CD	Heron	Best Of	Dawn	DCD 233	M/M	£3
12"	Wizzard	I Wish It Could Be Christmas	Harvest	12HAR 345	VG/EX	£1
CD	Beatles	Hey Jude	Apple	CPCS CD 3	M/M	£5
7"	Beatles	Yesterday	Parlophone	R 6745	VG/EX	£1
LP	Kevin Ayers	Sweet Deceiver	Island	ILPS 9234	VG/EX	£3.50
LP	King Crimson	Red	Island	ILPS 9243	VG/EX	£2

Note Not all of the repeating attributes will become tables.

Just because a logical entity (i.e. table) has been identified during analysis does not mean that it will become a table in your database. There is a simple rule of thumb for deciding whether an entity will be a short dropdown list instead of a table:

- Will an entity only have a few records during the life of the database?

In the above instance, Condition will only have around seven values (Mint, Excellent, Very Good, Good, Fair, Poor) and so can be easily dealt with by using a dropdown list (dropdown lists can look up other tables or they can be created 'manually' to hold just a few items). If we decide to alter the list later, this is easily done.

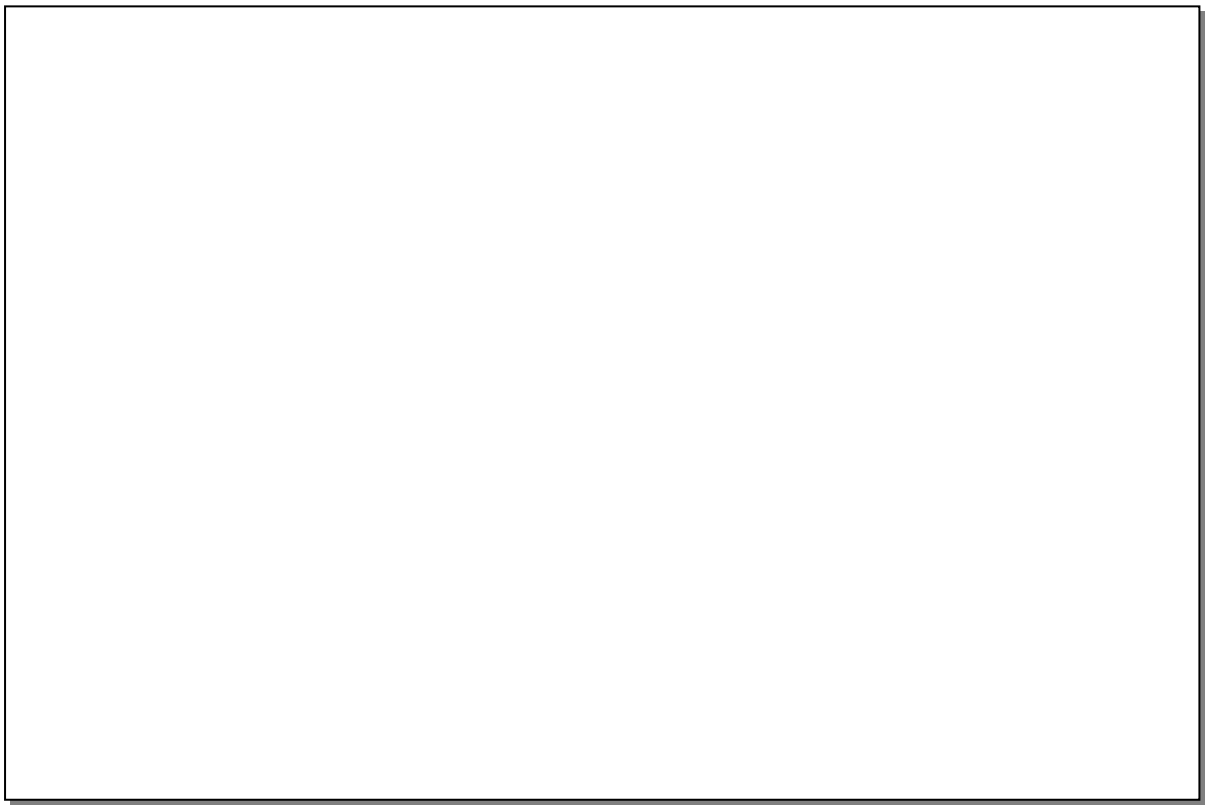
It is, however, your decision and there is nothing to stop you from creating a table for this logical entity. After all, this will make it easier to add or amend the data if necessary.

Also, you may have two records the same with the same catalogue number and price will be repeated, but these are otherwise infinitely variable in that it may just so happen that you will put the same data in every now and then. If there is the possibility of infinite variance, then this does not become a table in its own right but will be a data entry field in one or other of the identified tables.

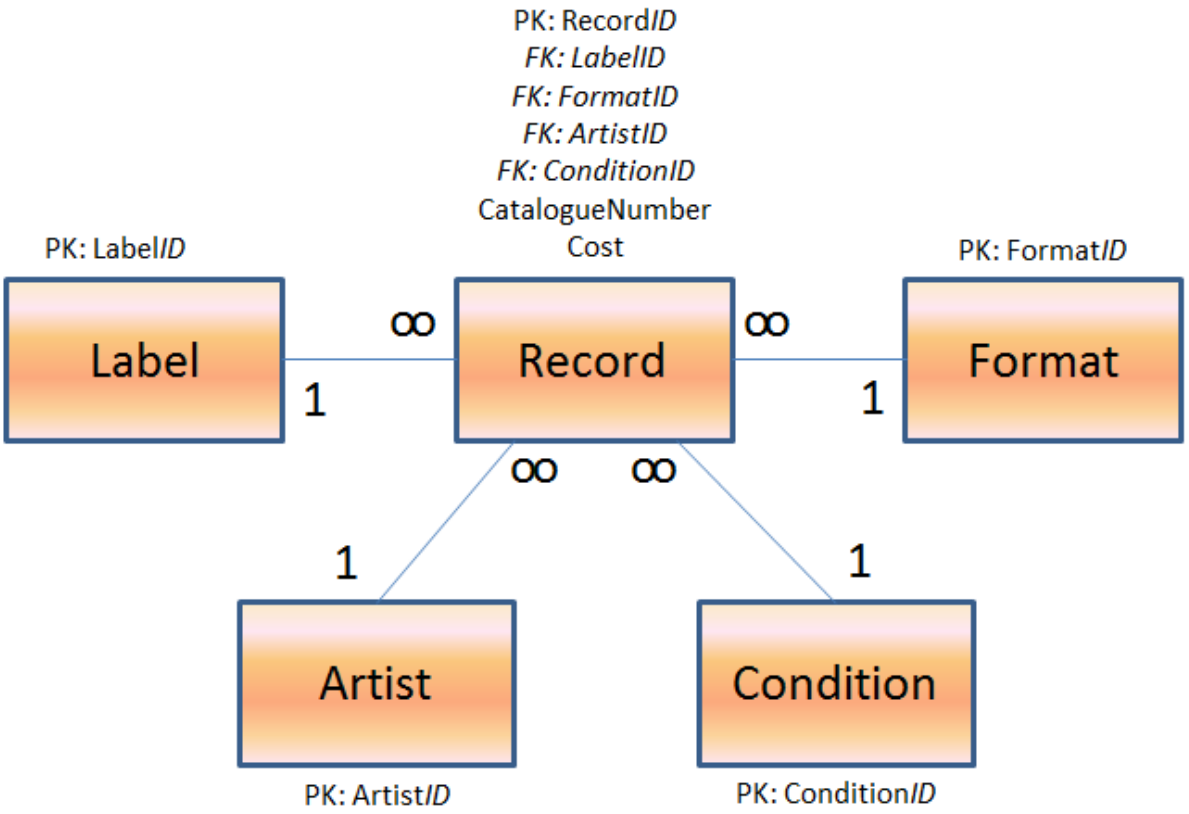
3.2 To complete the work on your database design (using the methods from Task 1):

The tables identified from 3.1 are Format, artist, label, condition. It's a list of records, so record is also a table. As stated, condition does not have to be a table, but from point of view of logical design,

- Work out the relationships between the tables you have identified.
- Which fields fit in which tables?
- Where do the Primary and Foreign Keys go?



See over the page for the correct solution. How close were you? Practice, I'm afraid, is the only answer!


PK: RecordID
FK: LabelID
FK: FormatID
FK: ArtistID
FK: ConditionID
CatalogueNumber
Cost
PK: LabelID
PK: FormatID
Label
Record
Format
∞
1
∞
1
∞
∞
1
1
Artist
Condition
PK: ArtistID
PK: ConditionID