

DB-MAIN

The Database Engineering CASE environment

March 2002

First Steps in Database design

DB-MAIN

First Steps in Database design

DB-MAIN

A product of the LIBD Laboratory

Database Application Engineering Laboratory

Institut d'Informatique

Rue Grandgagnage, 21 - B-5000 Namur • <http://www.info.fundp.ac.be/libd>

About this tutorial

This short tutorial is intended to introduce the first-time user to the basics of **database analysis**, **database design** and **SQL generation** through the DB-MAIN CASE tool. The database to build must represent facts described in an interview document, through which the manager of a small library explains how things work.

More practically, this walkthrough explains how the concepts underlying a **simple text**, that expresses **user requirements**, can be translated into a fully operational **SQL computer program** building the database structures that meet these requirements.

Most readers can finish this tutorial **in less than one hour**. The complete solution is provided in the project LIB.lun.

The use of the **DB-MAIN CASE tool**, version 4 or higher, is strongly recommended. The Education Edition of DB-MAIN, as well as other pedagogical material, can be downloaded free of charge from the **Database Engineering Laboratory** site (<http://www.info.fundp.ac.be/libd>).

This document is a plain text version of the hypertext tutorial *1st-step.hlp* distributed with the DB-MAIN tool.

Table of contents

1. Capturing the conceptual schema of the database
or *How to draw in an elegant way all the the ideas that are hidden into innocent texts.*
2. Generating a report of the conceptual schema
or *How to distribute our masterpiece among our colleagues.*
3. Producing the relational schema of the database
or *How to build data structures that support our conceptual schema.*
4. Producing the physical schema of the database
or *How to make these data structures work efficiently.*
5. Generating the SQL code that creates the database
or *How to cause our database engine to understand what we want.*

Conclusions

or *What have we learned in all these pages.*

... and now what?

The source interview report

The starting point of our project is the following interview report:

A library proposes copies of books to its borrowers. Each book has a unique number (ISBN), a title and some authors (one to five). Each copy of a book has a serial number that identifies it among all the copies of this book. It also has a location (made up of a store, a shelf and a row). A borrower is identified by his/her name and can have an address.

Figure 0.1 - The text of the interview

This text is in the file LIB.txt. This file is on the distribution CD-ROM and on the Web site. If it is not available, just type it quickly, for instance with Notepad or Wordpad.

Building a database requires four main phases:

1. Capturing the essence of the user requirements, and translating them into a *Conceptual schema*. This phase generally is called **Conceptual analysis**.
2. Translating the conceptual schema into a *Logical schema* that is made up of relational database structures (i.e., tables, columns and keys). This phase is called **Logical design**.
3. Enriching the logical schema with technical constructs such as indexes and storage spaces. This phase, called **Physical design**, produces the *Physical schema*.
4. Generating a *SQL script* that instructs the database engine to build the project database. This is the **Coding phase**.

To learn about these processes, follow the scenario of this tutorial step-by-step and apply the actions suggested in the *Action* parts.

Phase 1 - Capturing the conceptual schema of the database

Objective. To build the **conceptual schema**¹ of the future database, i.e., an abstract description of the concepts of the **application domain**² and of the information we want to record about them.

1.1 Analysis: read carefully the interview report

Objective: To find the main concepts the interview is talking about. More precisely, we must identify major classes of entities, entity properties and associations between entities.

Actions:


- Read.
- Think [by courtesy of IBM].

Result: According to the text, three major classes of entities seem to emerge, namely *books*, *copies* and *borrowers*. *Authors* are just mentioned as properties of books, and probably are not worth being considered major entities of the application domain. The text suggests that books have three specific properties: *ISBN*, *title* and *authors*. In the same way, each copy has a *serial number* and a *location*, while borrowers each have a *name* and an *address*. Finally, each copy is a *materialization of* a book and copies are *borrowed by* borrowers.

1.2 Define a new project with name LIB

Objective: To create a new document in which all the products (texts, schemas, SQL scripts, reports, etc.) of the project will be stored.

Action:

- Click on the button  in the standard tool bar or execute the command **File / New project**; give the project the name LIB; ignore the other fields of the project property box and close this box (click on the button OK).

-
1. DEFINITION. A **conceptuel schema** is a specification of the concepts of the application domain, of their properties and of the relationships that link them; it comprises entity types (or classes), attributes and relationship types (or associations)
 2. DEFINITION. The **application domain** is that part of the real world about which the database will record information. In this tutorial, the application domain comprises the library and its borrowers.

Result: A new project with name LIB is created (Figure 1.1); its project window is open; so far, it includes no schema.



Figure 1.1 - A new project, named LIB, has been created.

1.3 Import the text of the interview (LIB.txt) in the project

Objective: To introduce the external document named LIB.txt in the project; this document is the interview report from which we will build the database.

Action: • Drag & drop the file LIB.txt from the Explorer into the project window; alternatively, use the command **Product / Add file**.

Result: A first product, namely the interview text LIB.TXT, appears in the project window (Figure 1.2).



Figure 1.2 - The external text LIB.TXT has been introduced into the current project.

1.4 Open the text (LIB.txt)

Objective: To examine the contents of LIB.TXT.

Action: • Double-click on the icon of LIB.TXT.

Result: As expected (Figure 1.3) (note that the line numbers are not part of the text).

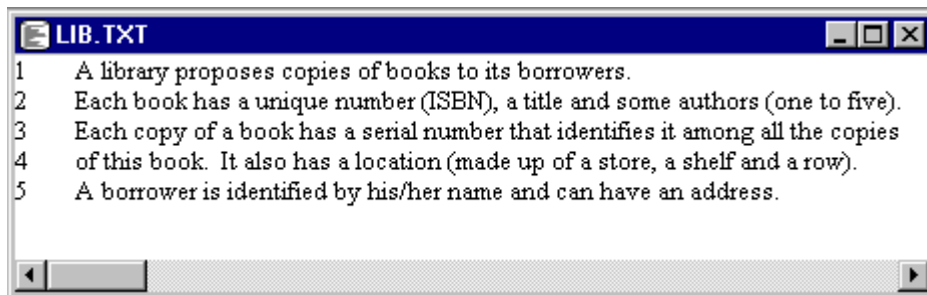


Figure 1.3 - The contents of the BIB.TXT text file.

1.5 Define a new schema (LIB/Conceptual) that depends on LIB.TXT

Objective: To create a new schema in which we will build the conceptual schema of the database.

Actions: • Command **Product / New schema**; give the schema the name LIB and the version Conceptual.
• (optional) Select (click on) LIB.TXT and execute the command **Product / Properties**.
• (optional) Click on the button Connection and tell that LIB/Conceptual depends on this text; close the box (click on the button OK).

Result: A new (empty) schema named LIB/Conceptual appears in the project window (Figure 1.4); an arrow is drawn from the interview text to this schema to indicate that the (future) schema is a formal expression of the interview text.

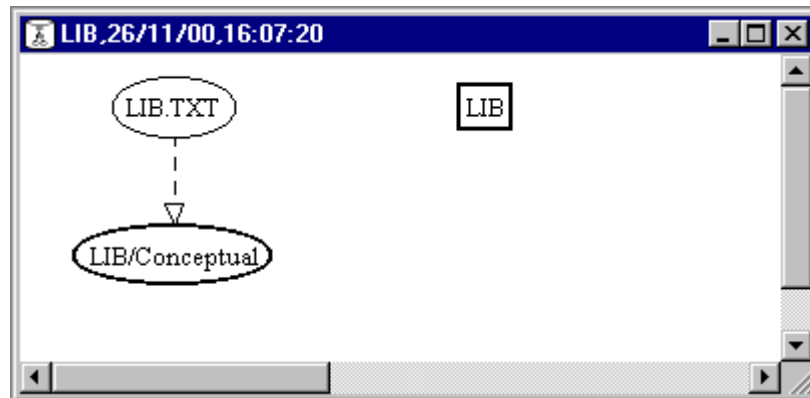



Figure 1.4 - A new, empty, schema named LIB has been created, with version *Conceptual*.

1.6 Define the entity types

Objective: To define the **entity types**³ which the interview text is talking about. The text suggests the existence of three entity types, describing respectively books, copies and borrowers.

- Actions:*
- Open the schema window by double clicking on the schema icon.
 - Define BOOK: select the "create entity type" tool by clicking on the button  (or command **New / Entity type**); double click somewhere in the space of the schema window; give the entity type the name BOOK; close the property box (click on the button OK).
 - Define COPY: double click elsewhere and give the name COPY.
 - Define BORROWER in the same way.
 - Close the "create entity type" tool by clicking on its button again or by pressing the Escape key.

Result: The schema window shows the three entity type boxes (Figure 1.5).

3. DEFINITION. **Entity type**: an entity is an outstanding object of the application domain; an entity type is a class of similar entities.

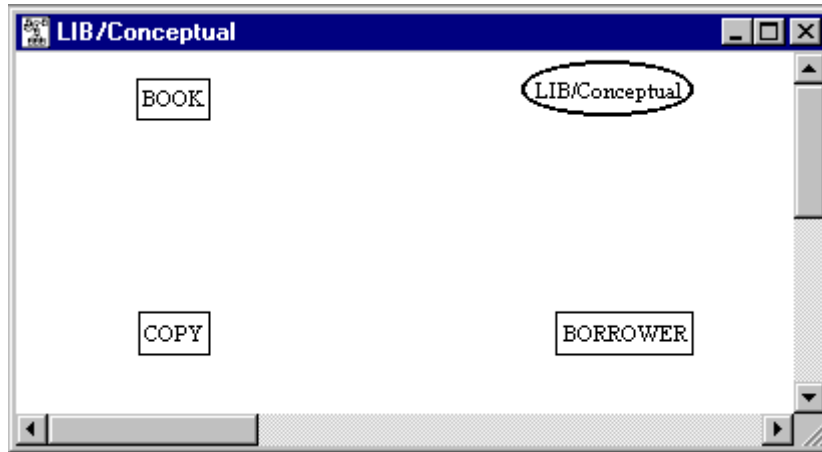


Figure 1.5 - Three entity types have been created to represent the main object classes suggested by the interview.

1.7 Define the attributes of entity type BOOK

Objective: To define the **attributes**⁴ of entity type BOOK. Each book has an *ISBN*, a *title* and *authors*.

- Actions:*
- Open (double click) the entity type BOOK; its property box opens.
 - Click on the button New att.; define the first attribute, with the name ISBN, type char and length 14.
 - Press the key Enter to define the next attribute, Title (*useless to close the property box by clicking on the button OK*);
 - Define the attribute Author in the same way; select the **cardinality**⁵ 1-5 to state that each BOOK entity has from 1 to 5 values of Author.
 - Terminate by clicking on the button OK.

Result: The entity type BOOK now has three attributes (Figure 1.6).

4. DEFINITION. **Attribute**: an intrinsic property of the entities of a given type; is given a value domain.

5. DEFINITION. **Cardinality**: a couple of integral numbers [i-j], where j can be N (standing for infinity); the most common cardinalities are [0-1], [1-1] and [0-N].

attribute cardinality: states that the number of values associated with each parent instance is between i and j;

role cardinality: states that the number of relationships in which each instance of the entity type taking this role is between i and j

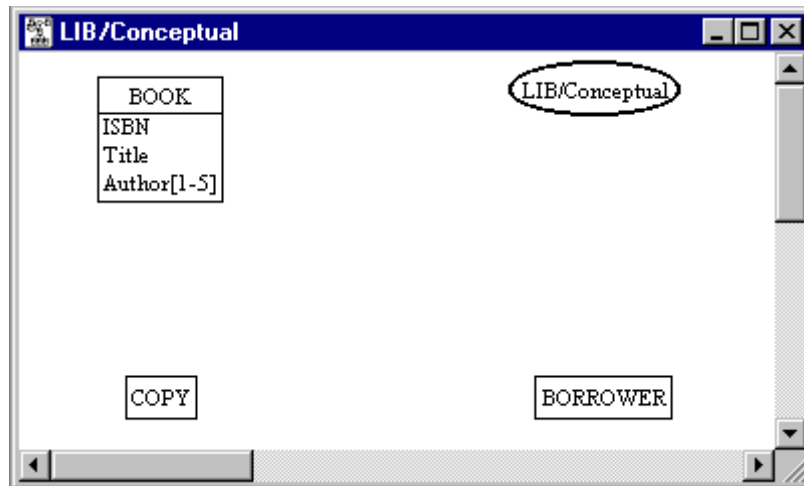


Figure 1.6 - The entity type BOOK has been given three attributes, that represent the common intrinsic properties of all books.

1.8 Define the attributes of entity types COPY and BORROWER

Objective: To define the attributes of the other entity types.

Actions:

- Define the attribute Serial number of COPY.
- Then define the next attribute, namely Location.
- This attribute happens to be made up of three **components**⁶; define them as follows: before closing the box of the attribute Location, click on the button First att., then define the attribute Store, (Enter), then Shelf, (Enter), and finally Row.
- Define the attributes Name and Address of BORROWER in the same way; the interview text suggests that the attribute Address is **optional**⁷; so, choose the cardinality 0-1 before closing the attribute property box.

Result: All the entity types have got their attributes (Figure 1.7).

6. DEFINITION. **Compound**: an attribute is compound if its values are made up of the concatenation of subattributes values (its components); otherwise, the attributes are atomic.

7. DEFINITION. **Optional**: if an attribute is optional, a parent instance may have no value for this attribute; otherwise, the attribute is mandatory

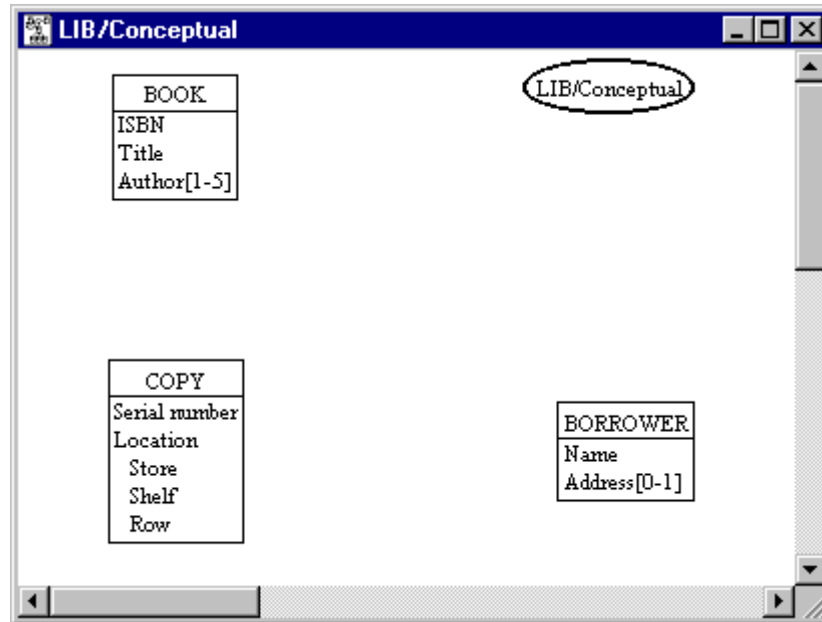


Figure 1.7 - The other entity types have received their own attributes.

1.9 Define the relationship types

Objective: The entities are not independent, but rather are linked through relationships. So, we define the **relationship types**⁸ that make these associations explicit.

Actions:

- Select the "create relationship type" tool by clicking on the button (or command **New / Role/rel-type**).
- Draw a line from the box of BOOK to that of COPY. Open the relationship type box by pressing the Enter key. Type the name 'of' and close the box (button OK).
- Draw another line from the box of BORROWER to that of COPY; open the relationship type box by pressing the Enter key. Type the name 'borrowed by' and close the box. Since not all copies are borrowed at each instant, we change the cardinality of the **role**⁹ COPY of 'borrowed by': we open the property box of

8. DEFINITION. **Relationship type** (or rel-type): a relationship is a group of entities logically related; a relationship type is a class of similar relationships; in a rel-type, each partner plays a specific role; a role is taken by an entity type and has a cardinality

9. DEFINITION. **Role**: a partner in a relationship type; a role is taken by an entity type and has a cardinality; a role can be given an explicit name

this role (by double clicking on the label '1-1') and we select the cardinality 0-1.

- Close the "create relationship type" tool by clicking on its button again or by pressing the Escape key.

Result: The relationship types between the entity types have been defined (Figure 1.8).

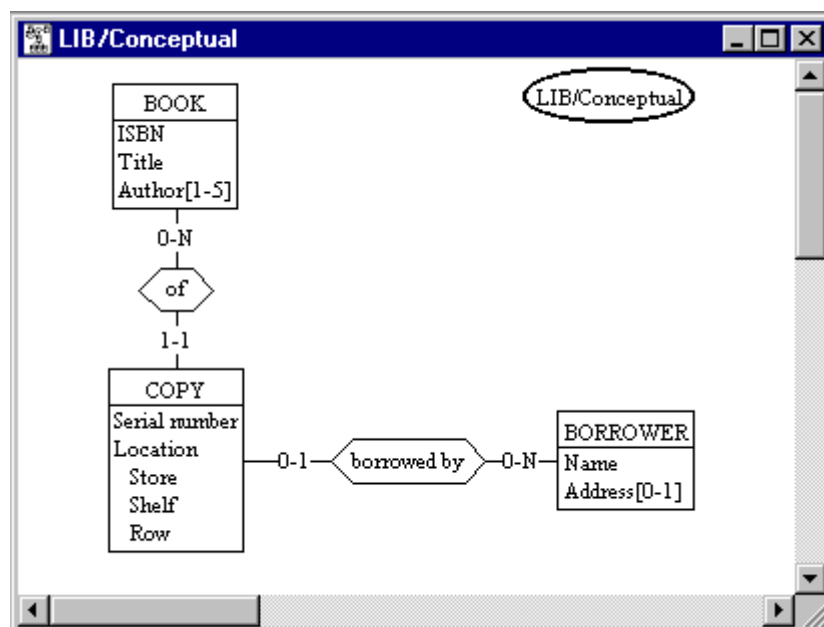





Figure 1.8 - Two relationship types define the links that hold between entity types.

1.10 Define the identifiers

Objective: To define the **identifiers**¹⁰ of each entity type, that is, the way the entities of each type can be identified, or uniquely denoted;

- Actions:*
- Define the identifier of BOOK: select the attribute ISBN, then click on the button .
 - Define the identifier of BORROWER: select the attribute Name, then click on the button .

10. DEFINITION. **Identifier**: the components of an entity type that can be used to uniquely designate an entity among its type; most often, an identifier is made of one or several attributes; it can also comprise one or more roles of rel-types in which the entity type participates; rel-types and multivalued attributes can have identifiers as well

- Define the identifier of COPY (this one is a bit more complicated): since each copy is identified by its serial number within the copies of its book, we make an identifier by selecting the attribute Serial number and (keep the Shift key depressed) the role BOOK (click on the label 0-N) of relationship type of; then click on the button .

Result: All the components of the conceptual schema have been defined (Figure 1.9). Their composition appears in the 3rd compartment of the entity type boxes. If an identifier is made up of attributes only, the latter are underlined in the attribute (2nd) compartment.

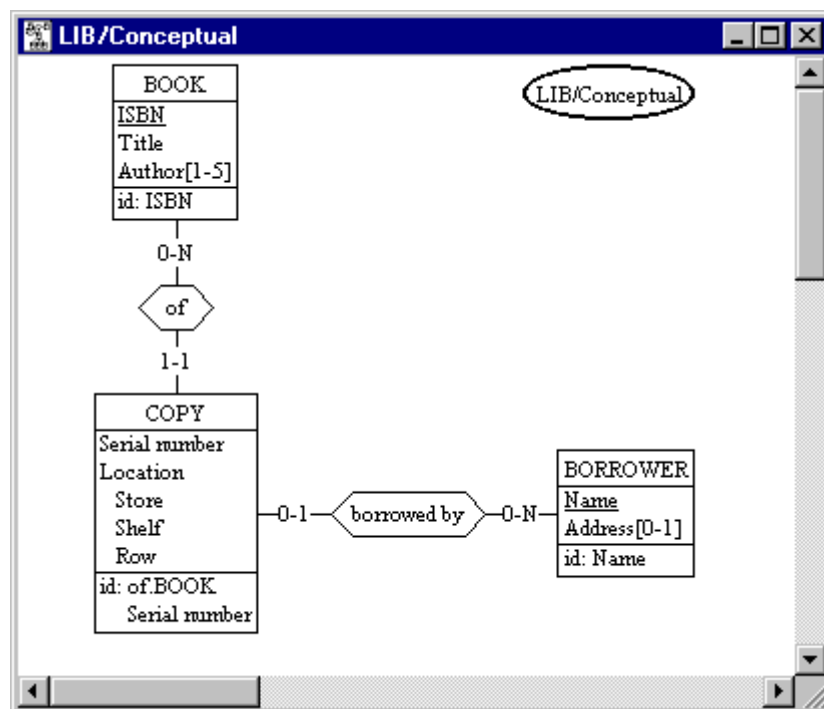


Figure 1.9 - For each entity type, we have defined the components of their respective identifier.

1.11 Document the objects

Objective: To associate with each object of the schema a precise description of its meaning.

Actions:



- Open (through a double click) the object to document.

- Click on the button SEM and introduce a text describing the meaning of this object (possibly by copying and pasting parts of the interview text).

Result: The schema is completely documented, and can be passed on to other persons without any risk of misinterpretation (*to be quite frank, we must recognize that this step seldom is carried out conscientiously, so that most conceptual schemas DO lead to misinterpretation!*).

1.12 Save the project

Objective: To record the state of the project on disk.

Action: • Click on the buttons  or  or execute the command **File / Save project as**


Result: The objective is reached. You can now switch your computer off safely. However, we rather recommend you to go on with the next phase!

Phase 2 - Generating a report of the conceptual schema

Objective. To produce a printable report describing the components of the conceptual schema.

2.1 Show the schema in text view

Objective: To present the schema as a pure text.

Action: • Click on  or **View / Text standard ...** (or any other Text view).

Result: Quite surprisingly, the schema now appears as a pure text.

2.2 Generate the report (LIB.dic)

Objective: To generate a text file that contains the text of the schema.

Action: • Execute the command **File / Report / Textual view**. Type the name LIB, and choose the target folder. Click on Show report generation. Click on the button OK.

Result: A new product appears in the project window, namely LIB.dic/1 (Figure 2.1). Drag it to the right side to make room for the following.

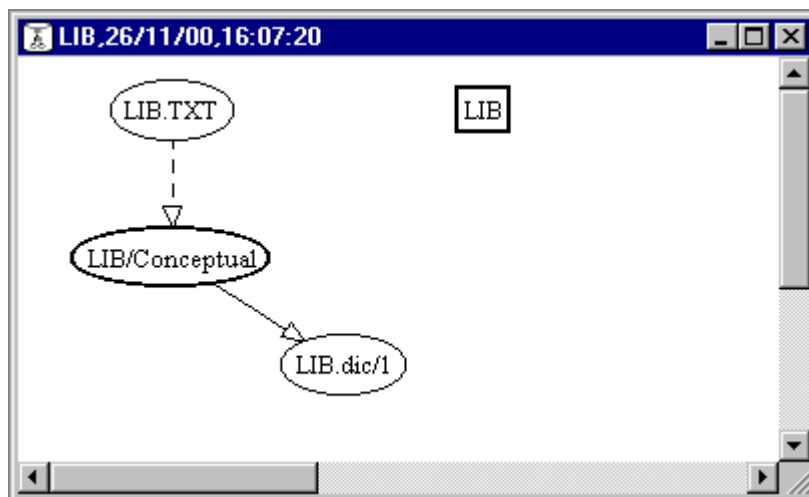


Figure 2.1 - An external text LIB.dic has been generated. It contains a simple report that list the components of the conceptual schema.

2.3 Reading and printing the report

Objective: To examine the contents of the report and to send it to the printer.

- Actions:*
- Open (double click) the report LIB.dic/1 that appears in the project window.
 - Execute the command **File / Print**.

Dictionary report

Project LIB

Schema LIB/Conceptual

```

BOOK
  ISBN
  Title
  Author[1-5]
  id: ISBN
BORROWER
  Name
  Address[0-1]
  id: Name
COPY
  Serial number
  Location
    Store
    Shelf
  Row
  id: of.BOOK, Serial number

borrowed by (
  [0-N] : BORROWER
  [0-1] : COPY )
of (
  [0-N] : BOOK
  [1-1] : COPY )

```

Phase 3 - Producing the relational schema of the database

Objective. To produce the data structures that implement the constructs of the conceptual schema according to the **relational database model**¹¹. This schema mainly comprises tables, columns, primary keys and foreign keys.

Preliminary remark

There is a lot of possible procedures that produce relational structures from a conceptual schema. The simplest would be the following:

- Actions*
- Open the conceptual schema.
 - Execute the command **Transform / Relational model**.

That's all. (*You even get a physical schema. For free!*)

However, this technique will teach us nothing on the very nature of database design. Thus, we will follow a more detailed procedure through which we will examine some of the main processes and products that real-size database projects are made up of.

If this walk seems to you a bit tedious, particularly when compared with the procedure described above, you will learn in the DB-MAIN mini-tutorial¹² how to automate the process to the exact extent that suits your needs.

11. DEFINITION. **Relational model:** the way the data are organized in a relational database managed by a RDBMS; a relational schema is made up of tables, columns, primary keys (identifiers) and foreign keys, to mention the most important constructs

12. NOTE. *Introduction to Database Design*, available on the distribution CD-ROM and on the DB-MAIN web site

3.1 Create a copy of the conceptual schema (LIB / Relational).

Objective: To create a new schema in which the relational database structures will be developed.

Actions:

- In the project window, select the conceptual schema.
- Execute the command **Product / Copy product**. Type "Relational" in the field Version.
- Double click on the new schema to open it.

Result: A new schema appears in the project window (Figure 3.1). Its contents are a mere copy of that of the conceptual schema, but it will progressively be transformed into purely relational constructs.

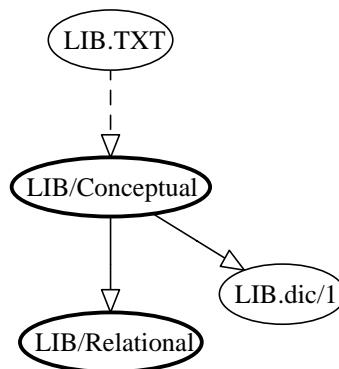


Figure 3.1 - A new schema has been derived from the conceptual schema. It is intended to contain the relational structures that implement the conceptual constructs.

3.2 Transform the multivalued attribute Author into an entity type

Objective: **Multivalued attributes**¹³ are not allowed in the relational model. We will get rid of the multivalued attribute Author[1-5] by extracting it as an autonomous entity type.

Actions:

- Select the attribute Author[1-5].
- Execute the command **Transform / Attribute / -> Entity type**.
- Choose the mode value representation.
- Give the new relationship type a significant name.

13. DEFINITION. **Multivalued attribute:** a multivalued attribute can associate more than one value with each parent instance; otherwise, the attribute is single-valued

Result: A new entity type, Author, is defined (Figure 3.2), containing the attribute Author, and linked to BOOK. Note the cardinality of Author in written by¹⁴.

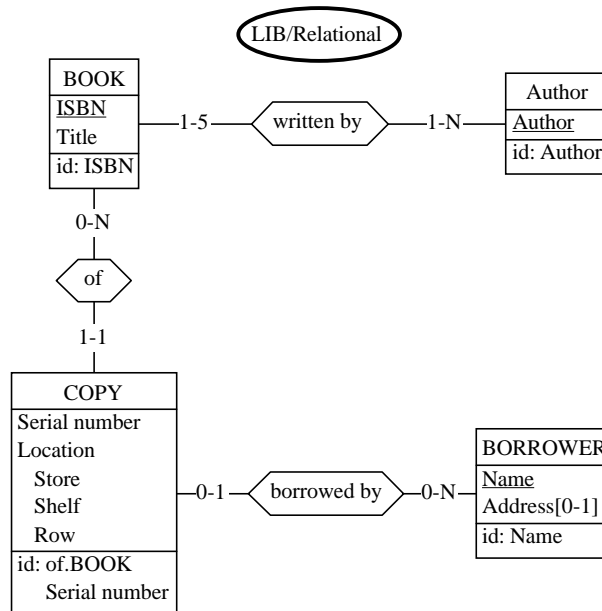


Figure 3.2 - First, the multivalued attributes (Author) are transformed into separate entity types.

3.3 Transform the *many-to-many* relationship type that just appeared (written by) into an entity type

Objective: Relationship types are not allowed in the relational model.

We first tackle the complex *many-to-many*¹⁵ relationship type written by, produced in the previous step, and we transform it into a new entity type and two, simpler, relationship types.

14. NOTE. **Cardinality of written by:** the entity type Author has a cardinality [1-N] in the relationship type written by; this means that an Author cannot be referenced in the database without him/her having written a book known by the library; this constraint directly translate the fact that, in the conceptual schema, the author(s) of a book is a mere attribute; in other words, an author "exists" only if it appears in at least one book; hence the cardinality; see later on how this cardinality will be translated into equi foreign keys

15. DEFINITION. **Many-to-many:** a many-to-many rel-type has two roles (binary); their cardinalities are such that $j > 1$ (e.g., [0-N]); a binary rel-type that is not many-to-many is one-to-many or one-to-one; the link between PLANT and PRODUCT is many-to-many if each plant can manufacture several products and each product can be manufactured by more than one plant

- Actions:**
- Select the relationship type written by.
 - Execute the command **Transform / Rel-type / -> Entity type**; choose significant names.
 - Move the objects to make the schema readable.
- Result:** A new entity type, written by, is defined, as well as two *one-to-many*¹⁶ relationship types (Figure 3.3). The schema now includes simple rel-types only.

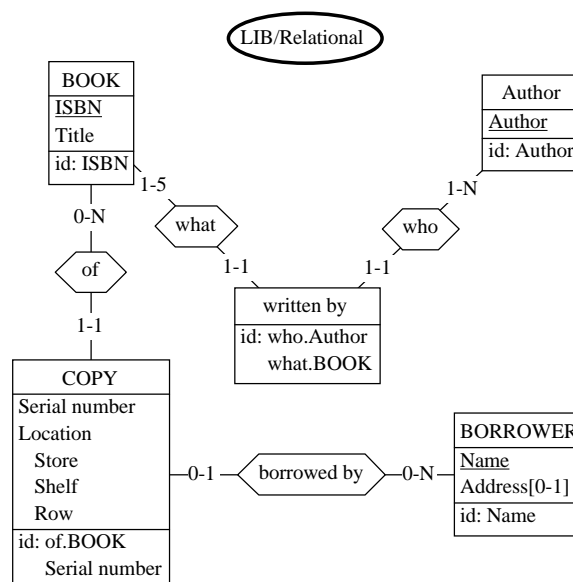


Figure 3.3 - Then, the many-to-many relationship types (written by) are transformed into entity types and simple relationship types.

3.4 Disaggregate the compound attribute Localisation

Objective: **Compound attributes**⁶ are not allowed in the relational model. We replace them with their components.

- Actions:**
- Select the attribute Location.
 - Execute the command **Transform / Attribute / Disaggregation**.

16. DEFINITION. **One-to-many**: a one-to-many rel-type has two roles (binary); the cardinality of one of them is such that $j=1$ (e.g., $[0-1]$ or $[1-1]$); one-to-one rel-types (both roles have a cardinality with $j=1$) can be considered as a special case of one-to-many rel-types; the link that expresses that CUSTOMERs places ORDERs is one-to-many if each customer can place several orders while each order is placed by one customer only

- Accept the proposed prefix.

Result: The compound attribute has been replaced with its three components. Their names have been prefixed with the short name of their former parent.

Now, all the attributes are **single-valued**¹³ and **atomic**⁶, that is, quite compliant with the relational model (Figure 3.4).

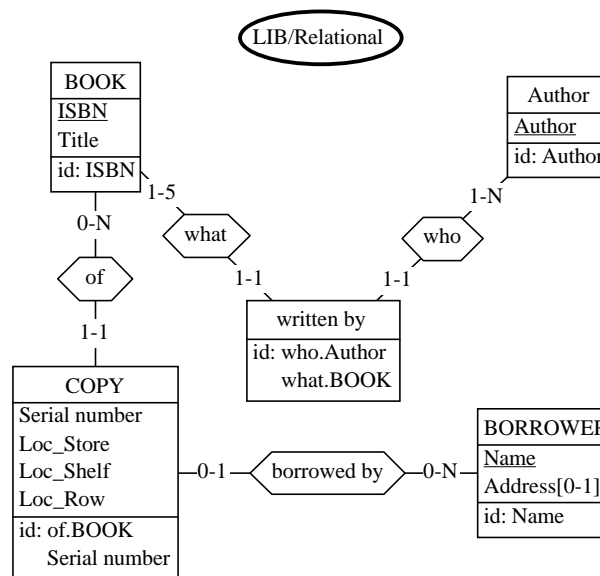


Figure 3.4 - Compound attributes (Location) are disaggregated.

3.5 Transform the *one-to-many* relationship types into foreign keys (1)

Objective: As already mentioned, relationship types are not allowed in the relational model. The schema still includes four *one-to-many* relationship types.

We will express the relationship type borrowed by into a **foreign key**¹⁷.

- Actions:*
- Select the relationship type borrowed by.
 - Execute the command **Transform / Rel-type / -> Attribute**.
 - Accept the proposal to insert a foreign key in the table COPY.

17. DEFINITION. **Foreign key:** a group of attributes (or columns) that can be used to identify another entity (a row) through its identifier (primary key).

- Choose the name of the column forming the foreign key (say Borrower); click on the button OK.

Result: A new column, Borrower, has been added to COPY, and declared a foreign key (ref) to BORROWER (Figure 3.5).

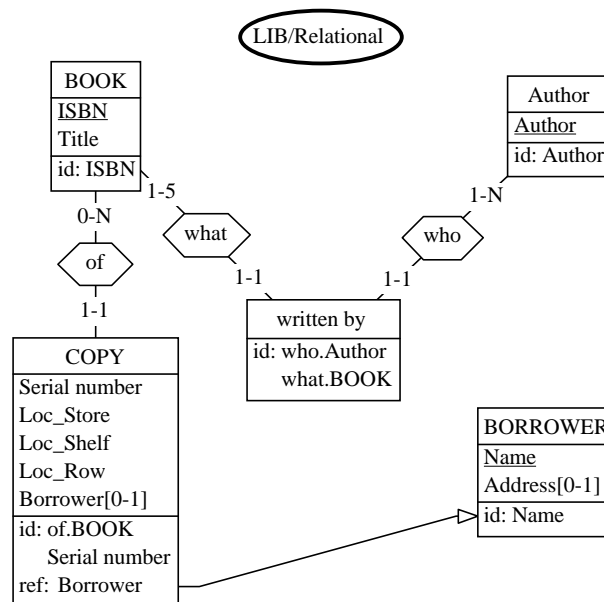


Figure 3.5 - The first *one-to-many* relationship type (borrowed by) is transformed into a foreign key.

3.6 Transform the *one-to-many* relationship types into foreign keys (2)

Objective: To transform all the remaining relationship types into foreign keys.

To make the process faster, we will make use of an **assistant**¹⁸ to process all of them in one step.

Actions:

- Call the *Global Transformation Assistant: Assist / Global transformation*.

18. DEFINITION. **Assistant:** an expert processor of the DB-MAIN tool that can help solve complex problems and carry out tedious tasks;
some available assistants: global transformation (elementary and advanced), schema analysis, schema integration (merging), foreign key elicitation, text analysis, etc.
 most assistants allow the user to develop scripts that automate parts of the engineering processes

- Click on the radio button Rel-types and select the item Binary 1-N in the list box. Click on the button OK.

Result: All the *one-to-many* ("Binary 1-N") rel-types have been transformed into *foreign keys* (Figure 3.6).

Note that some foreign keys are noted with symbol *ref* while *others are noted with symbol equ*¹⁹.

The data structures of the schema are compliant with the relational model.

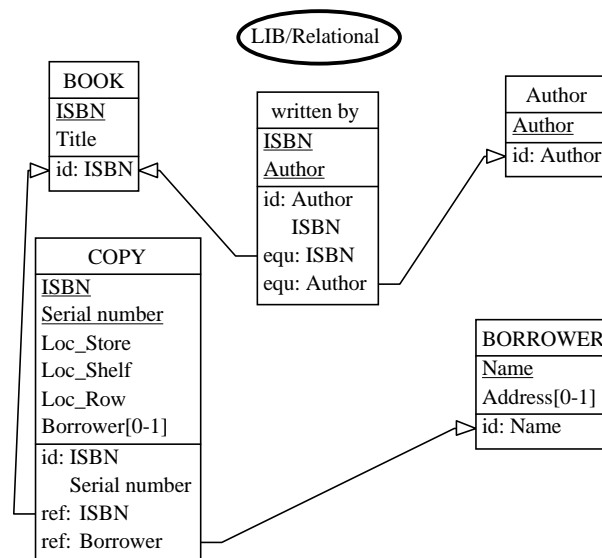


Figure 3.6 - All the other relationship types are transformed in the same way.

3.7 An artistic touch ...

Objective: To give the schema a look that shows that we are talking about tables and columns, and all that sort of technical things, and no longer about abstract conceptual constructs such as entity types and attributes.

19. NOTE. **equ**: let us consider the foreign key {equ: Author} from *written by* to *Author*; being a (mandatory) foreign key, each of its values must appear in the set of values of the column *Author* of the table *Author* (sorry for the confusing names!); in addition, the infamous cardinality [1-N] stated that each *Author* entity must be connected with a *written by* entity, which translates, in our relational schema, into the property that each value of *Author* of the table *Author* must be a value of *Author* of *written by* as well; in short, both sets of values are equal, hence the symbol **equ** which replaces the symbol **ref**; this property will be particularly difficult to translate in SQL.

- Actions:**
- Execute the command **View / Graphical settings**.
 - In the Entity types panel, check (click on) the radio button **Shaded**.
- Result:** The entity type boxes have been *shaded* to suggest that they now denote concrete, technical objects (Figure 3.7).

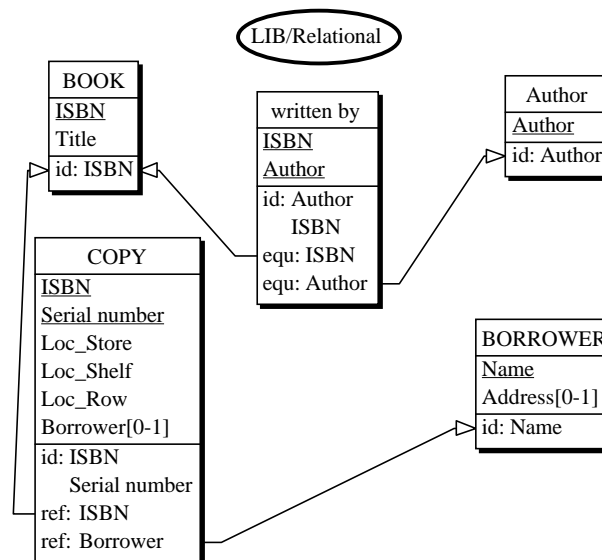


Figure 3.7 - Entity types have been *shaded*, to suggest that they have been transformed into relational tables.

3.8 Transforming names

Objective: To give the schema components names that fully comply with the syntax of SQL. In particular, we will replace spaces with, say, underscores and we will transform all the names in uppercase (not mandatory, but gives a more professional look!)

- Actions:**
- Execute the command **Transform / Name processing**.
 - Verify that the buttons **Global**, **Names**, **Entity types** and **Attributes** are checked.
 - Add a substitution pattern:
 - %o click on the button Add,
 - %o enter a space character in the field Search for
 - %o enter an underscore character in the field Replace by
 - %o Click on the button OK.
 - Click on the button lower -> uppercase.

- Click on the button OK.

Result: All the names have been transformed in such a way that the schema is now fully compliant with the relational model and the SQL syntax (Figure 3.8).

Note that other invalid symbols such as dashes ("-") or apostrophes ("'") could be transformed with specific substitution patterns.

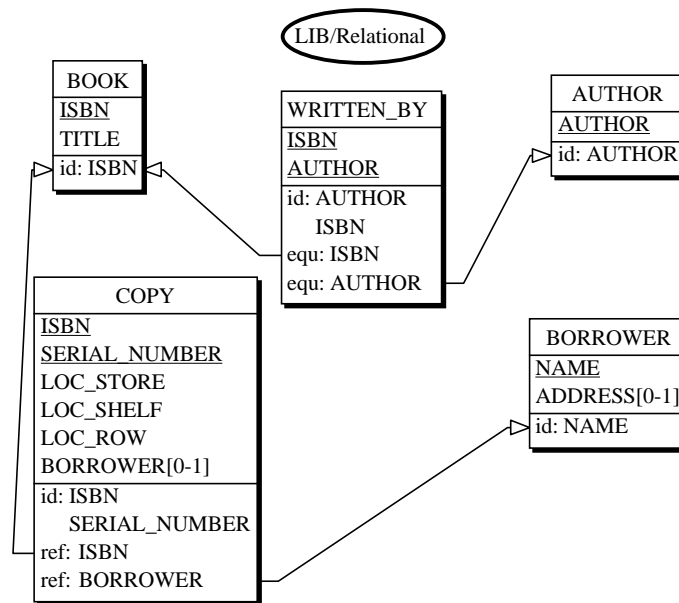



Figure 3.8 - All the table and column names are translated into SQL-compliant names.

3.9 Save the project

Action: Click on the button .

Phase 4 - Producing the physical schema of the database

Objective. To augment the relational schema with technical constructs such as **indexes**²⁰ and **storage spaces**²¹. A logical schema that includes performance-oriented technical constructs is called physical.

4.1 Create a copy of the logical schema (LIB / SQL)

Objective: To create a new schema in which the physical database structures will be developed.

Actions:

- In the project window, select the relational schema LIB/Relational.
- Execute the command **Product / Copy product**. Give the new schema the Version name SQL.
- Double click on the new schema to open it.

Result: A new schema appears in the project window, with name LIB/SQL (Figure 4.1). Its contents are a mere copy of that of the relational schema, but it will be enriched with technical constructs in the following steps.

4.2 Define the indexes (access keys)

Objective: To define the most useful indexes.

Actions:

- Double click on *each identifier* and *each foreign key* (noted id, ref and equ in the third compartment of the table box, not in the second one!)
- Check (click on) the button Access key.

Result: The symbol acc (for access key) is associated with each identifier and each foreign key (Figure 4.2).

20. DEFINITION. **Index:** a technical mechanism that provides fast access to the rows of a table that have given values for selected columns; an index is based on one or more columns of the table; such mechanisms are called *access keys* in DB-MAIN.

21. DEFINITION. **Storage space:** a physical container in which the rows of one or several tables are stored; storage spaces often are implemented as *files*; called (*entity*) *collection* in the DB-MAIN physical model.

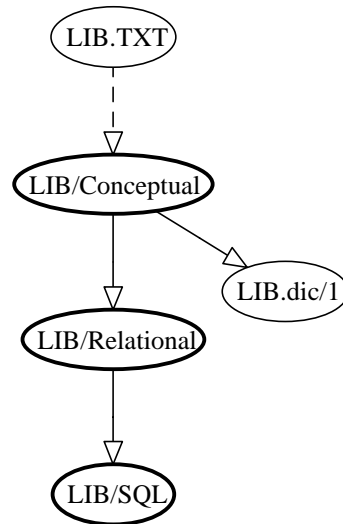


Figure 4.1 - A new schema is derived from the relational schema. It will contain the SQL physical schema of the database.

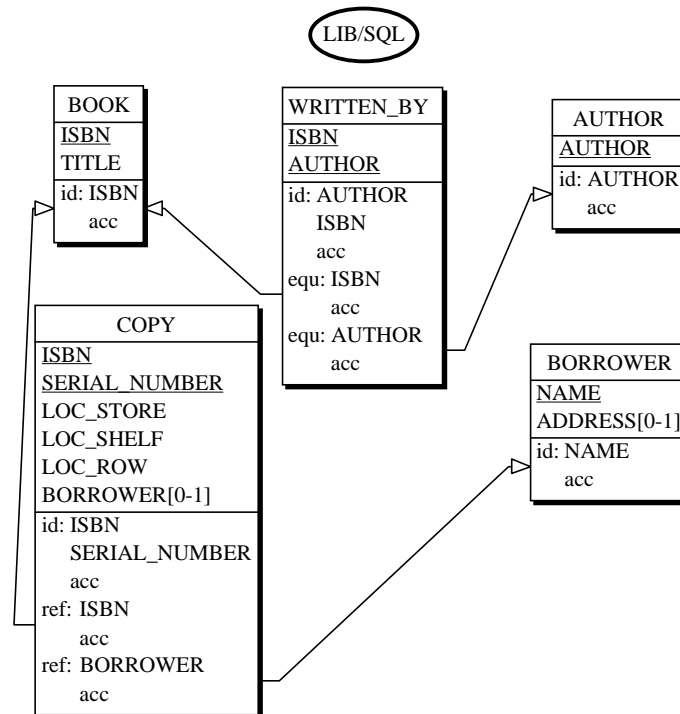


Figure 4.2 - The main indexes have been defined. They are associated with identifiers and foreign keys.

4.3 Optimization: remove the prefix indexes

Objective: To remove all the indexes (access keys) whose components appear in the first position in another index.

Such indexes are useless, since the SQL engine can use the larger index instead, without any performance penalty.

Actions

- Open the property box of the foreign key {equ: Author} of table WRITTEN_BY.
- Uncheck the button access key. Click on the button OK.
- Proceed in the same way with the foreign key {ref: ISBN} of the table COPY.

Result: All the prefix access keys have been removed, thus minimizing the number of indexes (Figure 4.3).

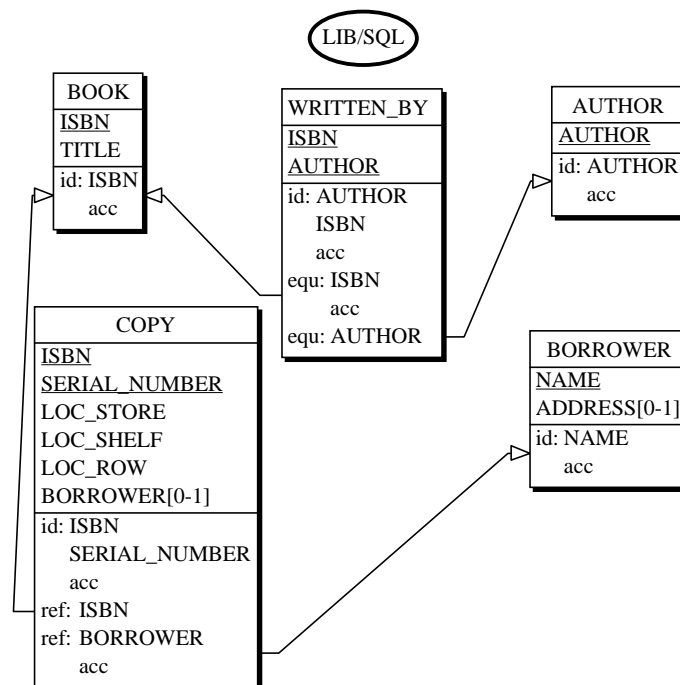



Figure 4.3 - All prefix indexes, that is those whose components are the first components of other indexes, are removed.

4.4 Define the storage spaces (collections)

Objective: Each table is assigned to a storage space (called collection in the DB-MAIN model).

More than one table can be assigned to the same storage space.

- Actions:**
- Click on the button  (or **New / Collection**) and create a collection.
 - Open it by double clicking (or pressing the Enter key) and name it BOOK_SPC.
 - Move the table names BOOK, WRITTEN_BY and AUTHOR from the right side list to the left side list. Click on the button OK.
 - Same job for the tables COPY and BORROWER, to be stored in the storage space BORROW_SPC.

Result: Now, table rows can be stored in safe containers! (Figure 4.4)

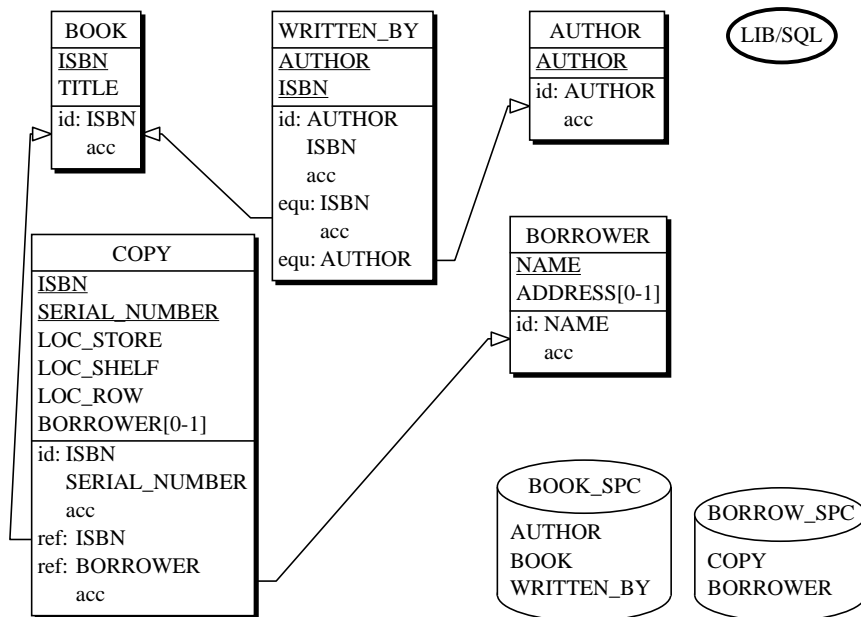



Figure 4.4 - The storage spaces, in which the table rows will be stored, have been defined.

4.5 Save the project

Action: Click on the button .

Phase 5 - Generate the SQL code that creates the database

Objective. To produce the SQL-DDL program that can be executed by a database engine to create the data structures of the physical schema.

5.1 Generate the SQL code of the physical schema (LIB.ddl)

Objective: To generate and introduce in the project the SQL script that codes the physical schema.

Actions:

- Open the physical schema.
- Execute the command **File / Generate / standard SQL (check)**. If needed, change the name and the folder of the generated text file.

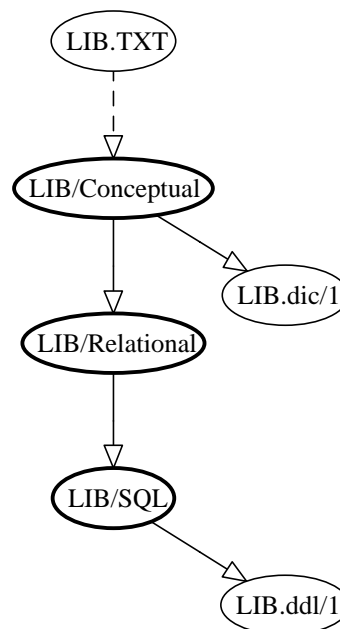


Figure 5.1 - The final SQL text has been generated in an external text.

Result: A new product, called LIB.ddl/1, appears in the project window (Figure 5.1). This text can be submitted to a RDBMS²² to create

the database corresponding to the library described in the interview report (LIB.TXT).

5.2 Examine the resulting code

Actions: In the project window, double click on LIB.ddl/1.

Result: The text comprises different sections, corresponding to the various constructs of the schema : the database itself, the storage spaces, the tables, the constraints and the indexes. This text may lead to compilation errors for some SQL engines. In particular, the equi foreign keys have been translated into check predicates that will be rejected by many RDBMS, such as DB2 and Oracle. For them, **alternative techniques**²³ will be used, that are outside the scope of this simple tutorial.

```
-- *****
-- * Standard SQL generation *
-- *-----*
-- * Generator date: Nov 28 2000 *
-- * Generation date: Thu Dec 07 09:15:11 2000 *
-- *****

-- Database Section --
create database LIB;

-- DBSpace Section --
create dbspace BORROW_SPC;
create dbspace BOOK_SPC;

-- Table Section --
create table AUTHOR (
    AUTHOR char(24) not null,
    primary key (AUTHOR))
in BOOK_SPC;

create table BOOK (
    ISBN char(14) not null,
    TITLE char(36) not null,
    primary key (ISBN))
in BOOK_SPC;

create table BORROWER (
    NAME char(20) not null,
    ADDRESS char(50),
    primary key (NAME))
```

22. DEFINITION. **RDBMS**: Relational Database Management System. Some examples: DB2, ORACLE, InterBase, Sybase, Informix, MySQL, Access (not easy to feed Access with SQL, but *not impossible!*), etc.

23. NOTE. **Advanced SQL translation techniques**: the parametric SQL generator of the DB-MAIN tool provides us with more sophisticated techniques for complex constructs and constraints (including *triggers* and *stored procedures*).

```
in BORROW_SPC;


create table COPY (
    ISBN char(14) not null,
    SERIAL_NUMBER char(1) not null,
    LOC_STORE numeric(2) not null,
    LOC_SHELF numeric(2) not null,
    LOC_ROW numeric(2) not null,
    BORROWER char(20),
    primary key (ISBN, SERIAL_NUMBER))
in BORROW_SPC;

create table WRITTEN_BY (
    AUTHOR char(24) not null,
    ISBN char(14) not null,
    primary key (AUTHOR, ISBN))
in BOOK_SPC;

-- Constraints Section --
alter table AUTHOR add constraint
    check(exists(select * from WRITTEN_BY
        where WRITTEN_BY.AUTHOR = AUTHOR));
alter table BOOK add constraint
    check(exists(select * from WRITTEN_BY
        where WRITTEN_BY.ISBN = ISBN));
alter table COPY add constraint FKOF
    foreign key (ISBN)
    references BOOK;
alter table COPY add constraint FKBORROWED_BY
    foreign key (BORROWER)
    references BORROWER;
alter table WRITTEN_BY add constraint FKWHAT
    foreign key (ISBN)
    references BOOK;
alter table WRITTEN_BY add constraint FKWHO
    foreign key (AUTHOR)
    references AUTHOR;

-- Index Section --
create unique index IDAUTHOR
    on AUTHOR (AUTHOR);
create unique index IDBOOK
    on BOOK (ISBN);
create unique index IDBORROWER
    on BORROWER (NAME);
create unique index IDCOPY
    on COPY (ISBN, SERIAL_NUMBER);
create index FKBORROWED_BY
    on COPY (BORROWER);
create unique index IDWRITTEN_BY
    on WRITTEN_BY (AUTHOR, ISBN);
create index FKWHAT
    on WRITTEN_BY (ISBN);
```

5.3 Save the project

Action: Click on the button .

Conclusions

This is the end of the walk. It is also time to list the concepts, techniques and practices you should have learned during this walk:

- a database project comprises a set of coherent and synchronized products (documents);
- the role of conceptual, relational and physical schemas;
- what is a conceptual schema, and what it is made up of;
- what is a relational schema, and what it is made up of;
- what is a physical schema, and what it is made up of;
- how a conceptual schema can be drawn;
- how a relational schema can be derived from a conceptual schema
- how a physical schema can be derived from a relational schema
- that a SQL script is a translation of a physical schema
- the concept of transformation
- the concept of assistant

In short, you are able to develop in a systematic way a (small) SQL database from a plain text that describes user's requirements.

We had a look at what the terms database analysis and design could mean, and how CASE tools can help carrying out these processes. Just think that in a true database project, all the aspects can be larger and more complex by orders of magnitude.

In a real-size database project:

- A database can include thousands of tables. For instance, the SAP database can comprise about 30,000 tables and 200,000 columns. 1000-table databases are not uncommon in many companies. Their conceptual schema could span several square meters!
- A SQL script can be several thousand pages long.
- A SQL script declares tables, columns and keys. It also declares more sophisticated constructs such as predicates (check), triggers, stored procedures, user's privileges, transactions and numerous physical parameters.
- The database is just one component of information systems. The other parts must be specified and generated through specific models and processes.

- Most databases are passive, in that they merely store static data that are used and managed by application programs. Advanced databases can take in charge some aspects of their dynamics. These active databases are built with the help of triggers and stored procedures. Their design, development and tuning are far more complex than for passive databases.
- A database can be implemented on several sites, through different RDBMS.
- The problem we solved was fairly simple, limited to elementary data structures and types (character, numeric, date, etc.) Many databases include more sophisticated data, such as spatial and temporal data, semi-structured data, multimedia data, etc.
- Besides relational databases, object-relational and object-oriented databases, or even mere file and record structures, can be derived from a conceptual schema.
- A database can include legacy components such as foreign databases and COBOL files.
- A database can result from the reengineering of a legacy database. Redocumenting this database generally is a complex and tedious process called Data reverse engineering.
- Databases evolve with time, just as programs do. Controlling this evolution is a complex process too.
- A database is strongly linked with its environment, including the Internet and complex documents exchange (through XML for instance).
- The interview report can be several thousand pages long, and include information on a great variety of media: reports, commercial leaflets, legal documents, voice/video documents, electronic documents, existing software, etc.
- The design team can comprise several dozens of analysts, so that cooperative work raises critical issues. For instance, how to integrate several large conceptual schemas into a single one, when they include many incompatibilities and discrepancies?
- All CASE tools can support simple tasks such as those we carried out in this tutorial. Large organizations often have special needs as far as methodology is concerned. Therefore, they will use CASE tools that are customizable.

All that to say that database engineering is a more exciting and challenging discipline than we could present it through this oversimplistic tutorial!

And now what?

To go further, read the DB-MAIN mini-tutorial:

- *Introduction to Database Design*, 5th Edition, 2002

and the in-depth tutorial on Database models:

- *Computer-Aided Database Engineering. Volume 1: Database Models*, 1999

Both can be found as PDF documents on the DB-MAIN distribution CD-ROM or on the DB-MAIN Web site.

<http://www.info.fundp.ac.be/libd>