

COMP23111: Fundamentals of Databases Practical Sessions Manual

2015-2016 edition

Alvaro A A Fernandes, Sandra Sampaio, Klitos Christodoulou
School of Computer Science
University of Manchester

| | |
|--|-----------|
| Introduction | 6 |
| <i>Aims</i> | <i>6</i> |
| <i>General Approach</i> | <i>6</i> |
| <i>On Plagiarism and Working Together</i> | <i>6</i> |
| General Information | 8 |
| <i>Communication and Announcements</i> | <i>8</i> |
| <i>Oracle Usernames and Passwords</i> | <i>8</i> |
| <i>Attendance</i> | <i>8</i> |
| <i>Timetable</i> | <i>8</i> |
| Lab Sessions/Example Clinics | 9 |
| <i>The Orinoco Case Study Used in Lab Exercises</i> | <i>9</i> |
| <i>Interpreting Examples, Requirements and Specifications</i> | <i>9</i> |
| On Specifications, Marking, Feedback, Deadlines and Extensions | 11 |
| <i>Specifications</i> | <i>11</i> |
| <i>Marking and Feedback</i> | <i>11</i> |
| <i>Lab Exercise Deadlines</i> | <i>11</i> |
| <i>Extensions to Lab Exercise Deadlines</i> | <i>12</i> |
| How to Work on and How to Submit Your Coursework | 13 |
| <i>COMP23111 Coursework Folders, Templates, Example Files, Scripts</i> | <i>13</i> |
| <i>Personalising the Files We Have Given You</i> | <i>15</i> |
| <i>Doing the Actual Work on the Exercise</i> | <i>15</i> |
| <i>What Each Submission Consists Of</i> | <i>15</i> |
| <i>Preparing the Files for Generating the Submission</i> | <i>16</i> |
| <i>Verifying You Have All the Files for Generating the Submission</i> | <i>17</i> |
| <i>Generating the Files for Submission</i> | <i>17</i> |
| <i>Submitting Your Coursework</i> | <i>17</i> |
| Examples Clinic 1 | 19 |

| | |
|--|-----------|
| <i>Understanding the Relational Paradigm</i> | 19 |
| <i>Specification</i> | 19 |
| Lab Exercise 01 | 22 |
| <i>First Steps with SQL*Plus and Dia for (E)ER Modelling</i> | 22 |
| <i>Specification</i> | 22 |
| <i>Overview</i> | 22 |
| <i>Introduction</i> | 22 |
| <i>Requirements</i> | 23 |
| <i>Instructions</i> | 23 |
| <i>Report Contents</i> | 23 |
| <i>Part 1: SQL*Plus TUTORIAL</i> | 23 |
| <i>Part 2: Getting Started with Dia</i> | 30 |
| Examples Clinic 2 | 32 |
| <i>Entity-Relationship Modelling</i> | 32 |
| <i>Specification</i> | 32 |
| Lab Exercise 02 | 34 |
| <i>(E)ER Modelling</i> | 34 |
| <i>Specification</i> | 34 |
| <i>Overview</i> | 34 |
| <i>Introduction</i> | 34 |
| <i>Requirements</i> | 34 |
| <i>Instructions</i> | 34 |
| <i>Report Contents</i> | 34 |
| <i>Part 1: Understanding the ER schema for the partial Orinoco database</i> | 35 |
| <i>Part 2: Extending the ER schema for the Orinoco database</i> | 36 |
| Examples Clinic 3 | 37 |
| <i>Mapping Entity-Relationship Diagrams into Relational Schemas</i> | 37 |
| <i>Specification</i> | 37 |

| | |
|--|-----------|
| Lab Exercise 03 | 38 |
| <i>EER-to-R Mapping, Functional Dependencies, SQL DDL/DML</i> | 38 |
| <i>Specification</i> | 38 |
| <i>Overview</i> | 38 |
| <i>Introduction</i> | 38 |
| <i>Requirements</i> | 38 |
| <i>Instructions</i> | 38 |
| <i>Report Contents</i> | 38 |
| <i>Part 1: Applying the EER-to-relational mapping to the extended Orinoco database</i> | 39 |
| <i>Part 2: Updating the schema/populating new relations in the Orinoco database</i> | 39 |
| Examples Clinic 4 | 42 |
| <i>Normalising Relational Schemas</i> | 42 |
| <i>Specification</i> | 42 |
| Lab Exercise 04 | 43 |
| <i>Making Use of a Relational Database</i> | 43 |
| <i>Specification</i> | 43 |
| <i>Overview</i> | 43 |
| <i>Introduction</i> | 43 |
| <i>Requirements</i> | 43 |
| <i>Instructions</i> | 43 |
| <i>Report Contents</i> | 43 |
| <i>Part 1: Writing simple queries as RA expressions</i> | 43 |
| <i>Part 2: Writing and executing complex SQL queries</i> | 44 |
| <i>Part 3: Using SQL queries to model data requirement scenarios</i> | 45 |
| Examples Clinic 5 | 46 |
| <i>Advanced SQL and Transactions</i> | 46 |

| | |
|---|-----------|
| <i>Specification</i> | 46 |
| Lab Exercise 05 | 48 |
| <i>Advanced SQL: Triggers and Procedures</i> | 48 |
| <i>Specification</i> | 48 |
| <i>Overview</i> | 48 |
| <i>Introduction</i> | 48 |
| <i>Requirements</i> | 48 |
| <i>Instructions</i> | 48 |
| <i>Report Contents</i> | 48 |
| <i>Part 1: Writing triggers</i> | 49 |
| <i>Part 2: Writing procedures</i> | 49 |

Introduction

Aims

This course unit aims to introduce the fundamentals of databases from one particular viewpoint, viz., that of business information systems, where modern database management systems play a pivotal role.

The lectures cover key topics in the area of database design and implementation while the practical sessions, which are the focus of this manual, provide substantial illustration of the topics covered, allowing you to apply and practice some of the notions and concepts introduced in the lectures.

Note that, in this course unit, we refer to practical sessions that are assisted, but not assessed, as *example clinics* because, although we recommend attendance (and record it), they are not strictly compulsory, as no marks are available in these sessions. However, if we slip into calling them *example classes*, please do not feel confused, as we do not mean something else, i.e., a different activity.

General Approach

The lab exercises involve the design and construction of a fairly complete database from a hypothetical case study. The sequence of lab exercises will be seeded by a partially completed database implementation that is provided to you.

The lab exercises are conducted in **five stages**. The first stage is mainly an introduction to the tools you need to use, and therefore contains little in the way of problem-solving. The following four, in contrast, contain significant problem-solving tasks.

It is very important that you bear in mind that the case study is **incremental** in the sense that later lab exercises build on **your own answers** to earlier ones. However, and very importantly, model answers are not handed out after each exercise (i.e., there is **no** reset after each lab exercise).

This means that you yourself must be aware of any deviation from the correct solution (broadly sketched out for you in the feedback you get from each exercise) and, if you have made any mistakes, fix the state of your coursework thus far. This needs to happen every time you get feedback on a piece of submitted coursework. In other words, from the start, think of the sequence of exercises as, in fact, a single task comprising several incremental stages. This mirrors real-world database design and development, which is a multistage process that unfolds over a period of months, rather than days or weeks.

When the coursework has been marked, we aim to give you such feedback as will help you fix any problems in your submission, so that the subsequent exercise is done from a correct intermediate position. If you want to discuss the feedback we have given you, please use the various mechanisms available, viz., the forum (for general questions) or a chat with us at a lab session, or else, for more specific and personalised feedback, use a direct email to the lecturers or the regular weekly one-hour open-office slot (see details of that in [the Moodle webpage for the course unit](#)).

On Plagiarism and Working Together

You must always work individually. You must only consult and discuss things with your friends once you have gained a thorough understanding of what constitutes academic malpractice. You are not allowed to copy solutions from anyone else, or to pass solutions to anyone else, in any form (conversation, paper, electronic media, etc.), unless otherwise authorised explicitly.

If you have not done so yet, make absolutely sure that you understand and abide by the guidance on plagiarism and cheating:

<http://studentnet.cs.manchester.ac.uk/assessment/plagiarism.php>

You should remind yourself of, and be wise to follow, John Latham's thoughts on the issue of students working together. Here is a reminder (very lightly adapted) for your convenience:

Many of you will be more than tempted to work together with your friends on the solutions to the problems. We have no objection to this in principle, but be aware of the fine line between working together and copying. We will not tolerate copying. If you do not actually do, on your own, with full understanding, every part you are supposed to do of each exercise that you get marked, then you have copied. Remember the requirement that you fully understand your work, and our reserved right to viva you on it without notice!

When working together in an informal group of friends, pay special attention to the relative abilities of people in that group. The real point of laboratory work is not to get the answer, but to learn by doing it yourself, even if you actually get the wrong answer! If you find yourself always telling your friends the answers, then you are not much of a friend to them! You are holding them back and patronising them, but more to the point, you are undermining their learning. Equally, if you find yourself with a friend who keeps telling you the answers, don't be grateful! There is a good chance that he, or she, is simply trying to impress you. Don't be impressed!

Everybody who works in an informal group should actually do the work themselves, individually. Such working together should be restricted to discussing ideas and getting the work off the ground. Anybody who cannot actually do the work, should get help from one of the TAs¹ or supervisors. These people are experienced at helping you in such a way that you can do the work yourself, rather than just giving you the answer.

In determining the line between help and plagiarism, a good rule of thumb to stick to is this: if you get help from another student, or give help to another student, make sure that the passage of information between you is at a much higher level of abstraction than the final answer, no matter what the medium is (soft copy, typed, written, spoken, etc.). So, for example, for exercises which involve you writing some program code (and which is not supposed to be team work!), never show your code, or your detailed pseudocode, to another student: that amount of "help" is cheating. Thus, for example, if you and a few friends develop a solution to a laboratory exercise together, you are, by definition, cheating.

On the other hand, you can help each other as much as you like about things that you have not been asked to create, such as explaining the exercise question to each other, or explaining material that has been covered in lectures, e.g., what is a foreign key in the context of the relational model, etc..

Read and heed the above. To reiterate once more: we do not tolerate copying.

¹ TA stands for Teaching Assistant. A TA used to be known as a Demonstrator in the past.

General Information

Communication and Announcements

Communication between you and the lecturers and TAs is primarily via Moodle, so it is important that you check [the Moodle webpage for the course unit](#) daily, and certainly at least once or twice a week. Also, check your School and University email inboxes and check the course unit forum in Moodle, which is accessible from the course unit webpage. Of course, for matters specifically related to you alone, emailing us is fine.

It is your responsibility to keep an eye for our attempts to communicate with you. Check for any such attempt often, read them carefully and respond promptly if a response is required.

Oracle Usernames and Passwords

You will be using the Oracle DBMS² under Linux (and, in particular, its command-line interface, Oracle SQL*Plus) in lab exercises (as well as for self-learning). For that, you will need an Oracle username and password.

We assign these to you. They are then sent to you using your University student email. If you have not received yours by the start of Week 3, email both lecturers to let them know ([their emails are available in the course unit webpage](#)).

Attendance

Attendance is **mandatory** for lab sessions and very **highly recommended** for examples clinics. If you miss one of those, it is your responsibility to ensure that you are aware of any announcements made during the session and that you obtain a copy of any handouts and similar material provided during the session.

Attendance is taken for all lab session and examples clinics. The session/clinic supervisor will attempt to see everyone and take their attendance, but it is your responsibility to make sure you have had your attendance taken.

Besides the basic fact that you need to practice in order to learn, your attendance record may become important if you need an extension (see below) or to plead for special treatment in case something goes wrong for you. If you have a good attendance record, your case is likely to be far stronger than if you do not.

Timetable

The [timetable](#) for the whole of the course unit is available in the School's StudentNet.

The course unit webpage provides further information on a week-by-week basis as the course unit is being delivered.

² We will be using Oracle 11g Release 2 (11.2). The Oracle Documentation is available online at https://docs.oracle.com/cd/E11882_01/nav/portal_5.htm .

Lab Sessions/Example Clinics

All lab exercises last for one lab session only (see below for deadlines and for the policy on extensions). Lab exercises are to be undertaken individually but examples-clinic exercises can be worked on in small groups.

As usual, lab sessions take place in A-weeks (except for Week 1) and examples clinics take place in B-weeks. Note that the week after Reading Week is Week 7, a B-week.

Examples clinics are mostly meant to prepare you for the subsequent lab session/exercise. However, you can use examples clinics to ask for ongoing help and face-to-face, one-to-one feedback on any aspect of the syllabus, so do take advantage of this if you find yourself in need of more clarification or more specific help.

Sample answers are provided for the examples in examples clinics, typically on the week after they were held. However, for reasons explained above, sample answers are not provided for lab exercises, only individual feedback. Since the lab exercises are incremental, this has important consequences that you need to manage yourself.

The *Orinoco* Case Study Used in Lab Exercises

The example clinics use several, different short scenarios from different application domains. The lab exercise, however, centre around the following single, hypothetical case study, the preamble of which is as follows:

Orinoco is a medium-sized record producer that operates on niche markets and offers music on CDs, tape, and vinyl. The firm has recently bought a number of smaller recording companies and is planning to expand.

There is a great deal of information that remains available only in the brochures that *Orinoco* produces, along with an accompanying price guide. Some information is stored on electronic files on an ad hoc basis. This situation has now become very cumbersome as such files lie in different computers, leading to inconsistency and mutual incompatibility. As a result, from the point of view of the software applications that the company aims to develop to underpin its expansion plans, the *Orinoco* data assets seem incomplete and are difficult to exploit to the full (particularly due to the great heterogeneity injected by the lack of proper assimilation of the data assets that came with the recent takeovers). For example, it is particularly difficult to find music that matches a customer's specific requests unless brochures are browsed carefully, which is time-consuming and error-prone.

The management at *Orinoco* have decided to computerise the information in their brochures, to reconsider the data stored in files, to include purchase information, and to sell directly over the web. There is a need, therefore, to make it easier for software applications to:

1. Gather specific information from customers and answer questions about products
2. Check the availability of specific products
3. Give costs of products
4. Keep information about artists, albums, and tracks
5. Keep information about the back catalogues of the companies recently taken over
6. Keep contract information about artist signings etc.

You may assume that the system is stand-alone and does not have to integrate with any other. For the purposes of this course unit, other simplifying assumptions have been made (e.g., that price information is fixed and is not flexible, etc.).

Interpreting Examples, Requirements and Specifications

You **must** solve the problems **as specified** in the instructions you are given, **not as you imagine** that they would be in your personal view of what the real world is like.

Ignoring the stated requirements because, as a designer, you feel you know best what is correct or what is needed is an outright mistake.

In a professional context, it would be a case of misconduct to ignore or override customer requirements at will. In the context of this course unit, you will lose marks and end up getting yourself in difficulties (remember that each exercise builds on the preceding ones without any reset).

Of course, if the requirements are incomplete and ambiguous, which would lead, in the real world, to further rounds of elicitation requirement interactively with the customers. In this course, you should instead consult the TA supervisor or the lecturers.

In other words, do **not** go beyond or distort what is stated as the user requirements. If you feel they are unclear, check with us.

On Specifications, Marking, Feedback, Deadlines and Extensions

Specifications

The specifications and detailed instructions for lab exercises and examples clinics are available below.

In both lab and examples-clinic exercises, some of the questions are, as you would expect, harder than others. Try your best and if you cannot quite answer them, do not be disappointed: they are there for us to understand how much you can figure out for yourself using self- as well as group-learning. When it happens that you cannot answer, ask for help and the lecturers/TAs will do their best to help you work it out.

Marking and Feedback

All lab exercises are worth up to 20 marks. The distribution of marks is spelled out in the lab specifications below.

Note that some marks are set aside for good report presentation. Broadly speaking, this means: compliance with specifications, proper formatting (including the use of indentation and fixed-width fonts when presenting formal languages), readable diagrams, and clear, grammatical English in answers not involving code or diagrams.

It is important, therefore, that you develop the discipline to produce a report for assessment that conforms in terms of layout and formatting, is clearly presented in terms of language, and is clearly explained in terms of the substance of your answers.

An important aspect of good report presentation is legibility of the results printed by Oracle SQL*Plus. For example, you are shown how to control the width of columns so that lines do not overrun. In general, you are required to use the controls at your disposal in Oracle SQL*Plus to make the report abide by the good presentation criteria that have just been discussed. Do get into the habit of using the Oracle online documentation.

As you will see below, we provide you with a LaTeX template that should help you avoid the worst mistakes of presentation in preparing your report, but it is still your responsibility to ensure that it looks presentable in the sense we gave the word above.

All coursework is marked offline but the lecturers and TAs will be present in the lab to help you learn and to give you feedback as you require, including on the marking.

We are committed to returning marks and feedback as promptly as possible. However, bear in mind that the single most important reason for delays lies in the unfortunate fact that some students fail to follow the submission instructions, which creates problems for the markers. So, do your part and you will be helping us achieve our aim of being prompt with the marking.

If you feel that you need more feedback, or more information on the feedback you have received, approach us in a lab session, where we will all be ready to talk to you about it.

Lab Exercise Deadlines

Every exercise is a one-session exercise. Therefore, the time you have for completing each lab exercise is the same for all five of them: a single lab session. This means that, **the Arcade-enforced deadline for every lab exercise is the same: the end of the corresponding lab session.**

It follows from this that we assume you will arrive at the lab session with a **thorough understanding** of the lab exercise specification and, as a rule, with some (even most) of the work already done. The lab session is therefore mostly to help you with face-to-face feedback on difficulties you have encountered when trying to do the exercise in advance of attending the session.

If you fail to submit your work by the end of the session, and you qualify for an extension (see below), you must ask for one.

Extensions to Lab Exercise Deadlines

For all lab exercises in this course unit, **if** you have been in the lab working on the exercise from start to end **and if** you have worked on it in a sustained and focussed manner, you qualify for an extension.

In terms of the general School policy, an extension to a lab exercise deadline covers the period from the end of the corresponding lab session *either* to the starting-time-of-the-lab-session one (term) week after the day of the lab session *or* to the beginning of the next scheduled lab session. Whichever of these is the shorter period is then the period covered by the extension. Arcade automatically enforces this rule.

For example, when lab sessions happen every fortnight (*as is the case for COMP23111*), for a Thursday 11:00-13:00 lab session, the corresponding lab exercise's extended deadline is the following Thursday (i.e. 7 term days later) at 11:00.

It cannot be overemphasised that extensions are dependent on attendance and on your working on the exercise throughout the lab session.

So, the lab supervisor will take attendance **10 mins after the start of the session**, if you are not present then, you will not be granted an extension at the end of the session unless you can prove that there were special circumstances causing you to be late or that have made it impossible for you to attend at all.

The lab supervisor will accept requests for extensions starting **20 mins before the end of the session**. If you need to leave early, you must have a compelling reason to give to the lab supervisor.

If your attendance was taken at the start of the session **and if** you have spent all time working on the exercise, then, if you ask for an extension, the lab supervisor will grant it.

Be warned that if you spend any time in any other activity without having already submitted your work, you will get a warning. If you persevere in this practice, you will **not** be granted an extension.

Also be warned that the lab supervisor will not take it lightly if, for example, you have your attendance taken then leave the session and only return at the end in order to ask for an extension.

The lab supervisor **may** choose to give you the benefit of the doubt, but this will be a one-off, i.e., if you are given the benefit of the doubt, your name will go into a black list. The black list is consulted whenever anyone pleads forgiveness. If your name is in the list and you are found to be pleading forgiveness again, you will not get it.

If you miss the deadline, your submission incurs the usual late penalty.

No further extensions are available, so refrain from asking for one unless you have special circumstances to report.

If you do, then both follow the proper procedure described in the School's StudentNet *and* alert the TA supervisor and the lecturers **by email** at the earliest possible opportunity: don't leave any special circumstances unreported for longer than strictly necessary.

How to Work on and How to Submit Your Coursework

The submission of coursework is **through Moodle and not Arcade** as perhaps you have been used to so far. Look for the appropriate submission link in the Moodle course unit webpage. Note, though, that it is Arcade that will determine whether you have met the deadline or not. In the latter case, any penalty is applied by Arcade.

You must submit your report through Moodle, and only through Moodle, using the appropriate links that are available for this purpose in the course unit webpage. Unless you have a good reason for it, do not do anything that alters the timestamps in your personal file space, as this would invalidate their status as evidence if we were to ask you to show them to us when we audit your submission.

For all exercises, you are required to produce a report as well as generate and then submit either your code (i.e., the SQL scripts as well as the spooled out result of running those scripts in Oracle SQL*Plus) or a diagram (in both .dia and .eps version) or both. For each lab exercise, when you submit your coursework for marking, Moodle will ask you to submit two (and only two) files, as described below.

When you are asked to provide information on your first name, your last name and your student ID make sure that you do so consistently with how Campus Solutions has recorded your personal data.

COMP23111 Coursework Folders, Templates, Example Files, Scripts

Before you do anything else in terms of labs and coursework, you should copy the following .zip file into your home directory:

```
/opt/info/courses/COMP23111/COMP23111.zip
```

You should then unzip it. We advise you to have unzip it in your your home directory, so, in a shell, do

```
$ cd
$ unzip COMP23111.zip
```

The 'COMP23111' folder has the following tree structure:

```
COMP23111
├── Ex1
│   ├── COMP23111_15_Lab_Exercise_1_FirstName_LastName.tex
│   ├── LabEx1.lst (example SQL output log file)
│   ├── LabEx1.sql (example SQL script file)
│   ├── example.dia (example Dia file)
│   └── example.eps (example Dia-exported EPS file)
├── Ex2
│   ├── COMP23111_15_Lab_Exercise_2_FirstName_LastName.tex
│   ├── example.dia (example Dia file)
│   └── example.eps (example Dia-exported EPS file)
├── Ex3
│   ├── COMP23111_15_Lab_Exercise_3_FirstName_LastName.tex
│   ├── LabEx3.lst (example SQL output log file)
│   └── LabEx3.sql (example SQL script file)
├── Ex4
│   ├── COMP23111_15_Lab_Exercise_4_FirstName_LastName.tex
│   ├── LabEx4_1.lst (example first SQL output log file)
│   ├── LabEx4_1.sql (example first SQL script file)
│   ├── LabEx4_2.lst (example second SQL output log file)
│   └── LabEx4_2.sql (example second SQL script file)
└── Ex5
    ├── COMP23111_15_Lab_Exercise_5_FirstName_LastName.tex
    ├── LabEx5_1.lst (example first SQL output log file)
    ├── LabEx5_1.sql (example first SQL script file)
    ├── LabEx5_2.lst (example second SQL output log file)
    └── LabEx5_2.sql (example second SQL script file)
```

```
l... fdbchk.py (verification Python script, more on that later)
l... fdbgen.py (generation Python script, more on that later)
```

Here's an explanation of what these folders and files contain and mean:

1. The root folder is called 'COMP23111'. If you have unzipped the file and such a folder already existed, normally the compressed files are added to the folder, no preexisting files are deleted and any overwriting is preceded by an interaction where you have a choice as to what to do with the preexisting files.
2. There is one sub-folder per exercise, so five subfolders in total. Each is named 'Ex<N>' where <N> is the one-digit number of the exercise.
3. The files in each folder more or less exemplify what you should need to submit. **They are just examples.** Some exercises have paired (.dia, .eps) diagram files, some don't. Some have both diagrams and paired script (.sql) and output log (.lst) files. Some exercises don't have script/log files, some do. For those that do, some have one pair of script/log files, others have two.
4. The .tex files are LaTeX templates. The lab exercises require you to generate a report using LaTeX and outputting PDF, which is then part of your submission (see below). You will need to add your answers when required, and edit the placeholders for scripts, output logs and EPS files corresponding to diagrams, as required in each exercise.
5. The .sql files are just examples for the placeholders in the .tex file template. Such files contain SQL*Plus commands and SQL statements that can be run with the START SQL*Plus command. When you have done all your coursework, as required or not by an exercise, you'll have composed your very own SQL script file. You will then need to place it in the correct Ex<N> directory, and ensure its name is referenced in the appropriate way in the .tex file, so that your SQL script is included in your final report to be generated when LaTeX is run.
6. The .lst files are also just examples for the placeholders in the .tex file template. Such files are generated when you run your SQL script file in SQL*Plus under the control of the SP00L command, which basically results in the redirection of formatted input and output onto the .lst file. When you have done all your coursework, as required or not by an exercise, you'll have obtained a corresponding .lst output log. You will then need to place it in the correct Ex<N> directory, and ensure its name is referenced in the appropriate way in the .tex file so that the output log of your work is included in your final report to be generated when LaTeX is run.
7. The .dia files are, again, just examples of the kinds of file that result from using the Dia tool. When you have done your coursework, as required or not by an exercise, you'll have generated your own diagram and saved it using the .dia suffix. You will then need to place it in the correct Ex<N> directory, **but**, unlike all other kinds of file, you don't need to worry about the name of a .dia file being referenced in the .tex file since no .dia file is included in the report, instead their EPS form is (see the next item).
8. The .eps files are, once more, just examples of the kinds of file that result from exporting diagrams from within the Dia tool. When you have done your coursework, as required or not by an exercise, you'll have exported any diagram into an equivalent EPS file using the Dia tool using the following sequence of menu interactions:

```
File > Export [select 'Encapsulated Postscript' format]
```

There are two versions of .eps, either of them will do. You will then need to place the .eps in the correct Ex<N> directory, and ensure its name is referenced in the appropriate way in the .tex file so that the EPS version of your diagrams resulting from your work are included in your final report to be generated when LaTeX is run.

9. fdbchk.py and fdbgen.py are Python scripts we provide you with. They are explained later.

As always, talk to the TAs and the lecturers if you have any doubts.

Personalising the Files We Have Given You

After you have copied and unzipped the folder structure we provide you with, before you start doing any actual work on Exercise <N>, you should:

1. Go into COMP2311/Ex<N>, i.e., the folder correspond to the lab exercise number in hand.
2. Decide what names you're going to assign to the .dia, .eps, .sql and .lst files as required by Exercise <N>.
3. Rename the .tex file replacing 'FirstName' with your first name and 'LastName' with your last name.
4. Edit the .tex file so that it uses the right file names you have decided to use. **This is very important.**

Note that you can use differently-named scratch (i.e., temporary) files for you to experiment, debug, etc., and rename/copy/tidy-up these for inclusion in your report. What is crucial is that you ensure that the .tex files are edited to include the correct (final, best-attempt) files!

As always, talk to the TAs and the lecturers if you have any doubts.

Doing the Actual Work on the Exercise

If Exercise <N> involves producing a diagram using Dia (not all do), then make sure that you generate a .dia file with the name you have chosen and that you export it as EPS with the name you have chosen for it to be included in the .tex file.

If the exercise involves using SQL*Plus (not all do), then we suggest that most of the time you mostly work using a text editor to prepare SQL*Plus commands and SQL statements, as appropriate, and then, through copying-and-pasting try them out in the SQL*Plus command line interface.

In the beginning, you will make mistakes, of course. You then make changes in the text editor and try again, in very much a debugging frame-of-mind.

Once you got it right, for each such piece of work in the lab specifications, you will have completed a step in the SQL script that we ask you to submit, so you may wish to save/append it to an .sql file with the name you have chosen for it to be included in the .tex file. Some exercises require more than one script file, and some require none.

After the SQL script file/s is/are ready, then you can run it/them under SPOOLing to produce the .lst file/s that will get submitted too. These, of course, must have the name you have chosen for it/them to be included in the .tex file.

Now, the above process is just an idea for you to consider. In practice, you should use your common sense and knowledge of both computing and the available tools to find out what, in the detail of it, works best for you

Once you have all the files that are needed, then you need to prepare for submission in Moodle.

As always, talk to the TAs and the lecturers if you have any doubts.

What Each Submission Consists Of

For each lab exercise, when you submit your coursework for marking, Moodle will ask you to submit **two** (and **only two**) files, as follows:

- I. A report, which is what we will mark. This must be written in LaTeX, using the template we provide you with (see above) to produce a PDF document. You *don't* submit the .tex* file separately, rather you **must** submit the PDF file. Keep reading to learn how to generate this PDF report file.

- II. A .zip file that is a compressed archive of the folder where you will organise the outcome of your work on a given lab exercise and that we will need for reference purposes in marking the report and for auditing malpractice, if needed. Again, keep reading to learn how to generate this file.

As always, talk to the TAs and the lecturers if you have any doubts.

Preparing the Files for Generating the Submission

Here are some instructions on how to go about preparing the files for the generation of the required files for submission:

1. In the COMP23111/Ex<N> folder, locate the template file (but you should have renamed it already, so, if you haven't, do so now):

COMP23111_15_Lab_Exercise_<N>_FirstName_LastName.tex

2. Open this file using your favourite text editor in Linux.
3. Type your answers where and when required by the Exercise <N>. The template for the exercise provides some structure and some tips.
4. If required by Exercise <N>, ensure that all .sql script files and all the .lst files containing formatted output from running the scripts are in the folder and are correctly named.
5. Then, look for lines similar to these

`\includesql{LabEx<N>.sql}`

6. and these

`\includesql{LabEx<N>.lst}`

and change it/them to reference the file/s you have prepared.

7. Then, look for lines similar to these

`\centerline{\includegraphics[width=0.75\textwidth]{example}}`

and change it/them to reference the filename of the .eps file/s you have prepared (but do not include the .eps suffix in the LaTeX line above). Notice the reduction factor (0.75) in `\textwidth`: if the figure doesn't print, or doesn't look, right, i.e., legible and clear, try adjusting the reduction factor in the line above.

8. Save the .tex file.

We provide you with two scripts (recall the description above of the directory structure of your COMP23111 folder and see below) that will help you to generate the files to be submitted to Moodle.

It is very important that you understand that:

- The .py scripts have only been designed to run in Linux: there is **no** guarantee that they will work under another OS.
- It is **your** responsibility, always, to ensure that both the report and zip files are correct.
- The scripts **does not** sprinkle magic dust on your prepared files and make them right.
- The scripts **cannot** possibly detect all the mistakes you may possibly have made.
- The scripts are provided with a view to help reducing the chores, **not** to give you hard and fast guarantees that all is well with your submission.
- Be warned then: **do not** assume that using the scripts guarantees that everything is OK.
- It is for **you** to ensure the submitted files are the ones you want to submit.

As always, talk to the TAs and the lecturers if you have any doubts.

Verifying You Have All the Files for Generating the Submission

The `fdbchk.py` script **only checks** that (roughly) valid parameters have been passed and that directories and files are (roughly) as one would expect them to be.

To use it, ensure you are in your COMP23111 folder (**not** the `Ex<N>` folder), then run the script as exemplified here (where we assume that your COMP23111 folder is, as recommended, in your home directory):

```
$ cd ~/COMP23111
$ python fdbchk.py <exercNumber> <firstName> <lastName> <studentID>
```

with all the (self-explanatory) parameters (in angle brackets) being required.

Note that, among other things, the script cannot check that the filenames you have typed into the `.tex` file for inclusion in your report do exist.

The script only checks that there exist files with the expected suffixes for the exercise number passed as parameter and in the right place.

This means that you can still get an error when LaTeX is run (see below).

As always, talk to the TAs and the lecturers if you have any doubts.

Generating the Files for Submission

The `fdbgen.py` script replicates what `fdbchk.py` does **and also** tries to generate the report file and the zip file for uploading to Moodle.

To use it, ensure you are in your COMP23111 folder (**not** the `Ex<N>` folder), then run the script as exemplified here (where we assume that your COMP23111 folder is, as recommended, in your home directory):

```
$ cd ~/COMP23111
$ python fdbgen.py <exercNumber> <firstName> <lastName> <studentID>
```

with all the (self-explanatory) parameters (in angle brackets) being required.

The script does the verification and if it is successful, runs `pdflatex` with the appropriate parameters in order to generate the report file in PDF format, then runs `zip` to generate the required zipped file.

If all goes well, these two files will have been placed in the your COMP23111 folder (**not** the `Ex<N>` folder) from where you can submit them to Moodle (see below).

As always, talk to the TAs and the lecturers if you have any doubts.

Submitting Your Coursework

To submit your work, go to the Moodle webpage for the course unit, and click on the submission link for the appropriate lab exercise.

You will be required to submit two files:

1. one is the report (the PDF document),
2. the other is the archived folder (the zip file within which we can find the SQL scripts, logs and diagrams we need).

As always, talk to the TAs and the lecturers if you have any doubts.

Examples Clinic 1

Understanding the Relational Paradigm Specification

The goal of this examples clinic is to help you improve your understanding of the relational model, the relational algebra and basic SQL. It aims to prepare you for Lab Exercise 1, in which you get the first opportunity to use the Oracle DBMS.

DURATION: 50 minutes examples clinic time.
DESCRIPTION: The examples clinic consists of questions on the relational model and SQL. Work through the questions, show them to the lecturers and TAs in order to obtain feedback.

TASKS

T1. Consider the tables in Figure EC-01. Identify some of the relationships that exist in the tables below (and propose a suitable name for them). Also, explain, in terms of the relational data model, the reasoning behind your identification of these relationships.

STUDENT

| Name | Student_number | Class | Major |
|-------|----------------|-------|-------|
| Smith | 17 | 1 | CS |
| Brown | 8 | 2 | CS |

COURSE

| Course_name | Course_number | Credit_hours | Department |
|---------------------------|---------------|--------------|------------|
| Intro to Computer Science | CS1310 | 4 | CS |
| Data Structures | CS3320 | 4 | CS |
| Discrete Mathematics | MATH2410 | 3 | MATH |
| Database | CS3380 | 3 | CS |

SECTION

| Section_identifier | Course_number | Semester | Year | Instructor |
|--------------------|---------------|----------|------|------------|
| 85 | MATH2410 | Fall | 07 | King |
| 92 | CS1310 | Fall | 07 | Anderson |
| 102 | CS3320 | Spring | 08 | Knuth |
| 112 | MATH2410 | Fall | 08 | Chang |
| 119 | CS1310 | Fall | 08 | Anderson |
| 135 | CS3380 | Fall | 08 | Stone |

GRADE_REPORT

| Student_number | Section_identifier | Grade |
|----------------|--------------------|-------|
| 17 | 112 | B |
| 17 | 119 | C |
| 8 | 85 | A |
| 8 | 92 | A |
| 8 | 102 | B |
| 8 | 135 | A |

PREREQUISITE

| Course_number | Prerequisite_number |
|---------------|---------------------|
| CS3380 | CS3320 |
| CS3380 | MATH2410 |
| CS3320 | CS1310 |

A database that stores student and course information.

Figure EC-01: Tables on Students and Courses

T2. Consider the UNIVERSITY relational schema in diagrammatic form shown in Figure EC-02.

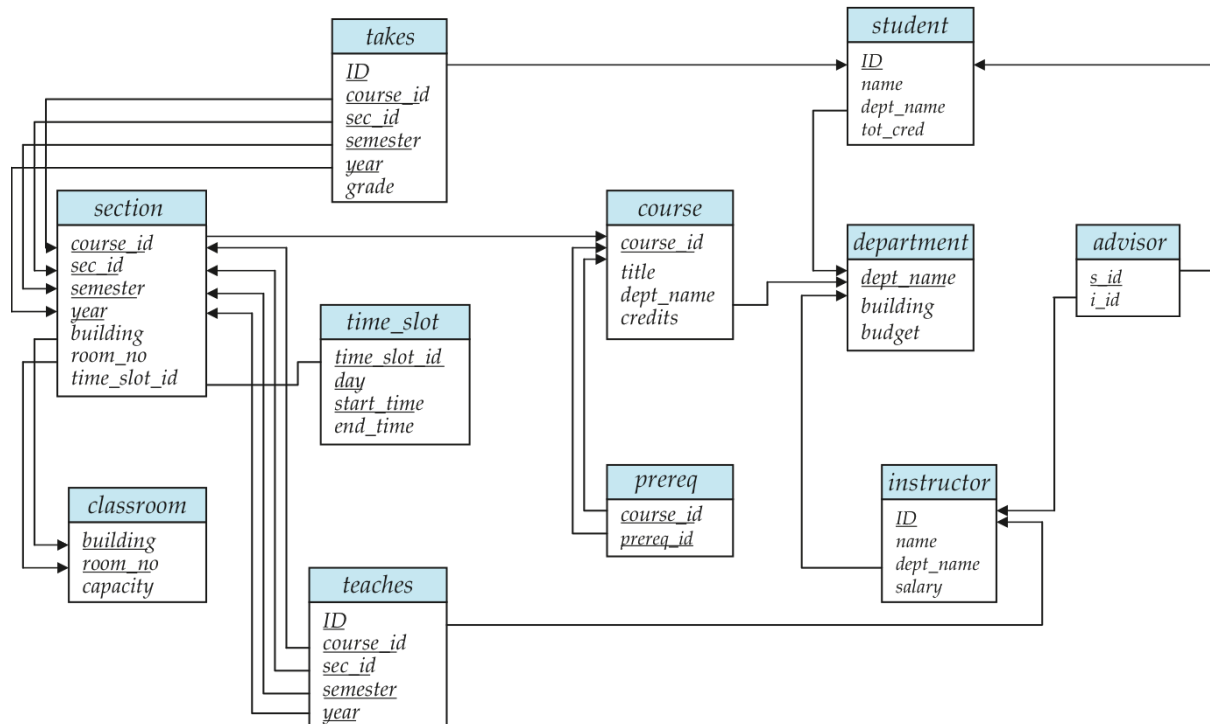


Figure EC-02: UNIVERSITY Relational Schema in Diagrammatic Form

This diagrammatic form denotes a relation with a rectangle, the relation name (e.g., takes) appears shaded/light-blue, the attributes are below the relation name (e.g., ID, course_id, sec_id, semester, year, grade), with the attributes that participate in the primary key underlined. An arrow from an attribute a' in relation S to an attribute a in relation R denotes that the latter occurs as a foreign key in the former. For example, there is an arrow from takes.ID to student.ID so you can read the arrow as 'takes.ID references student.ID'.

Now, consider the advisor relation, with s_id as its primary key. Suppose a student can have more than one advisor. Explain why would s_id not be an appropriate primary key of the advisor relation any more? What would an appropriate primary key for the relation be in that case?

T3. In the context of the UNIVERSITY schema given above for **Q3**, write the following queries in SQL:

- I. Find the names of all students who have taken at least one Computer Science course; make sure there are no duplicate names in the result.
- II. Find the IDs and names of all students who have not taken any course offering before Spring 2009.
- III. For each department, find the maximum salary of instructors in that department. You may assume that every department has at least one instructor.
- IV. Find the lowest, across all departments, of the per-department maximum salary computed by the preceding query.

(Note that you may need more SQL constructs than were discussed in the lectures. This is a good reminder that you need to read the mandatory readings.)

T4. (Note that we may not have yet covered relational algebra in the lectures at this point. If so, you may prefer to leave this final question to answer later, when you have caught up on the mandatory readings.)

Consider the following relational schema:

```
employee (person name, street, city)
works (person name, company name, salary)
company (company name, city)
```

Work out what are the primary and foreign keys, then, for each of the following data requirements, give a relational algebraic expression that captures them:

- I. Find the names of all employees who work for "First Bank Corporation".
- II. Find the names and cities of residence of all employees who work for "First Bank Corporation".
- III. Find the names, street address, and cities of residence of all employees who work for "First Bank Corporation" and earn more than \$10,000.

Acknowledgements

The tasks in this practical session are minor adaptations of material authored and made available to instructors by (a) A. Silberschatz, H. F. Korth, and S. Sudarshan to accompany their textbook Database System Concepts, 6th Edition McGraw-Hill, 2006, 978-0-07-352332-3, (b) H.P Garcia-Molina, J.D. Ullman, and J. Widom to accompany their textbook Database Systems: The Complete Book, 2nd Edition Pearson Prentice Hall, 2009, 978-0-13-135428-9, and (c) Ramez Elmasri and Shamkant B. Navathe to accompany their textbook Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8 Copyright © 2011 Pearson Education, Inc. Copyright remains with them, whom we thank. Any errors are our responsibility.

Lab Exercise 01

First Steps with SQL*Plus and Dia for (E)ER Modelling Specification

Overview

The goal of this lab exercise is to provide you with an opportunity to interact with the Oracle DBMS using Oracle SQL*Plus, the command-line interface provided by Oracle, and with the Dia program for drawing structured diagrams.

You are given scripts that allow you to create/drop/recreate some tables from the *Orinoco* case-study database and populate them with tuples. Using SQL*Plus, this then allows you to explore how SQL is implemented by Oracle.

You will gain a basic understanding of the data dictionary component of the Oracle DBMS and will take the initial steps in using the data definition and manipulation aspects of SQL, as implemented in Oracle, as well as basic query forms. You will also gain an understanding of how referential integrity is implemented.

Finally, you will be asked to reproduce an (E)ER diagram using Dia, just to ensure that you become familiar with its support for (E)ER modelling.

For attendance, marks available, time available, submission requirements, deadline, and policy on extensions, see the initial sections of this manual.

Introduction

This exercise has two parts: a tutorial followed by an exercise.

You are given information on how to run some scripts that will result in your being the owner of the following populated tables from the *Orinoco* database.

| | |
|---------------|--|
| Manager | (<u>manager_ID</u> , name) |
| ManPhone | (<u>manager_ID</u> [fk:Manager.manager_ID], telephone) |
| Artist | (<u>artistic_name</u> , genre, managedBy [fk:Manager.manager_ID]) |
| Album | (<u>album_ID</u> , title, createdBy [fk:Artist.artistic_name]) |
| MasterTrack | (<u>track_ID</u> , working_title, working_length, recordedBy [fk:Artist.artistic_name], editedBy [fk:SoundEngineer.sound_eng_id]) |
| SoundEngineer | (<u>sound_eng_ID</u> , name) |
| ContractInfo | (hasContract [fk:Artist.artistic_name], <u>date_from</u> , <u>date_to</u> , duration [derived as (date_to - date_from)]) |
| FinishedTrack | (<u>originatesFrom</u> [fk:MasterTrack.track_ID], <u>version</u> , released_title, final_length) |

Since these are some, but not all the tables in the *Orinoco* database, some of the overall semantics is missing, but broadly speaking, the database as a whole is intended to meet, among other, the following requirements:

- Each artist has an artistic name, works within a musical genre and is managed by a manager who has an ID, a name and can have several contact telephone numbers.
- Artists also have a contract with *Orinoco*, for which the following information is recorded: the period of the contract in terms of the date from which, as well as the date up to which, the contract holds. The duration of the contract in days is also stored.
- An artist records a number of master tracks, each of which has an ID, a working title, a working length. Information on the artist who recorded it and the sound engineer who edited it are also kept.

- A finished track originates from a master track and has a version number (as there may be different versions of it), a released title and a final length.
- A sound engineer has an ID and a name.
- An album groups together finished tracks into a sequence (i.e., a play list). An album has an ID, a title. Information is kept about the artist who created it. In this initial model, we are not including information that links an album with the finished tracks it contains. We will do that in later exercises, when we will also store information as to whether an album can be distributed as a CD, on vinyl, or as a cassette tape.

Requirements

From a machine running Linux, you will use Oracle SQL*Plus to interact with the Oracle DBMS. You will also use the Dia diagramming tool, and LaTeX which are also available in Linux machines.

Instructions

Firstly, work your way through Part 1, i.e., the SQL*Plus tutorial below.

Note what you need to submit in your report and learn how to save the information as you go. There are suggestions for that in the tutorial itself. The overall goal is for you to learn about the tables that you own, their definitions (i.e., the columns, and their types, that characterise each table) and the data that is stored in them (i.e., the rows that each table currently contains). You will also learn how to find out about the primary, and the foreign keys (along with their target primary keys) for the tables you own. Another important goal is for you to learn how to control the way SQL*Plus displays, and prints out to file, the results of command execution.

At the end of Part 1, i.e., the tutorial, you will be in a position to provide answers to items 1 to 3 below.

Secondly, in Part 2, use Dia to replicate the diagram shown in Figure 6 below. This will allow you to provide an answer to item 4 below.

Report Contents

Your Report must contain:

- | | |
|--|-----------|
| 1. A listing of the structure of <u>any</u> 3 tables in the Orinoco database | [3 marks] |
| 2. A listing of the primary and foreign keys of <u>all</u> tables | [6 marks] |
| 3. A listing of the data in <u>any</u> 3 tables | [3 marks] |
| 4. A Dia diagram as described below | [6 marks] |

Additional marks are available as follows:

- | | |
|---|-----------|
| 5. Good report presentation (see the Lab Sessions/Example Clinics section above) | [2 marks] |
|---|-----------|

Part 1: SQL*Plus TUTORIAL

You must use Linux.

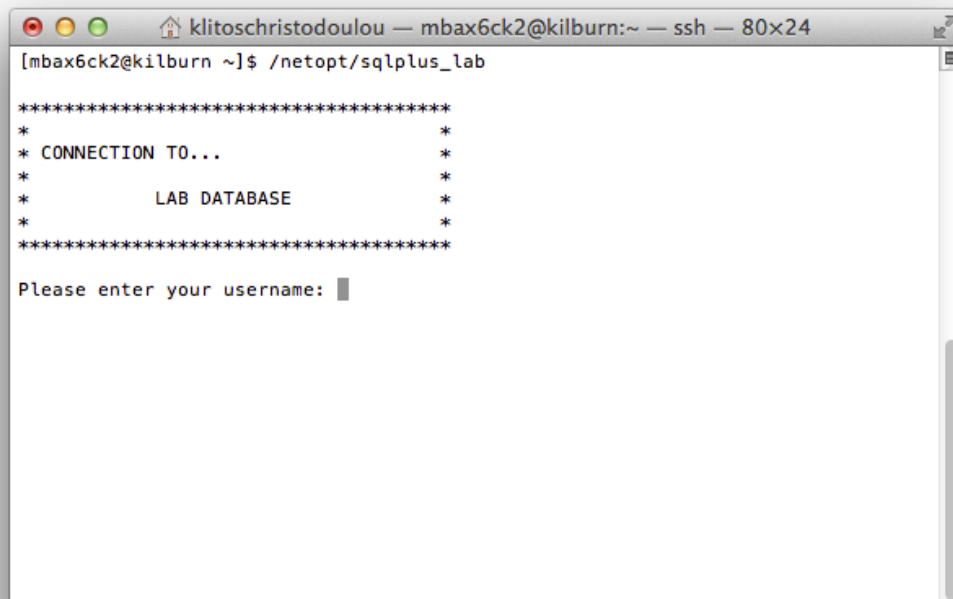
Starting SQL*Plus

Full information with examples for SQL*PLUS commands can be found in the online [Oracle documentation](#). In particular, the [SQL*Plus User's Guide and Reference](#) is also online.

You may also benefit from the e-books on SQL available online through the The University of Manchester Library (TUoML). Two that are useful for you to know about and consult are:

- **Beginning Oracle SQL**, Lex De Haan, Tim Gorman, Inger Jørgensen, Melanie Caffrey, Apress, 2014. [Online Version](#)
- **Pro Oracle SQL**, Karen Morton, Kerry Osborne, Robyn Sands, Riyaj Shamsudeen, Jared Still, Apress, 2013. [Online Version](#).

Follow the example in Figure LX-01 to launch SQL*Plus. You should use the user name and password sent to you. If you find a problem with the path even though you are running a Linux shell in a SCS UG lab machine, let the lecturers or TAs know. (If the full path does not work, try typing just `sqlplus_lab`.)



```

klitoschristodoulou — mbax6ck2@kilburn:~ — ssh — 80x24
[mbax6ck2@kilburn ~]$ /netopt/sqlplus_lab

*****
*                                     *
*  CONNECTION TO...                  *
*                                     *
*          LAB DATABASE              *
*                                     *
*****

Please enter your username: 
```

Figure LX-01: Logging in to Oracle SQL*Plus

When you see the SQL*Plus prompt, you can simply type in a SQL*Plus command. (Try `HELP TOPIC`, and you will have something to explore.)

We will aim to be precise in differentiating SQL*Plus *commands* (which are specific to Oracle) and SQL *statements* (which are, by and large, standard across many DBMSs). Also, we often refer to a SQL *query*, and ideally this would only apply to SQL statements that do not cause the database to change, but the use of the more precise word as a synonym for statement is very widespread and we'll probably fail to be consistent in our usage.

You can also paste commands/statements after typing and copying them in a text editor. (This is **highly recommended**, because retyping is tedious and SQL*Plus is not always as usable as, say, a modern Unix shell. For example, it doesn't support completion.)

Separate words with spaces or tabs (choose one approach and use it consistently). Commands and statements can appear in upper or lower case (again, choose one approach and use it consistently), but literals (as we will see) are case-sensitive.

There are also special SQL*Plus commands for formatting, setting options and editing and storing SQL commands (see the documentation for details and explore with the `HELP` command.). It is highly recommended that you explore.

Spooling Work

With the `SP00L <filename>` command, you can log all commands/statements plus their result into the text file `<filename>`. When you want to stop logging, use `SP00L OFF`.

Running Scripts

You can run a textual sequence of SQL statements stored in a file as a script (using the *.sql suffix). In SQL*Plus, the `START <filename>` command executes the SQL commands in <filename>.

We have provided you with three files with SQL commands to give you something to play with. Their full paths are:

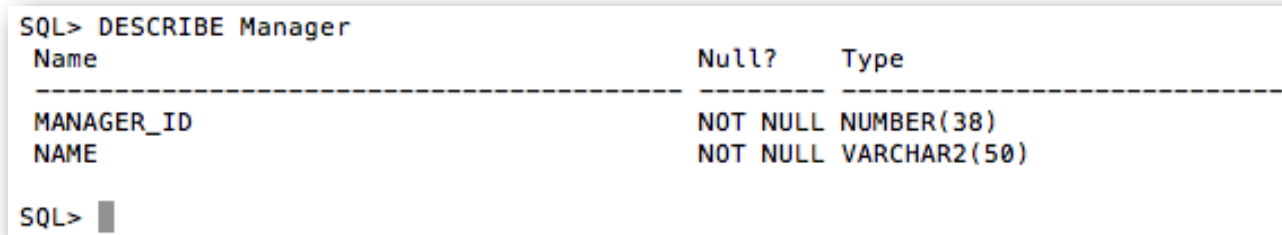
```
/opt/info/courses/COMP23111/create-Orinoco-initial-tables.sql
/opt/info/courses/COMP23111/drop-Orinoco-initial-tables.sql
/opt/info/courses/COMP23111/recreate-Orinoco-initial-tables.sql
```

The first one creates the tables mentioned above in the **Introduction** section and populates them with some data. The second deletes those tables (and their content). The third is essentially the append of the second to the first, and therefore returns the tables to the initial state by first deleting them, then recreating and repopulating them from scratch.

Use the SQL*Plus `START` command to execute the commands in the files above as you need. The first step is, of course, to create the tables. This produces lots of output. Just browse through it to make sure that no errors occurred. If so, then continue exploring.

Listing a Table Definition

One of the tables you now own is `Manager`. (Later in this tutorial, you will learn how to find all the tables you own.) Use the SQL*Plus command `DESCRIBE` to find out the definition of `Manager`, as it is held in the Oracle Data Dictionary. You should see what is shown in Figure LX-02, i.e., the column names and the types of the values stored (plus an indication as to whether the columns can store null values).



| Name | Null? | Type |
|------------|----------|--------------|
| MANAGER_ID | NOT NULL | NUMBER(38) |
| NAME | NOT NULL | VARCHAR2(50) |

Figure LX-02: Example Description of a Table

Make a mental note of the fact that SQL*Plus is printing lots of white space between value columns, the reason being that, in the data dictionary, the column where the `Name` attribute is stored is defined to be quite wide. You will learn to control what is printed, and you will need to exercise judgement as to readability and good layout in your lab exercise reports if you are to merit the full marks for good report presentation. So, be attentive and make time to explore and learn on your own.

Ending a SQL statement

You generally will use a semicolon (;) to end a SQL statement. Hitting the enter/return key causes the statement to be executed. A statement can span many lines: so, it is the semicolon that denotes its end and it is needed, therefore.

Halting the execution of a SQL statement

If execution hangs (which should be very very very rare), use `Ctrl-C` to try and halt an execution.

Selecting all the rows and all the columns in a relation.

To select all columns and all rows in a table, instantiate the SQL template

```
SELECT * FROM <table_name>;
```

The '*' stands for 'all columns'. Try it with `Manager` as the table name.

Selecting some of columns from some of the rows.

To select some of the columns only, you list their names in the `SELECT` clause.

You can also select some of the rows only. If you also add a `WHERE` clause that specifies a predicate (i.e., a Boolean-valued expression) then a tuple must satisfy that predicate in order to be part of the output.

The full template is

```
SELECT <col_name1>, ..., <col_name_n>
FROM   <table_name>
WHERE  <predicate>;
```

Try finding out the columns that comprise the `Artist` table. Then look at the entire content of the table.

Now, try finding just the names of the artists that satisfy the following predicate `Genre='Dance'`.

The single quotes to enter a string literal are important. Note also that while SQL and SQL*Plus are not, in general, case-sensitive, when you're passing literals in a statement (such as `'Dance'`) case does matter.

When using quotation marks for literals, be attentive to situations in which your text editor is too clever and generates an illegal character instead. Also, be careful when you copy and paste from PDF files; sometimes the character that gets pasted is not what it looks like.

When you've tried the above, this is the result you should see what is shown in Figure LX-03.



```
ARTIST_NAME
-----
Scandal
JK Row1
PJ Blox

SQL> █
```

Figure LX-03: Example Output of a `SELECT-FROM-WHERE` query

After displaying the results, SQL*Plus displays the command prompt again.

Ordering the results

You can use the additional template clause (after the `WHERE` clause, or the `FROM` clause if no `WHERE` clause is given)

```
ORDER BY <col_name1>, ..., <col_name_n>
```

to sort the returned tuples in ascending, cascading order. Try sorting the results of your previous SQL statement by artistic name. Use the same statement as above but now add the appropriate ORDER BY clause.

Removing duplicates

Sometimes the results contain duplicates, but you can eliminate them, if you wish, by adding the keyword DISTINCT immediately after the SELECT keyword.

Creating and modifying SQL statements

Typically you will enter an SQL statement across several lines, execute it and then want to alter it (as we all make mistakes, and more so when we are learning) and try again.

It is highly recommended that you create and modify your statements using a text editor (avoid word processors, as they might tamper with, e.g., quote characters).

Type your statements into the text editor, then copy and-paste them into the SQL*Plus prompt. The line numbers may flash up and make the query look a bit of a mess, but they do not affect its execution. In this way, you can also maintain a history of different queries you have tried and edit/retry complex queries until you get the syntax right.

The most common problems are likely to be errors in the syntax that are difficult to identify from the error messages. You are strongly advised to break your queries into separate lines, and build a complex SQL statement up slowly, bit by bit. This will also help you gain a feel for compositionality, i.e., how a query, for example, can be made of subqueries (i.e., nested queries).

You can also make use of the arrow keys³ to scroll up and down the command/statement history, but although this is often quicker, it is not as good for documenting your work as using the editor, so, use both methods and get the best of both worlds.

Dealing with errors

SQL*Plus has syntax rules for commands and so does SQL for statements in the language. If you break a syntax rule, you will get an error and the command/statement will not execute.

Most error messages will, in the usual way, give you a clue as to what you may have done wrong. If you get an error message that you cannot understand, learn how to use the online Oracle documentation (see the link at the start of this specification). If this is not helpful, ask the lecturers or TAs for help.

Try, for example, misspelling Manager as Minager on purpose and ask SQL*Plus to DESCRIBE it. You'll get an error. Then, in a web browser, use the search field in the online Oracle documentation webpage (see the link above) to try and find out more about the error by typing in the error code you got. Give or take some inessential differences, you should find in the documentation an explanation like the one shown in Figure LX-04.

ORA-04043: object *string* does not exist

Cause: An object name was specified that was not recognized by the system. There are several possible causes:

- An invalid name for a table, view, sequence, procedure, function, package, or package body was entered. Since the system could not recognize the invalid name, it responded with the message that the named object does not exist.
- An attempt was made to rename an index or a cluster, or some other object that cannot be renamed.

Action: Check the spelling of the named object and rerun the code. (Valid names of tables, views, functions, etc. can be listed by querying the data dictionary.)

Figure LX-04: Example Explanation of an Error Message in Oracle's Online Documentation

³ This is not standard SQL*Plus functionality. The server we use has been enhanced with this capability.

Saving your queries, and the results of your queries, for inclusion in reports

For producing the final PDF document required for submission, Do study the section of this manual where detailed instructions are given for this.

For solutions to the tasks from each lab exercise you must incrementally create an SQL script file(s) in a text editor (for example, `ned`, or `vi`, or `vim`, or `emacs`), run the script(s) and, when you're satisfied with it, run it complete, making sure that you spool the result(s) into a separate `*.lst` file(s) for inclusion in your report.

Getting the results to look better formatted

When the results are returned in a format that makes them difficult for you to even read them, you must use the various commands that SQL*Plus offers to adjust display characteristics. To do so, explore them with `HELP INDEX`.

Use `HELP` to learn about the various things you can `SET`. Also look at the `COLUMN` command. Use `HELP` to learn how you can specify how columns get displayed, and so on. Finally, learn about the `CLEAR` command too. If needed, go beyond the help facility in SQL*Plus and search the Oracle online documentation.

Sensible use of the above can mean the difference between your reports being readable or not, and therefore meriting the good presentation marks or not.

The Oracle Data Dictionary

Like most DBMSs, Oracle has a data dictionary containing all the management information on the objects (i.e., tables, views, indexes, etc) in the database. You are not given authorisation to alter this dictionary directly, but you can look at it: this information is held in the form of tables too, so the usual SQL statements also work for the data dictionary tables.

The Oracle Database Administrator's Guide (see the online documentation) has copious amounts of information on that, as you would expect, but you shouldn't need to delve into it for the purposes of this course.

Listing the tables you own

The data dictionary table (well, it is actually a view) `user_tables` contains information on all the tables the user (i.e., you) own.

You can try asking SQL*Plus to `DESCRIBE` it, but you'll find that a lot of very technical information about the data dictionary comes flooding out. For the moment, the most interesting column is `table_name`. Since `user_tables` is just a table, you can access the information stored in it using SQL statements. For example, try this SQL statement:

```
SELECT table_name FROM user_tables;
```

Notice that the data strings in the dictionary are in upper case, so any `WHERE` clause that uses literals will have to be aware of this. This is so irrespective of what case was used in the strings that appeared in the SQL statement that created, altered or populated the database.

Use `DESCRIBE` to have a look at `user_tables` and see just how much metadata (i.e., data about data) an Oracle table has.

Integrity constraints in the data dictionary

The `DESCRIBE` command accesses information in the Oracle data dictionary. Notice that this does not tell us anything about keys or any integrity constraints defined on the table. We have to actually query the data dictionary to find out such information.

To find out the integrity constraints on, say, the Manager and the ManPhone, we need to look at two dictionary views:

- user_constraints for constraints on the tables
- user_cons_columns for constraints on the columns in the tables.

Starting with the tables

```
SELECT constraint_name, constraint_type,
       table_name, r_constraint_name,
       delete_rule, status
FROM   user_constraints
WHERE  table_name = 'MANAGER' or table_name = 'MANPHONE';
```

You may find that it prints something difficult to read. This is the time to set a wider line width. Find out how to use the SET command to extend the line width to 250, say, and then enlarge your window enough to accommodate a line as wide as that. Then, try again. You should see something like shown in Figure LX-05.

| CONSTRAINT_NAME | C | TABLE_NAME | R_CONSTRAINT_NAME | DELETE_RU | STATUS |
|-----------------|---|------------|-------------------|-----------|---------|
| SYS_C00250434 | C | MANAGER | | | ENABLED |
| M_ID_PK | P | MANAGER | | | ENABLED |
| MP_ID_PK | P | MANPHONE | | | ENABLED |
| MP_ID_FK | R | MANPHONE | M_ID_PK | NO ACTION | ENABLED |

SQL> █

Figure LX-05: Example Constraints on Tables

We need to go through this result. First of all, notice that the literal values in the dictionary table are all in upper case. This means that when they are queried for, they must be passed as upper case literals.

The constraint type (i.e., the values in the the column named C) can be:

- P Primary key, i.e., the identity integrity constraint
- R Foreign key, i.e., the referential integrity constraint
- C Constraint, i.e., a domain integrity constraint in the form of an SQL expression such as NOT NULL or CHECK IN ('cis','cs')

Constraints named SYS_xxxxxxxx are created by Oracle for unnamed constraints (and will, most likely, be different in your case from what you see in Figure LX-05). Rows 2 and 3 state that the Manager and ManPhone have a primary key defined and that it is enabled. The constraints have been named by the owner (well, the script we gave you did that) with specific constraint names. Row 4 states that ManPhone has a foreign key pointing to the primary key of Manager.

The admissible values in the DELETE_RULE column for referential integrity constraints can be NO ACTION or CASCADE (which we will learn about in due course). Finally, the STATUS of the constraint can be ENABLED or not.

Now, we can find all type C constraints using:

```
SELECT constraint_name, table_name, search_condition
FROM   user_constraints
WHERE  constraint_type = 'C' AND table_name NOT LIKE '%BIN%';
```

(Disregard, for now, the condition `table_name NOT LIKE '%BIN%'`. It is being used here to filter out the many constraints that Oracle generates of its own accord. If you are curious and want to have a look, just rerun the query without the conjunct that filters it. You can ignore the system generated information: the data dictionary is a repository of both user-level and system-level information, so querying it can be, as here, a bit messy.)

The tuples in the result all have, of course, a C-type constraint. The C stands for 'check'. In this case, to check that particular attributes are NOT NULL.

The `user_constraints` table/view just tells us on which tables there are constraints, but not the columns that the constraints act upon. To see this we have to look at the `user_cons_columns` table/view.

Use the following query now:

```
SELECT constraint_name, table_name, column_name, position
FROM   user_cons_columns
WHERE  table_name = 'MANAGER' or table_name = 'MANPHONE';
```

Again, this will likely print in a difficult-to-read way. You will find that the column `COLUMN_NAME` is too wide. It is time to tinker with the way it is displayed. Recall that we suggested that you looked into what the `COLUMN` command in SQL*Plus does. You should have learned to how use it, so set the column `COLUMN_NAME` to a narrower width, as follows:

```
COLUMN column_name FORMAT a20
```

and try again.

We can now see that the constraint `MP_ID_PK` refers to column `Telephone` in the table `ManPhone`. The name suggests that this is a primary key constraint. It also tells us that `MP_ID_FK` in the `ManPhone` table refers to the column with name `MANAGER_ID`, suggesting that this is a foreign key constraint referring to the `Manager` table's primary key (identified by `MP_ID_PK`). We can infer much of this information from the constraint names if we always make an effort to give constraints sensible names. (Note that `POSITION` refers to the position of a column in a composite key.)

Reloading the tables if they have become corrupted

If you've managed to delete or otherwise damage the data in the tables, use the recreation script mentioned above.

This is the end of the tutorial, though you are expected to explore much much further on your own, at your own time.

For now, work on items 1-3 of your report. Remember to save and organise your input and output and to incorporate it to the report you must submit. Then, move on to Part 2.

Part 2: Getting Started with Dia

[Dia](#) is a Linux application for creating technical diagrams, whose interface and features are loosely based on Visio, the Microsoft Windows program.

Dia is easy enough to learn without much hassle and flexible enough to make power users feel at home when compared to commercial tools.

The first thing to do with Dia is to have a look at the [online Dia documentation](#). Then, play with the tool and familiarise yourself with it.

Now, consider Figure LX-06. It gives you an indication of the notation for (E)ER modelling that Dia supports.

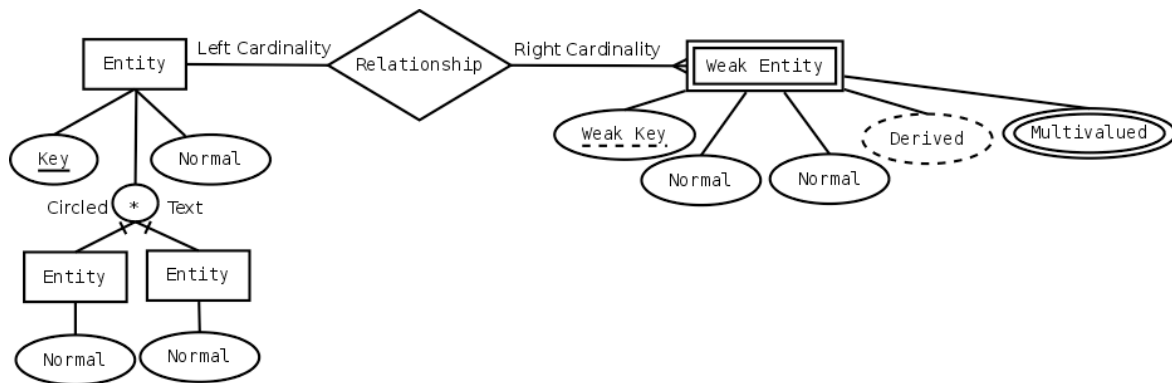


Figure LX-06: Legend: (E)ER Constructs in Dia

We have not covered (E)ER modelling yet, but we soon will. For this lab exercise, you do not need to worry about what each of the symbols, notation and concepts actually mean in detail. All you need to do is to use Figure LX-06 as a crib (by structural similarity) for you to reproduce Figure LX-07.

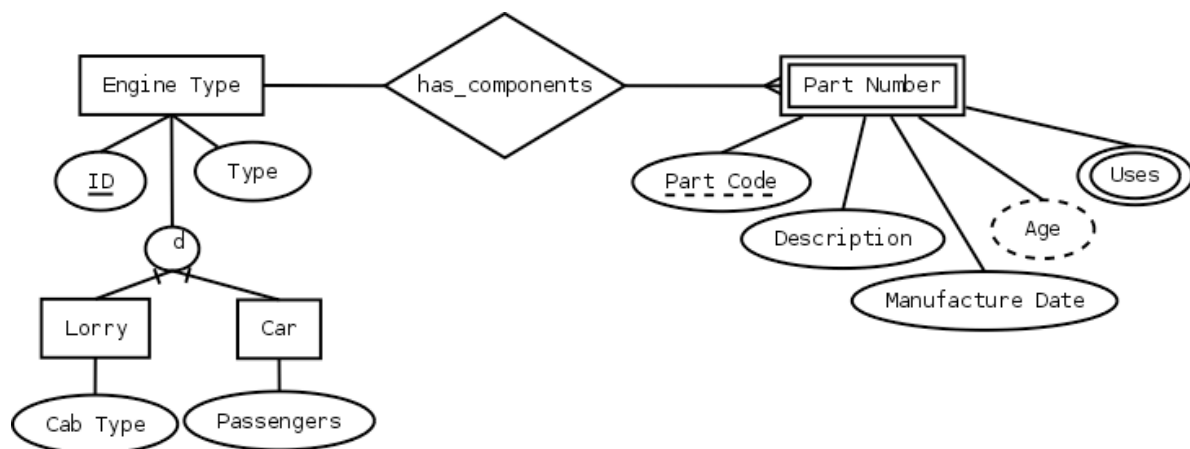


Figure LX-07: An Auto Manufacturer (E)ER Diagram in Dia

In other words, your objective is to use Dia and, using Figure LX-06 as your guide, create the diagram in Figure LX-07 in Dia, which is what will merit marks. You do not need to draw both diagrams, you only need to draw the one in Figure LX-07. Also, there is nothing tricky going on here: all we are asking you to do is, really, to replicate the diagram using Dia. We just want to make sure that you have used the tool and learned the basic functionality needed.

Work out by yourself how to draw the bottom part by exploring the tool and referring to the online documentation if you need to. A hint here is for you to, first of all, learn (either exploring or reading the online documentation) how Dia supports different diagrammatic notations and, in particular, how it supports the drawing of (E)ER diagrams. Another hint is that some lines are special, but do not look so. A final hint is that not all constructs you see drawn are supported in the same way (e.g., in terms of entering text into a shape).

Once you have finished, make sure you save your diagram as a .dia file in the appropriate directory (as with all other parts of any submission). Also, using Dia, you need to export the diagram into an EPS figure and include it in the LaTeX document. Do study the section of this manual where detailed instructions are given for this.

You are required to submit all the files used to generate the report using the LaTeX template we provided you with. This includes the .dia file, any .lst file(s) and the SQL script(s) you created, all stored in their appropriate directories. Again, study the section of this manual where detailed instructions are given for this.

Examples Clinic 2

Entity-Relationship Modelling

Specification

The goal of this examples clinic is to provide you with some practice on entity-relationship (ER) modelling. It aims to prepare you for the lab exercise in which you are asked to explain and extend an ER model for the Orinoco database that was used in the first lab exercise.

DURATION: 50 minutes examples clinic time.

DESCRIPTION: The examples clinic consists of two tasks: the first one asks you to draw an ER diagram given the specification of the data requirements; the second one asks you to reverse engineer the specification of the data requirements given an ER diagram. Work through the questions, demonstrate them to the lecturers and TAs in order to obtain feedback.

TASKS

T1. Consider the following set of requirements for an American UNIVERSITY database that is used to keep track of students' transcripts:

- I. The university keeps track of each student's name, student number, social security number, current address and phone, permanent address and phone, birthdate, sex, class (freshman, sophomore, ..., graduate), major department, minor department (if any), and degree program (BSc, MSc., ..., PhD). Some user applications need to refer to the city, state, and zip of the student's permanent address, and to the student's last name. Both social security number and student number have unique values for each student.
- II. Each department is described by a name, department code, office number, office phone, and college. Both name and code have unique values for each department.
- III. Each course has a course name, description, course number, number of semester hours, level, and offering department. The value of course number is unique for each course.
- IV. Each section has an instructor, semester, year, course, and section number. The section number distinguishes different sections of the same course that are taught during the same semester/year; its values are 1, 2, 3, ..., up to the number of sections taught during each semester.
- V. A grade report has a student, section, letter grade, and numeric grade (0, 1, 2, 3, 4 and F, D, C, B, A, respectively).

Design an ER schema for this application, and draw an ER diagram for that schema. Specify the key attributes of each entity type and the structural constraints on each relationship type. Note any unspecified requirements, and make appropriate assumptions to make the specification complete.

T2. Consider the ER diagram in Figure EC-03 (next page), which shows a simplified schema for an airline reservations system. Reverse-engineer from the ER diagram the textual specification of requirements and constraints that resulted in this schema. Try to be as precise as possible.

Acknowledgements

The tasks in this practical session are minor adaptations of material authored and made available to instructors by Ramez Elmasri and Shamkant B. Navathe to accompany their textbook Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8 Copyright © 2011 Pearson Education, Inc. Copyright remains with them, whom we thank. Any errors are our responsibility.

An ER diagram for an AIRLINE database schema.

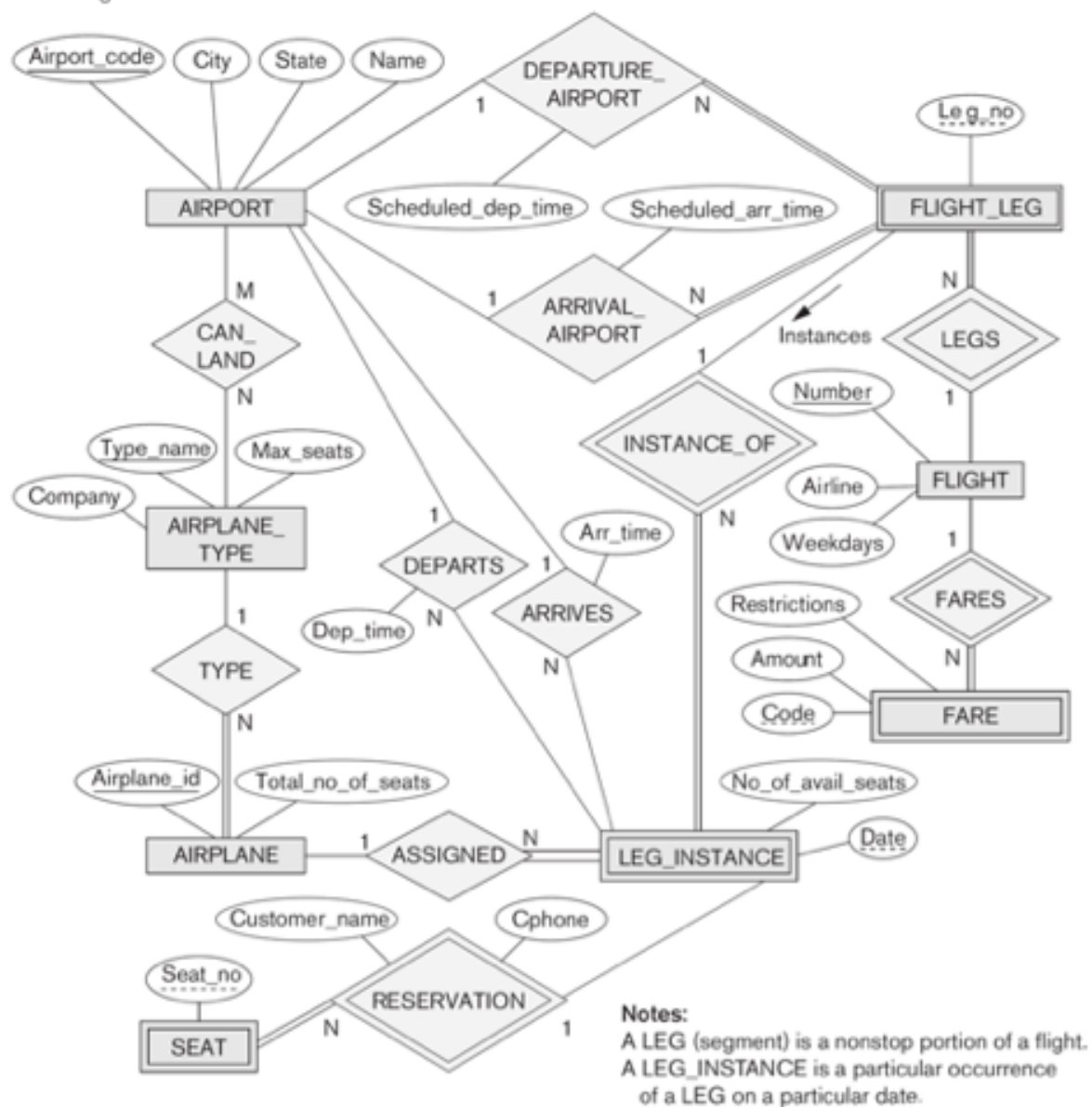


Figure EC-03: ER Diagram for an Airline Reservations System

Lab Exercise 02

(E)ER Modelling Specification

Overview

The goal of this lab exercise is to provide you with an opportunity to gain some practice in the conceptual design of databases.

You will start with a conceptual schema in the form of a ER diagram that represents the *Orinoco* case-study database that you used in Lab Exercise 1. You will be asked to write an explanation of concepts in the given diagram to demonstrate that you understand what information an ER diagram captures. We will then provide you with additional requirements for the *Orinoco* database and you will proceed to extend the given ER diagram to an extended (E)ER diagram so as to capture the new requirements.

For attendance, marks available, time available, submission requirements, deadline, and policy on extensions, see the initial sections of this manual.

Introduction

This lab exercise has two parts: in **Part 1**, your aim is to explain concepts (and in one case write down the data requirements) that can be inferred from a given ER diagram; in **Part 2**, your aim is to extend the given ER diagram to an enhanced entity-relationship (E)ER diagram that captures the additional data requirements provided.

Figure LX-08 (below) shows an ER schema corresponding to the (partial) *Orinoco* database as created in Lab Exercise 1 in Oracle using SQL scripts. This conceptual schema will be used as a starting point for this exercise and future laboratory exercises as well. The corresponding .dia file is available here

`/opt/info/courses/COMP23111/OrinocoInitial.dia`

you should make a copy of it in whatever directory in your personal file space you are going to work from in this lab exercise.

Requirements

From a Linux machine, you will use the Dia diagramming tool. Note that some diagrammatic conventions for constructs that belong to the Enhanced ER model may not be directly supported in Dia.

Instructions

Firstly, work your way through Part 1. Note what you need to submit and learn how to save the information as you go. At the end of the Part 1 you will be in a position to provide answers to items 1 to 6 below.

Secondly, in Part 2, use Dia to extend the diagram shown in Figure 1 below (working on the corresponding .dia file). This will allow you to provide an answer to item 7 below.

Finally, study the additional requirements and your extended diagram and consider providing the answer sought in the optional item 8 below.

Report Contents

Your report must contain:

1. An answer to Items 1-5 in Part 1 [5 marks]
2. An answer to Parts a-b of Item 6 in Part 1 [4 marks]
3. A figure, i.e., a Dia-drawn (E)ER diagram (*), including the requirements in Item 7 in Part 2 [9 marks]

Additional marks are available as follows:

4. Good report presentation (see the **Lab Sessions/Example Clinics** section above) [2 marks]

Part 1: Understanding the ER schema for the partial Orinoco database

To demonstrate that you understand the information represented by the schema, consider Figure LX-08.

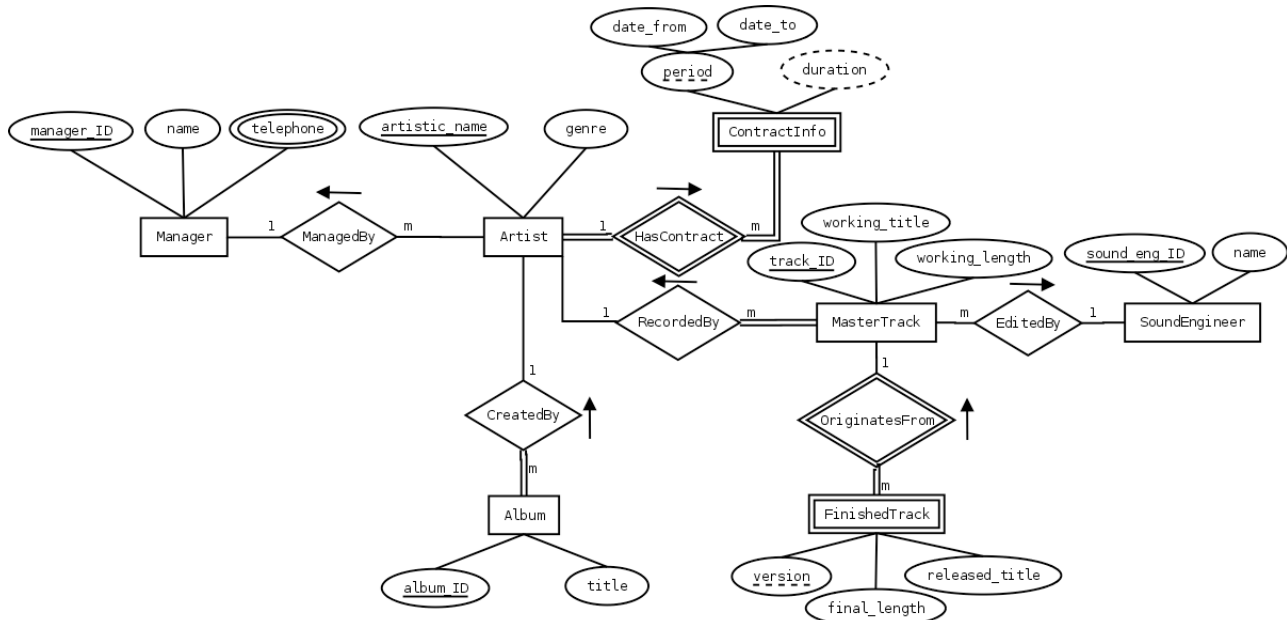


Figure LX-08: ER Diagram for the (Partial) Orinoco Database

Using Figure LX-08 (but you should also work with your copy of the corresponding .dia file in Dia), you are asked to explain the following:

1. What kind of attribute is the telephone attribute of the Manager entity type?
2. What kind of entity type is ContractInfo? What condition causes an entity type to be categorised as being of this kind?
3. What kind of relationship type is HasContract? What condition causes a relationship type to be categorised as being of this kind?
4. What kind of attribute is the period attribute of the ContractInfo entity type? What does the dashed underline underneath it mean? (Do not answer the second question with the same answer you gave to the first! We're looking for two pieces of knowledge here: what is it called and what is/are its defining characteristic/s!)
5. What kind of attribute is the duration attribute of the ContractInfo entity type? What does the dashed oval that encircles it mean? (Again, do not just repeat for the second question the answer you gave to the first! Again, we're looking for two pieces of knowledge here: what is it called and what is/are its defining characteristic/s!)
6. Write no more than three or four short paragraphs in the style of a data requirements specification document⁴ that might have led to the diagrammatic model you see in Figure 1 for:
 - a. the MasterTrack entity type and its attributes,
 - b. the three relationship types in which it participates including their corresponding cardinality ratio and participation constraints.

If you find you are writing more than half a page in giving answers to (1)-(6) above, you are probably misunderstanding what is being asked.

⁴ For an example of what is meant here by a *data requirements specification document*, see Chapter 7, Section 7.2, p. 197-198 of Elmasri And Navathe's Database Systems (6th ed.). This chapter is part of the collection of PDF files made available to you in the Mandatory Readings assignments. See also item 7 in this lab exercise for another example of what form such specifications can take. You do not need to follow any style strictly but you do need to convey in words the relevant information that is captured in that required parts of the ER diagram.

Part 2: Extending the ER schema for the Orinoco database

To demonstrate you can create ER conceptual schemas, you are now asked to extend the ER schema depicted in Figure LX-08 to an enhanced entity-relationship (E)ER diagram, which corresponds to the Orinoco database as you have created it in Lab Exercise 1 in Oracle using SQL scripts.

7. You should start with your copy of the given .dia file (see above for its path/location) and, using the Dia program, extend the diagram in order to capture the following new requirements:
 - a. Recall that in Lab Exercise 1 we said that, in the initial database we gave you to work with then, we did not include information that linked an album with the finished tracks it contains. It is now time for you to model the fact that an album must have at least one, and normally many, finished tracks and that a finished track may appear in many albums. You must also capture the sequence in which finished tracks appear in an album. Note that the same finished track may appear in different albums in different sequences.
 - b. The system must keep an online collection of catalogue entries, each of which references an existing album. An album must be listed as a catalogue entry at least once, and may be listed as several catalogue entries. Each catalogue entry is differentiated by its release date (to account for re-releases of the album). Both the price and the stock for each catalogue entry are also stored.
 - c. We record the name of buyers as well as the date at which they have registered with the Orinoco store. We also record a buyer ID to uniquely identify buyers.
 - d. A buyer can place any number of orders. An order must be placed by a buyer and is identified by an order number. We also store the date an order has been placed on by a buyer and the dispatched date of an order. An order comprises several catalogue entries. A catalogue entry can be part of many orders.
 - e. We want to model the fact that there exist two different types of artists: solo artists and group artists. A group artist can have as members more than one solo artist. You must also capture the date a solo artist has joined a particular group and the fact that a solo artist can be a member of more than one group. For solo artists, we would also like to store their real names and when they had their first performance. For group artists we would like to store the date a particular group has been formed. You should consider whether this specialisation leads to overlapping or disjoint sub-entities.
 - f. We also want to model the fact that an album is distributed as a vinyl album or a tape album or a CD album. A vinyl album can have many distinct colours, a CD album can have many different extras (e.g., an associated videoclip), and a tape can have exactly one label. You should also consider whether this specialisation leads to overlapping or disjoint sub-entities. (Hint: When a book is released in several forms, e.g., hardback, paperback, audiobook, does each format merit a different ISBN or not? So, is a book in hardback the same entity as a book in paperback or not? So, are hardback books and paperback books overlapping or disjoint sub-entity types?)

Now, you should think about the following **(optional)**⁵:

8. Is it possible for many artists to appear on one album? (Hint: follow the relationship paths and keep the cardinality ratios in mind.)

⁵ Things to think about. There are no marks associated with this task.

Examples Clinic 3

Mapping Entity-Relationship Diagrams into Relational Schemas

The goal of this examples clinic is to provide you with some practice on mapping entity-relationship (ER) conceptual models in the form of an ER diagram into a relational schema. It aims to prepare you for the lab exercise in which you are asked to map the ER model for the *Orinoco* database into a relational database schema.

DURATION: 50 minutes examples clinic time.

DESCRIPTION: The examples clinic consists of two tasks: the first one asks you to map a given ER schema (illustrated in Figure 1) into a relational schema; the second one asks you to specify ALL primary and foreign keys for the schema you generated for the first task. Work through the questions, demonstrate them to the lecturers and TAs in order to obtain feedback.

TASKS

T1. Consider the ER schema in Figure 1 for a database that may be used to keep track of transport ships and their locations for maritime authorities. Map this schema into a relational schema. Use the seven-step algorithm presented in the lectures.

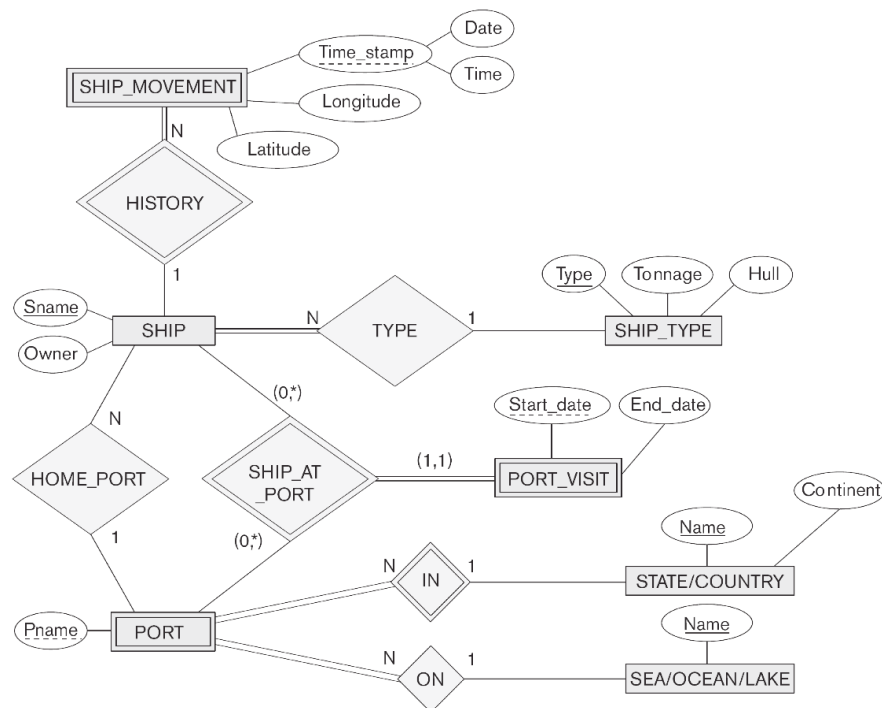


Figure EC-03: ER Diagram for a Maritime Transport Tracking Database

T2. Specify all primary keys and foreign keys for the schema you generated in T1.

Acknowledgements

The tasks in this practical session are minor adaptations of material authored and made available to instructors by Ramez Elmasri and Shamkant B. Navathe to accompany their textbook Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8 Copyright © 2011 Pearson Education, Inc. Copyright remains with them, whom we thank. Any errors are our responsibility.

Lab Exercise 03

EER-to-R Mapping, Functional Dependencies, SQL DDL/DML Specification

Overview

The goal of this lab exercise is to provide you with an opportunity to gain some practice in mapping a conceptual schema onto a relational one, and in identifying what functional dependencies hold in the latter, as well as in generating SQL DDL statements that populate a database with relations and the latter with instances.

You will start with your conceptual schema in the form of an (E)ER diagram that represents the *Orinoco* case-study database as it stands after the extensions carried out in Lab Exercise 2. You **must** use the EER diagram that you submitted for Lab Exercise 2 but you should incorporate the feedback provided to you. If you have doubts about this feedback and how to get your (E)ER diagram into a correct state, talk to the lecturers/TAs of the course.

We will provide you with a partial relational schema for the *Orinoco* database (almost corresponding to the ER diagram as it was at the start of Lab Exercise 2). You will be asked to complete this mapping task so that you obtain a relational schema for the *Orinoco* database as it stands after the extensions carried out in Lab Exercise 2. You will also be asked to identify the functional dependencies that hold in the relational schema.

You will also rely on the scripts used in Lab Exercise 1. You will be asked to extend them with SQL statements that create the new tables resulting from the extensions carried out in Lab Exercise 2 as well with SQL statements that populate the new tables.

For attendance, marks available, time available, submission requirements, deadline, and policy on extensions, see the initial sections of this manual.

Introduction

This lab exercise has two parts: in **Part 1**, your main aim is to complete the EER-to-relational mapping task so as to capture the extensions to the *Orinoco* database carried out in Lab Exercise 2; in **Part 2**, your main aim is to extend and execute the SQL scripts used in Lab Exercise 1 to cater for the extensions made in the *Orinoco* database since Lab Exercise 1.

Figure LX-09 (below) shows a partial relational schema, capturing the ER diagram of the *Orinoco* database (in a textual form) as it was at the start of Lab Exercise 2.

Requirements

From a Linux machine, you will use SQL*Plus to access the Oracle DBMS. The instructions for that were given in Lab Exercise 1. You should use the same user-name and password as you used in Lab Exercise 1.

Instructions

Firstly, work your way through Part 1. This should result in a complete relational schema for the extended *Orinoco* database that corresponds to the conceptual schema you submitted for Lab Exercise 2 and includes the partial relational schema in Figure LX-09. You are not required to explain the workings of the mapping algorithm, but you can do this if you wish and also, as always, make explicit any assumptions you feel you need to make if you believe information is missing, incomplete or conflicting. Then, working from the original SQL scripts, create the new one(s) that meet the requirements in Part 2.

Report Contents

Your report must contain:

1. The complete relational schema for the extended *Orinoco* database resulting from Part 1 [8 marks]

2. The SQL script resulting from Part 2 [10 marks]

Additional marks are available as follows:

3. Good report presentation (see the **Lab Sessions/Example Clinics** section above) [2 marks]

Part 1: Applying the EER-to-relational mapping to the extended Orinoco database

To demonstrate that you know how to map a conceptual schema to a relational one using the algorithm described and discussed in the lectures and available in the handouts (and the assigned readings), take as your starting point the conceptual schema for the extended *Orinoco* database, as shown in a textual form in Figure LX-09 (below). This shows the *Orinoco* database as it stood at the end of Lab Exercise 2.

Figure LX-09 shows the outcome of the EER-to-relational mapping algorithm if applied to the original *Orinoco* database, represented by the ER diagram you were given at the start of Lab Exercise 2. This part of this lab is about completing the relational schema in Figure LX-09 (below) so that it captures all the information in the conceptual schema you have submitted for Lab Exercise 2.

Using Figure LX-09 as your starting point, you must do the following:

1. Complete the mapping from EER-to-relational but going through all the entity types and relationship types (as well as their attributes, of course) that were added after Lab Exercise 2 (as reflected in your submitted solution). Two hints for you: (1) when mapping specialisation hierarchies, give priority to the approach that derives separate relations for both super- and sub-entity types, and (2) when mapping multivalued attributes consider whether you end up with no new, additional information in the corresponding relation, and, if so, make sure you note this in your report.
2. Make sure that you indicate clearly which are the (possibly composite) primary keys as well as the foreign keys (and, for the latter, what primary key in what relation the foreign key references).
3. For each of the relations you have added as a result of Steps 1-2 above, list the functional dependencies that they exhibit.

| | |
|---------------|--|
| Manager | (<u>manager_ID</u> , name) |
| ManPhone | (<u>manager_ID</u> [fk:Manager.manager_ID], telephone) |
| Artist | (<u>artistic_name</u> , genre, managedBy [fk:Manager.manager_ID]) |
| Album | (<u>album_ID</u> , title, createdBy [fk:Artist.artistic_name]) |
| MasterTrack | (<u>track_ID</u> , working_title, working_length, recordedBy [fk:Artist.artistic_name], editedBy [fk:SoundEngineer.sound_eng_id]) |
| SoundEngineer | (<u>sound_eng_ID</u> , name) |
| ContractInfo | (hasContract [fk:Artist.artistic_name], <u>date_from</u> , <u>date_to</u> , duration [derived as (date_to - date_from)]) |
| FinishedTrack | (<u>originatesFrom</u> [fk:MasterTrack.track_ID], <u>version</u> , released_title, final_length) |

Figure LX-09: Relational schema partially corresponding to the ER Diagram for the original *Orinoco* database

Part 2: Updating the schema/populating new relations in the Orinoco database

To demonstrate that you know how to use the basic data definition and data manipulation statement in SQL, take as your starting point the following:

- The relational schema you have created in Steps 1-2 in Part 1 above.
- The initial, original *Orinoco* database implemented in the Oracle DBMS in the state it was left at end of Lab Exercise 1. Remember that, in Lab Exercise 1, three SQL scripts were made available for you to create, delete and repair the tables and tuples in the original, initial *Orinoco* database.
- Consider again the scripts used in Lab Exercise 1 (you may have a personal copy in your own file space that you made when you did that lab exercise):

```

/opt/info/courses/COMP23111/create-Orinoco-initial-tables.sql
/opt/info/courses/COMP23111/drop-Orinoco-initial-tables.sql
/opt/info/courses/COMP23111/recreate-Orinoco-initial-tables.sql

```

The goal in this part of the course is for you to develop an SQL script (you should call it `create-Orinoco-extended-tables.sql`) that does for the extended *Orinoco* database the same job that the original SQL script did for the initial, original *Orinoco* database in Lab Exercise 1.

A side note here to motivate the task of populating the new tables that we ask that you do. Over the years, we have received feedback on this task that essentially is negative towards the amount of seemingly routine and "boring" effort involved in it. However, on reflection we hope that you will agree that, given the fact that the DBMS enforces integrity constraints, populating the database acts as a means to debug the schema you have provided. In this sense, populating the database is far from routine. Another reason why we must ask you to perform the task below is because we cannot provide you with a script for doing so. This is due to the fact that, as you know, the COMP23111 coursework is best understood as a single design/implementation task with each exercise being a stage in that longer process. As such, we cannot predict (so as to put in a script) what specific table and column names you will have chosen in your design/implementation.

Yet another reason why it is important that you populate the database as we ask you to is that the exercises that follow have examples (and marking schemes) that assume you have populated the database in the way we ask you to. So, with apologies for the rather repetitive nature of the population task below, we require you to do the following:

1. Create the new script(s)⁶ making sure that it contains:
2. All the table-creation information contained in the original script, except that you must make the following adaptations:
 1. All the table-creation information for the new relations that you derived in Steps 1-3 above (in doing so, you are advised to follow the style and conventions of the original scripts);
 2. All the SQL statements need to populate the tables as follows:
 1. The instance-level information in the original SQL script must be replicated in the new script(s) for the following tables: `Manager`, `ManPhone`, `Artist`, `Album`, `MasterTrack`, `SoundEngineer`, `ContractInfo` and `FinishedTrack`.
 2. The following buyers have registered with Orinoco: 'Simon Plant' on 28th Oct 2009; 'Julie Last' on 04th Aug 2011; 'Rina Blue' on 10th Mar 2012; 'Kay Srabonian' on 20th Dec 2009; and 'Yos Willis' on 10th Jan 2008. Buyers were assigned IDs 1, 2, 3, 4, and 5, respectively.
 3. Of the artists already present in the `Artist` table some are solo artists and some are groups. You should assume that the date a group was formed is the same as the date in which the first member of the group joined it. Some of the solo artists are members of one or more groups. The following is a complete list:
 1. Simon Palm's real name is 'Fred Maynard': he's a solo artist performing since 09th of Jan 2004 but is also a member of both the groups 'Goldfrat' since 23rd of Nov 2010 and 'Scandal' since 12th of Nov 2012.
 2. John Cliff has no stage name, he is known just as 'John Cliff': he's a solo artist performing since 12th of Dec 2005 and a member of 'Goldfrat' since 10th of Oct 2010.
 3. Jay Blancard's real name is 'Mervin James': he has been performing as a solo artist since 09th of Jun 2003 but is also a member of 'Goldfrat' since 10th of Oct 2010.
 4. Palmer John is really called 'Samantha Jules': she has been performing as a solo artist since 12th of Feb 2006 but is also a member of 'Flut' and 'Scandal' since 08th of Jun 2009 and 20th of Oct 2012 respectively.
 5. JZ is really called 'Julie Zeb': she's a solo artist performing since 09th of Mar 2004 but is also a member of both 'Flut' and 'Scandal' since 10th of Dec 2009 and 20th of Oct 2012 respectively.

⁶ You may wish also to study the other two scripts, the one that deletes tables and the one that repairs them. Notice that the repair script is nothing more than the concatenation of the delete and the create scripts, in that specific order. Then, after you have done the script that creates the tables (including the new ones), it may make your life easier to recreate the delete and repair scripts for the new database. However, pay attention that because there are dependencies between tables you may need to propagate the deletion. Look up the search terms `DROP CASCADE` in the Oracle documentation. Notice, furthermore, that it may make your life easier to break the original create scripts into two separate scripts: one that script that creates the tables and indexes, and another script that contains the statements to populate the tables created by the previous script. This means that you'd have two scripts rather than just one. For your submission, you can concatenate them back into a single one again.

6. JK Rowl's real name is 'Marisa Jaher': she has been performing as a solo artist since 11th of Jan 2003 but is also a member of both 'Flut' since 08th of Jun 2009 and 'Zero7' since 15th of Dec 2011.
7. Finally, PJ Blox's real name is 'Pandar Rahoux': he's a solo artist performing since 04th of Apr 2007 but is also a member 'Zero7' since 15th of Dec 2011.
4. Additional information on albums is as follows:
 1. 'Goldfrat' have recorded one album called '20 People in a Field', which has been released on CD, on vinyl and on tape, with assigned IDs '1c', '1v' and '1t' respectively. There are 2 CD releases: one with extra information available in the 'Web', and another with an extra 'Video'. There are also 3 vinyl releases: one on 'Red' vinyl, another one on 'Black', and a third one on 'Green'. The tape release has a 'Fancy' label.
 2. 'My Feet' by 'Flut' is released on CD (with an extra 'Game') and on 'Red' vinyl with assigned IDs '2c' and '2v' respectively.
 3. Finally, Jay Blancard's album 'Debut' is only released on CD with assigned ID '3c' but has two versions: one with extra 'Video' and another with an extra 'Game'.
5. The information about album tracks is as follows:
 1. The CD version of '20 People in a Field' has the following tracks in the sequence they are listed here: 'Hello', 'Extended Hello', 'Me', 'You', 'Them', 'Us', 'How', 'Be', 'Peace' and 'Smile'.
 2. The vinyl and tape versions of '20 People in a Field' only have the first 7 tracks of the CD version.
 3. Both the CD and vinyl versions of 'My Feet' have the following tracks in the sequence they are listed here: 'Wow', 'Whizzy', 'Who Are You?', 'Place and Time', 'Milko', 'Mad', and 'Bling'.
 4. Finally, 'Debut' has the following tracks in the sequence they are listed here: 'Power', 'What is your name?', 'Mix', 'Manic', 'Streets', 'Road', 'Forest', 'Deep', 'Danger' and 'Danger Remix'.
6. The online catalogue has been populated with entries for sale to the buyers. Each of the albums has an entry:
 1. '20 People in a Field' was firstly released on 24th of Dec 2010 as a tape, on 10th of Jan 2011 as a vinyl and five days later, on 15th of Jan 2011 as a CD. It is priced at £14.99 on CD for which the current stock is 2, it is priced at £12.99 on vinyl for which the current stock is 5, and, finally, it is priced at £10.99 on tape, for which the current stock is 9 tapes.
 2. 'My Feet' was firstly released as a vinyl on 20th of Jan 2010 and then on 15th Nov 2010 as a CD album. It is priced at £14.99 on CD and there is 1 copy in stock, while on vinyl it is priced at £12.99 but there is no stock left.
 3. Finally, 'Debut' by Jay Blancard was released on 10th Dec 2008. It is available only as a CD album at a price of £14.99 and there is one copy left in stock available.
7. Three orders have been placed, as follows:
 1. The first order was placed by Simon Plant, on the 17th of Jan 2011 and was dispatched three days later on the 20th. It had been assigned order ID=1 and consisted of one catalogue entry: Goldfrat's '20 People in a Field' on CD.
 2. The second order was dispatched to Rina Blue on the 15th of Mar 2012, after it was placed on the 12th of Mar 2012. It had been assigned order ID=2 and was for two catalogue entries: Goldfrat's '20 People in a Field' but this time on tape and the vinyl album 'My Feet' by 'Flut'.
 3. Finally, the third order had been placed again by Simon Plant on the 06th of Dec 2011 and was dispatched on the 10th of Dec 2011. It had been assigned order ID=3 and was for three catalogue entries: Goldfrat's '20 People in a Field' but this time on vinyl, the CD album 'My Feet' by 'Flut' and the CD album 'Debut' by 'Jay Blancard'.
8. Indexes provide opportunities for fast retrieval. We'll study them later in the course. For this exercise, you'll just imitate the index creation in the original SQL scripts you were given in Lab Exercise 1. In general, most DBMSs tend to create indexes on primary keys and foreign keys by default. However, most database applications that interact with humans also benefit from fast retrieval on human-readable information. Primary key and foreign key indexes are typically not (or not easily) human readable. You can see in the original SQL script that some indexes were created that aim to cater for these assumptions. Now, consider the new tables you're creating and their columns. Add SQL statements to your script that create indexes to facilitate fast retrieval on human-readable information they contain.
3. Test and debug your script until you're confident that it fulfils its purpose, i.e., to reflect the schema for the extended *Orinoco* database (as you developed it in Steps 1-2 in Part 1) and to populate this extended database with the instance-level information as described in Part 2.

Examples Clinic 4

Normalising Relational Schemas

Specification

The goal of this examples clinic is to provide you with some practice on normalising relational schemas. It aims to complement the lab exercise in which you are asked to map the EER model for the Orinoco database into a relational database schema.

DURATION: 50 minutes examples clinic time.

DESCRIPTION: The examples clinic consists of two tasks: in the first one you will examine a relation and, given the functional dependencies that hold within it, you need to decide which normal form the relation is in, explaining your answer, and then, in the second you need to normalise the relation, justifying each decomposition step. Work through the questions, demonstrate them to the lecturers and TAs in order to obtain feedback.

TASKS

T1. Consider the following relation about published books:

BOOK (Book_title, Authorname, Book_type, Listprice, Author_affil, Publisher)

Note that attribute Author_affil refers to the affiliation of the author (e.g., the university she or he belongs to).

Assume that the following dependencies hold in Book:

Book_title → {Publisher, Book_type}
 Book_type → Listprice
 Author_name → Author_affil

1. State what normal form the relation is in and explain your answer.
2. Apply normalisation until you cannot decompose the relations further, justifying each decomposition step.

T2. Consider the universal relation

$R = \{A, B, C, D, E, F, G, H, I\}$

and the set of functional dependencies

$F = \{ \{A, B\} \rightarrow \{C\}, \{A\} \rightarrow \{D, E\}, \{B\} \rightarrow \{F\}, \{F\} \rightarrow \{G, H\}, \{D\} \rightarrow \{I, J\} \}.$

1. What is the key for R?
2. Decompose R into 2NF, then 3NF relations.

Acknowledgements

The tasks in this practical session are minor adaptations of material authored and made available to instructors by (a) Raghu Ramakrishnan and Johannes Gehrke to accompany their textbook Database Management Systems, 3rd Edition, McGraw-Hill, 2002, 978-0-072-246563-1 Copyright © 2002 McGraw-Hill, and (b) Ramez Elmasri and Shamkant B. Navathe to accompany their textbook Database Systems: Models, Languages, Design, and Application Programming, 6th (Global) Edition, Addison-Wesley Pearson, 2011, 978-0-13-214498-8 Copyright © 2011 Pearson Education, Inc. Copyright remains with them, whom we thank. Any errors are our responsibility., whom we thank. All errors are our responsibility.

Lab Exercise 04

Making Use of a Relational Database Specification

Overview

The goal of this lab exercise is to provide you with an opportunity to gain some practice in writing and executing more complex queries (e.g., queries containing joins, aggregates, such as COUNT, and Views.).

You will start by building language-neutral queries using relational algebra (RA) and mapping your RA expressions into simple SQL Queries. Then, you will be requested to write more complex SQL queries and decompose scenarios, whose descriptions are provided to you, into SQL queries. To complete this lab exercise, you will need to understand the relational model, the syntax of SQL (including the syntax of update and insert), and index-creation concepts.

For attendance, marks available, time available, submission requirements, deadline, and policy on extensions, see the initial sections of this manual.

Introduction

This lab exercise is composed of three parts: in **Part 1**, you are requested to create language independent relational algebra expressions for a number of simple selection queries; in **Part 2**, you are requested to write and execute more complex SQL expressions, containing joins, aggregates and other modifiers; in **Part 3**, you are requested to write and execute SQL queries based on the description of scenarios provided to you.

Requirements

From a Linux machine, you will use SQL*Plus to access the Oracle DBMS. The instructions for that were given in Lab Exercise 1. You should use the same user-name and password as you used in Lab Exercise 1.

Instructions

Firstly, work your way through Part 1. This should result in a set of RA expressions for all the queries in that part. Then, work your way through Part 2. This time no RA expressions result, only SQL queries that, once more, you need to ensure you obtain the desired results when you evaluate them against the database. Finally, work your way through Part 3, for which the outcome is similar to Part 2, i.e., the SQL statements that capture the requirements and the results that those statements produce when you evaluate them against the database.

Report Contents

Your report must contain:

1. The RA expressions required in Part 1 along with a plain-English description of each expression, stating what you think your expression says [7 marks]
2. The SQL queries requested in Part 2, with the evaluation results [7 marks]
3. The SQL queries requested in Part 3, with the evaluation results [4 marks]

Additional marks are available as follows:

4. Good report presentation (see the **Lab Sessions/Example Clinics** section above) [2 marks]

Part 1: Writing simple queries as RA expressions

Starting from your extension to the *Orinoco* database (as resulted from your work on Lab Exercise 3), you need to write down language-neutral RA expressions for each of the queries below. As an **optional task**⁷ try

⁷Things to think about. There are no marks associated with this task.

to translate the RA expressions into SQL queries over the schema that resulted from your work on Lab Exercise 3. You should run each query, check the results it produces and then, if necessary, debug either (or both) the queries and the schema as appropriate.

Remember that it is best to use a text editor to write then either copy and paste it into SQL*Plus or invoke the .sql file as a script. Also, remember that it is often possible, and desirable, to build a query by stages, incrementally.

The queries are:

- I. Select all columns/fields from each of the following tables:
 - A. manager
 - B. manager phone
 - C. artist
 - D. solo artist (as you named it in Lab Exercise 3)
 - E. group artist (as you named it in Lab Exercise 3)
 - F. sound engineer
- II. Select all columns/ fields from the following tables:
 - A. solo artist (as you named it in Lab Exercise 3), just for entities with the artistic name of Palmer John;
 - B. group artist (as you named it in Lab Exercise 3), just for entities called Goldfrat;
 - C. master track, just for entities edited by Max Spring.
 - D. hasMember, just for entities of the group Goldfrat.
- III. Project **both** names of solo artists that are members of the group Flut.
- IV. Select all columns/fields from master track, in two ways, as follows:
 - A. ordered by working title in ascending order
 - B. ordered by working title in descending order.
- V. Project the columns 'recordedBy' and 'working_title' from master track, for the tracks edited by Max Spring, ordered by working title in reverse order.

Part 2: Writing and executing complex SQL queries

Write SQL queries for each of the following retrieval requests and then execute them to check they compute the right answer.

1. Select all columns/fields from master track and sound engineer joining them on the sound engineer's ID and using an alias for each table.
2. Find all solo artists who are members of Goldfrat, only printing the columns in the solo artists table and using an alias for each table.
3. Display each phone number joined to its associated manager's name (use an outer join for this).
4. Count the number of tracks in the master track table.
5. List the artists that are members of either Goldfrat or Scandal (or both).
6. Who belongs to both Goldfrat and Scandal?
7. Who are the members of Goldfrat that do not belong to Scandal?
8. Create a view selecting all the members of Scandal.
9. Create a view selecting all the members of Goldfrat.
10. Compute the intersection of the views in (8) and (9) above.
11. Create a nested query to select the price of the catalogue entry with the highest amount of stock.
12. Create a nested query to select all buyers that have placed an order.
13. Create a view to return the album information for the CD titled *20 People in a Field*.
14. Now use this view ((13) above) as part of a nested query to show the sequence in which the finished tracks appear as well as their release title.

Part 3: Using SQL queries to model data requirement scenarios

Write SQL statements that meet the requirements stipulated in the following scenarios:

1. Often a buyer will wish to see a list of tracks on an album in the order they appear. One such buyer wants to see the play list for the CD called *My Feet*. You were asked to write a SQL query that fulfils this requirement taking into account the constraint that your manager told you to only use joins (as opposed, e.g., to set-theoretic operations or nested subqueries) to accomplish it.
2. If buyers wish to purchase a catalogue item, they are only allowed to do so if there is enough stock. If a purchase can be made, the stock in the catalogue must be updated to reflect that purchase. You should create a single SQL statement that performs both tasks, updating the catalogue only if the desired item exists and if there is enough stock. Use the CD *20 People in a Field* as your test case.

Examples Clinic 5

Advanced SQL and Transactions

Specification

The goal of this examples clinic is to provide you with some practice on writing triggers and procedures, and on reasoning about transactions. It aims to complement the lab exercise in which you are asked to write triggers and procedures over the Orinoco database.

DURATION: 50 minutes examples clinic time.

DESCRIPTION: The examples clinic consists of two tasks: in the first one you will practice writing a trigger and a procedure, and then, in the second you practice writing concurrent schedules given serial ones.

Work through the questions, demonstrate them to the lecturers and TAs in order to obtain feedback.

TASKS

T1.

- I. What is PL/SQL?
- II. Write a procedure called `test1` to insert the variables `firstname`, `surname`, `age`, and `salary` into a table called `staff`.
- III. What is a trigger and what can triggers be used for?
- IV. Consider the tables shown below. The `numBookings` table stores the number of bookings (`num`) for a given train (`trainnum`) departing on a given date (`traindate`), where the number of bookings for a train on a date is obtained by counting the number of tuples in the `booking` table, which associates a given train (`trainnum`) and date (`traindate`) with a customer (`custnum`). Now, write a trigger that monitors inserts in the `booking` table and updates the `numBookings` table after each insert, keeping the number of bookings for each train up-to-date.

```
CREATE TABLE numBookings (
    trainnum    VARCHAR2(10) references train(trainnum),
    traindate   DATE,
    num         NUMBER,
    primary key (trainnum, traindate)
);
```

```
CREATE TABLE booking (
    custnum     VARCHAR2(10) references customer(custnum)
                on delete cascade,
    trainnum    VARCHAR2(10) references train(train#),
    traindate   DATE,
    primary key (custnum, trainnum, traindate)
);
```

T2.

- I. Consider transactions T_1 and T_2 described below, where $r_n(x)$ indicates a read operation on data item x by transaction T_n , and $w_n(x)$ indicates a write operation on data item x by transaction T_n :

Transaction T_1 : $r_1(x)$ $w_1(y)$

Transaction T_2 : $w_2(z)$

Also, consider the following serial schedules for transactions T_1 and T_2 :

Serial schedule T_1, T_2 : $r_1(x)$ $w_1(y)$ $w_2(z)$
 Serial schedule T_2, T_1 : $w_2(z)$ $r_1(x)$ $w_1(y)$

Create a concurrent schedule that is equivalent to both serial schedules.

- II. Similarly, consider transactions T_1 and T_2 described below, where $r_n(x)$ indicates a read operation on data item x by transaction T_n , and $w_n(x)$ indicates a write operation on data item x by transaction T_n :

Transaction T_1 : $r_1(z)$ $r_1(q)$ $w_1(y)$
 Transaction T_2 : $r_2(q)$ $w_2(z)$

Also, consider the following serial schedules for transactions T_1 and T_2 :

Serial schedule T_1, T_2 : $r_1(z)$ $r_1(q)$ $w_1(y)$ $r_2(q)$ $w_2(z)$
 Serial schedule T_2, T_1 : $r_2(q)$ $w_2(z)$ $r_1(z)$ $r_1(q)$ $w_1(y)$

Create a concurrent schedule that is equivalent to the serial schedule T_1, T_2 .

Is the schedule you created also equivalent to the serial schedule T_2, T_1 ? Why or why not?

Lab Exercise 05

Advanced SQL: Triggers and Procedures Specification

Overview

The goal of this lab exercise is to provide you with an opportunity to gain some practice in writing and executing triggers and PL/SQL procedures. To be able to solve this exercise, you will need to understand what triggers are and how they can be used to enforce integrity; in addition, you will need to understand how to create PL/SQL programs by using a combination of SQL and control statements, including the use of cursors.

For attendance, marks available, time available, submission requirements, deadline, and policy on extensions, see the initial sections of this manual.

Introduction

This lab exercise is composed of two parts: in **Part 1**, you are requested to write and test two simple triggers to support the maintenance of a derived attribute in the *Orinoco* database; in **Part 2**, you are requested to write and execute two PL/SQL procedures to support the maintenance of database integrity constraints and one in support of generate a formatted listing.

Requirements

From a Linux machine, you will use SQL*Plus to access the Oracle DBMS. The instructions for that were given in Lab Exercise 1. You should use the same user-name and password as you used in Lab Exercise 1.

In addition to the lecture handouts and the mandatory readings, it could be useful for you to consult reputable, trustworthy online sources for information on PL/SQL. Here is one recommendation that is available online to you as a University of Manchester student:

Oracle and PL/SQL Recipes
A Problem-Solution Approach
Josh Juneau, Matt Arena
ISBN: 978-1-4302-3207-0 (Print) 978-1-4302-3208-7 (Online)
<http://link.springer.com/book/10.1007/978-1-4302-3208-7>

Note that, often, there will need to be some adjustments due to the local setting of our Oracle DBMS.

Instructions

Work your way through each of the three parts above.

Report Contents

Your report must contain:

- | | |
|--|-----------|
| 1. The code for the trigger in Part 1 with the evaluation results ⁸ | [4 marks] |
| 2. The code for the procedures and view in Part 2 with evaluation results: | |
| 2.a The PL/SQL procedure for the ContractInfo | [4 marks] |
| 2.b.i. The AlbumDistribution view | [4 marks] |
| 2.b.ii. The PL/SQL procedure over the view from 2.b.i | [6 marks] |

Additional marks are available as follows:

⁸ Here and elsewhere, in your report, aim to show in your report the log of Oracle runs, e.g., as obtained by spooling, which we discussed in Lab Exercise 1. For this lab exercise, only a single correct run for each trigger or procedure is needed to help the marking.

3. Good report presentation (see the **Lab Sessions/Example Clinics** section above) [2 marks]

Part 1: Writing triggers

In this lab exercise (for all parts), remember that it is best to use a text editor to write then either copy and paste it into SQL*Plus or invoke the .sql file as a script. Also, remember that it is often possible, and desirable, to build a query by stages, incrementally.

If you get an error (and, of course, you're likely to), you can get more information on what caused the error by using the appropriate form of the `SHOW ERRORS`, so, look it up in the Oracle online documentation.

One additional thing to pay attention to here is that when you're debugging your triggers (and procedures) later on, an error may leave the database in an unexpected and undesirable state. Because of this, always think it through that you need to take action to reverse the changes and how to do so.

Finally, consider carefully the need to add a single slash in as the first and single character in the final line of a trigger or procedure.

Starting from your extension to the *Orinoco* database (as resulted from your work on Lab Exercise 3), you need to write a single trigger that fulfils the following requirements:

- The `duration` column in the `ContractInfo` table stores a derived attribute (you can see this on the EER diagram you were given). Create a trigger to compute the value of the `duration` column when a tuple is inserted in `ContractInfo`. Also, make sure that the trigger computes the value of the `duration` column when a tuple in `ContractInfo` is updated.

In doing this task, it is best to:

- retrieve all the tuples in the table (so that you know the state before the trigger is fired), then
- perform an operation that fires the trigger (e.g., `INSERT INTO ContractInfo VALUES ('Goldfrat', '10-oct-2008', '09-oct-2010', 0)`; for the first part of the trigger above and `UPDATE ContractInfo SET duration=0`; for the second part of the trigger above, then, finally,
- retrieve again all the tuples in the table (so that you can see and compare the state after the trigger has fired with that that existed before).

In your report, you must necessarily include the trigger and its results. As always, also include the output of the SQL script you created as part of the PDF document. In addition, make sure that you include the SQL script(s), within the .zip file you are uploading in Moodle, that causes the creation of the trigger, and make sure when it runs it produces the results from your report.

Part 2: Writing procedures

Starting from your extension to the *Orinoco* database (as resulted from your work on Lab Exercise 3), you need to write SQL statements and PL/SQL procedures, as follows:

- a) Create a single PL/SQL procedure that wraps the insertion of tuples in the `ContractInfo` table so as to make sure that the start and end dates of the new tuple do not lie between the start and end dates of any contract already present in the table for the given artist, and that the from and to dates are in the correct temporal order. The procedure takes as input the name of the artist whose contract this is, the start date and the end date of the contract. Note that the `BETWEEN` operator in SQL may come handy here (so, look it up in the Oracle online documentation). Note also that if you only need to test that a relation `R` violates a condition Θ then a `COUNT(*)` query with `R` in the `FROM` clause and Θ in the `WHERE` clause expresses your need.
- b) Write SQL statements and PL/SQL procedures that meet the requirements:

- (i) Create a view, named it AlbumDistribution, to return the album information along with a new attribute called is_distributed_as to store information as to whether an album can be distributed as a CD, on vinyl, or as a cassette tape (e.g., 'c', 'v', or 't').
- (ii) Recall the track listing scenario from Lab Exercise 4. It stated:

Often a buyer will wish to see a list of tracks on an album in the order of the play list. One such buyer wants to see the play list for a CD called "My Feet". It is your job to fulfil this request but your manager only wants you to use joins to accomplish it.

Now, use the view (2.b.i from above) and create a PL/SQL procedure that extends and wraps the resulting as follows. Firstly, it must take as input parameters the album type (e.g., 'c', 'v', or 't') and the album title. Secondly, in the case of album title, it must allow one to enter wildcard sequences, (e.g., '%e%', or '%t%'). Finally, it must produce a formatted listing, with the sequence followed by a comma, then the title of the track followed by a greater-than (i.e., right angle-bracket), followed by the album title. **Hint:** remember that you need to set the server output on.

In doing these tasks, it is best to:

- retrieve all the tuples in the table (so that you know the state before the procedure is invoked for execution), then
- invoke the procedure with values that test the logic of your code (e.g., in the the case of the first procedure, use one or two INSERTs that are illegal and one or two that are legal, and so on), then, finally,
- retrieve again all the tuples in the table (so that you can see and compare the state after the procedure has executed with that that existed before).

In your report, you must necessarily include the PL/SQL procedures as well as the SQL statements that create the view. As always, include the output of the SQL script you created as part of the PDF document. In addition, make sure that you include the SQL script(s), within the .zip file you are uploading in Moodle that causes the creation of the trigger, and make sure when it runs it produces the results from your report. Do study the section of this manual where detailed instructions are given for this.