## Overview [Lab work and Homework]

As discussed in the lecture we will start off in the module by presenting you with an opportunity to get setup with some of the basic tooling which will be used for the module. In this lab we will introduce some of our basic **tech stack** and begin exploring some basic use of the various elements of this tech stack.

Many of the exercises below are intended to prompt both **thinking** and **discussion**, don’t just be satisfied with a **surface level of understanding** talk to your tutors in class and share your thoughts and ideas with us!

This tutorial makes use of the XAMPP software package, we recommend using a [portable version](https://sourceforge.net/projects/xampp/files/XAMPP%20Windows/8.2.4/xampp-portable-windows-x64-8.2.4-0-VS16.zip/download) of this software package The lab tutorial forms both the lab work and homework for the week 5 activities.

**Objectives**

* Analyse and dissect existing sample entity relationship diagrams to understand the planning / design information they convey to the Engineering team.
* Produce SQL statements which would be capable of putting example records into the tables represented within the design found in a given ERD.
* Explain the purpose of some basic SQL statements used to implement a database system.
* Produce an entity relationship diagram which conveys information about the structure of an existing database system.[[1]](#footnote-2)

## Additional information: Tracking your learning progress.

To help track your progress early in the module, we will be using a little system in these first few weeks to **help you track your understanding so far**. This will give you an opportunity to show how you are getting on with understanding the material each week during the early stages of the module so we can **better guide you** through **developing your knowledge and understanding**. When attempting your answer also be honest and discuss with **us when you needed to do some additional background research**[[2]](#footnote-3), and **show us where you got your information from**[[3]](#footnote-4).

**Progress checker**

The progress checker is included at the end of our various exercises to allow you to keep a record of your learning, giving you an opportunity to quickly show us your progress and discuss in class during the following week. It also encourages you to keep a record of this progress to help you to reflect on your developing skills and understanding during the module.

Example “Progress checker”

Please select the answer you feel **best describes your level of understanding** for **each exercise** (an example progress checker can be seen in **figure 1**.

|  |  |
| --- | --- |
| **Now let’s check how well you have understood so far! 😊** | |
| How did you find attempting this exercise? | Choose one (**X**) |
| Grinning Face I’m **very confident** that I have understood and completed this task successfully | **X** |
| Slightly Smiling Face I’m **fairly confident** that I have understood and completed this task successfully |  |
| Thinking Face I think I’m **getting there** but I need to **work on understanding this a little more** |  |
| Grinning Face with Sweat I **got a bit stuck** and I will need to **work on understanding this a lot more** |  |

Figure 1 An example 'progress checker'

**Food for thought**

The “Food for thought” messages boxes provide some prompts for you to work on deepening your understanding as you progress through the labs a good Computer Scientist / Data Scientist / Software Engineer recognises the value of picking apart the tools and concepts they are learning about to deepen their understanding. Fundamentally, we aim for this module to not just be about you “Developing Database-backed Web Applications” but about refining your approach to problem solving in Software Development, to help you become a more well-rounded Engineer 😊

To help this process along we will use the “Food for thought” message boxes to provide some suggested starting points to investigate various aspects of what you are doing in the labs further, moving you away from approaching development as simply **“remembering ‘recipes’ from a cookbook”** to **“understanding what tools/techniques/concepts to apply”**, **“when to apply them”**, **“how to apply them”** and **“why we tend to apply them in a particular way”**



Figure 2 The Swedish Chef (By Disney.com, Fair use, https://en.wikipedia.org/w/index.php?curid=40291413)

Think about it like this: developers are a bit like chefs, we don’t just memorise and follow a bunch of “recipes” (example programs) from someone else’s “cookbook” (tutorial / documentation / example code) we are experienced enough to understand **how** those recipes work and **why** they work so we can use that understanding to develop our own recipes and serve up our own tasty dishes to our customers (our own software applications)

Example “Food for thought”

|  |
| --- |
| CookiesThinkingIdea**Food for thought** |
| Let’s deepen our understanding by thinking a little deeper about this exercise and discussing our thoughts in class. |
| We will have a sample prompt to think about / experiment with some aspect of the lab in this bit usually. |

**Be an active learner. Help us help you to grow.**

As well as recording your progress with the lab work **so far**[[4]](#footnote-5) and discussing with us during the following week, please make sure you tell us **in class**[[5]](#footnote-6) about any aspects of the current week’s work you are particularly struggling to understand. From what we have seen **so far** based on your progress at Level 4 many of you are still working your way up from the **lower levels** in your **understanding of programming / software development**. To turn you into an effective **Software Engineer** / **Software Developer** we need you to **work with us** to move away from those **basic**, **lower levels of understanding**, which will require **you** to work on **actively developing your skills** (with our guidance along the way 😊) to reach those **higher levels**[[6]](#footnote-7).

We want you to move from:

* **Memorising** recipes / definitions

By

* **Understanding** the tools, techniques and concepts underpinning our examples / exercises

Then working on

* **Applying** these tools, techniques, and concepts. Experimenting with them to gain deeper understanding.

So that you can eventually

* **Create** your own original software solutions through an informed approach to planning, designing, and implementing software.
* **Analyse** the different approaches (e.g tools, techniques and concepts) that could be applied to help solve a problem and how best to use them.
* **Evaluate** your approach (and others) to help develop your better judgment when creating software.

A diagram of a diagram of a diagram

Description automatically generated

Figure 3 The different levels of learning, to discuss in class (ThoughtCo, 2019)

# Week 4 – Designing Databases part 1

Before attempting the below lab exercises ensure that you have set up an XAMPP instance capable of running the Apache web server and MySQL on the device you are using, work with your lab tutor to familiarise yourself with this process – they will likely demo this process as part of the introductory lab talk.

You may find the following documentation available online useful for your exploration and interpretation of Entity Relationship Diagrams

* [Article] [Visual Paradigm: What is Entity Relationship Diagram (ERD)?](https://www.visual-paradigm.com/guide/data-modeling/what-is-entity-relationship-diagram/)
  + To bolster your understanding of what information is contained within an ERD.
* [Article] [Visual Paradigm: What is Data Modeling?](https://www.visual-paradigm.com/guide/data-modeling/what-is-data-modeling/)
  + To bolster your understanding of why we would use data modelling techniques such as entity relationship modelling in Software Engineering projects.
* [Article] [Work with entity relationship table shapes in draw.io](https://www.drawio.com/blog/entity-relationship-tables)
  + This will be useful for providing some software capable of drawing out your design diagrams for various ERDs.
* [[Video] Entity Relationship Diagram (ERD) Tutorial – Part 1](https://www.youtube.com/watch?v=QpdhBUYk7Kk)
* [[Video] Entity Relationship Diagram (ERD) Tutorial – Part 2](https://www.youtube.com/watch?v=-CuY5ADwn24)
* [[Video] Database Keys Made Easy – Primary, Foreign, Candidate, Surrogate, & Many More](https://www.youtube.com/watch?v=8wUUMOKAK-c)
* [[Video] Create Entity Relationship Diagrams (ERD) with draw.io](https://www.youtube.com/watch?v=JYZPdU5F2iM)

Note: The diagrams below were created using Visual Paradigm Community Edition. However, equivalent Crows Feet notation ERDs can be created using the Software at [draw.io](https://draw.io/).

## Exercise 1

Please answer the below questions based on the following entity relationship diagram:

Diagram

Description automatically generated

Figure 4 Example ERD 1

1. Is this kind of ERD a logical, conceptual, or physical ERD model of a database system? Justify your answer.
2. What are the entities that are represented within this system design?
3. What attributes are represented within this system design?
4. Write out some [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statements in SQL which would insert some example tuples/records into Customer, Customer\_Movie, Movie and Producer (aim for at least 2 records for each) – you do not need to create the database beforehand! Just think about what records you would like to add and what SQL code you could write to insert those records.
5. What benefits might storing the FirstName and LastName of the customer as separate items provide?
6. Why might it be better to use a date datatype for the Customer’s data of birth rather than a TEXT, VARCHAR or String datatype?
7. What issues might occur with storing a value from 1 to 5 for a star rating in the form of a single character be?
8. What is the meaning of the key symbol next to ‘MovieID’ on Customer\_Movie?
9. What is the meaning of the key symbol next to ‘CustomerID’ on Customer\_Movie?
10. Why are Customer and Movie not connected directly to each other? What benefit does Customer\_Movie provide?
11. What cardinality of relationship exists between Customer and Customer\_Movie? What do you think this relationship represents?
12. What kind of software system do you believe this partial database design might be targeted towards?

|  |
| --- |
| CookiesThinkingIdea**Food for thought** |
| Let’s deepen our understanding by thinking a little deeper about this exercise and discussing our thoughts in class. |
| What limitations does this current database design present about the kind of software system which it could underpin? Think about the kind of data that would be able to be represented within the system based on the entities, attributes and relationships present. |

|  |
| --- |
| **My solution** Idea |
| Record your notes about how you attempted to answer the question / solve the exercise in this section (just edit the text box 😉)   1. The given ERD is a logical model. It shows entities like Customer, Movie, Producer, and Customer\_Movie, along with their attributes such as ID, FirstName, and Title. It indicates the type of data for each attribute, like varchar or integer, and suggests primary keys with "ID" fields and foreign keys through matching attributes in related tables. However, it does not include specific database data types, clear notations for primary and foreign keys, indexes, constraints, or other details needed for a physical ERD. It also does not provide a high-level view like a conceptual ERD, which would leave out attributes. Thus, it remains at the logical level, outlining the data structure and relationships without detailing the physical implementation. 2. The Entities that are present in the system design are: customer, movie, producer, customer\_movie. 3. The attributes represented in the system Design are:   Customer: ID (primary key), FirstName, LastName, Address, BirthDate, PhoneNumber.  Movie: ID(primary key), ProducerId (Foreign key), Title, Duration, Rating.  Producer: ID, CompanyName, Country.  Customer\_Movie: MovieID (Primary Key, Foreign Key), CustomerID (Primary Key, Foreign Key), Date\_Rented, Due\_Date.   1. For Customer table:  * INSERT INTO Customer (ID, FirstName, LastName, Address, BirthDate, PhoneNumber)   VALUES (1, 'Ankit', 'Mahato', 'maitidevi', '2004-02-15', 1234567890);   * INSERT INTO Customer (ID, FirstName, LastName, Address, BirthDate, PhoneNumber)   VALUES (2, 'Alan', 'Walker', 'Oak Ave', '1985-05-20', 9876543210);  For Movie table:   * INSERT INTO Movie (ID, ProducerID, Title, Duration, Rating)   VALUES (1, 1, 'The Shawshank Redemption', 142, 'R');   * INSERT INTO Movie (ID, ProducerID, Title, Duration, Rating)   VALUES (2, 2, 'The Godfather', 175, 'R');  For Producer table:   * INSERT INTO Producer (ID, CompanyName, Country)   VALUES (1, 'Castle Rock Entertainment', 'USA');   * INSERT INTO Producer (ID, CompanyName, Country)   VALUES (2, 'Paramount Pictures', 'USA');  For Customer\_Movie table:   * INSERT INTO Customer\_Movie (MovieID, CustomerID, Date\_Rented, Due\_Date)   VALUES (1, 1, '2024-07-20', '2024-07-27');   * INSERT INTO Customer\_Movie (MovieID, CustomerID, Date\_Rented, Due\_Date)   VALUES (2, 1, '2024-07-28', '2024-08-04');   1. Keeping customer FirstName and LastName as separate fields has many benefits for managing data and improving application features. It allows for different ways to format and show names, makes it easier to search and sort by each name part, and helps maintain data accuracy by avoiding inconsistencies in combined names. This approach also respects various cultural naming traditions, enhances personalized interactions with users, and boosts data analysis by allowing separate examination of name parts. Additionally, it makes it easier to connect with other systems that need specific name formats. 2. Using a DATE data type for a customer's birthdate is better than using text types like TEXT or VARCHAR. It helps keep data accurate, saves storage space, and improves functionality. DATE ensures that only valid dates are entered, which helps maintain consistency. It usually takes up less space than text formats. It also standardizes how dates are formatted, making data easier to process and reducing format issues. Additionally, date data types allow for the use of useful built-in functions for calculations and comparisons, enhance the efficiency of queries and reports that filter or sort by date, and support effective indexing for faster query performance. Lastly, using the right data type helps maintain data integrity by clearly defining what kind of data is expected, preventing the storage of incorrect values. 3. Storing star ratings (1-5) as single characters causes several problems. First, it creates confusion about data types, making calculations harder and leading to possible errors. When sorting, characters are arranged alphabetically instead of numerically, which complicates the process. Input validation must check both the type of character and its range. Although it may save a little space, using integers is generally more efficient. This method can confuse both developers and users, hiding the fact that the data is numeric. Additionally, if the rating scale needs to be expanded, using single characters can complicate things and may require changing the data type. Storing ratings as integers avoids these issues, allowing for correct calculations, easier sorting, and clearer validation while making it simpler to extend the rating scale. 4. In the Customer\_Movie table, the key symbol next to MovieID shows that it is part of the composite primary key. When combined with CustomerID, it ensures that each record in the database is unique, meaning a specific movie can only be associated with one client for a single rental. 5. In the Customer\_Movie table, the key symbol next to CustomerID also signifies that it is a component of the composite primary key. It gives each rental record a unique identity in addition to MovieID. This makes it possible for the same client to rent more than one movie, yet each rental is unique.      1. The Customer and Movie entities have a many-to-many relationship, meaning they are not directly linked. One customer can rent multiple movies, and multiple customers can rent the same movie. To manage this relationship effectively, the Customer\_Movie table acts as a junction table. It also allows for the storage of additional details specific to each rental, such as Date\_Rented and Due\_Date. This separation leads to improved data normalization, reduced redundancy, and a more adaptable database design. 2. A customer can have multiple entries in the Customer\_Movie table because of the one-to-many connection between Customer and Customer\_Movie. Each entry in this table represents a different rental transaction, showing that a customer can rent many movies. 3. This database design is aimed at a system for managing movie rentals, suitable for a video rental store (either physical or online) or a digital movie library. The entities and attributes help in tracking customers, movies, producers, and rental transactions, including due dates. Although it can be adapted for general e-commerce, the specific focus indicates it is primarily for movie rentals. This allows for effective inventory management, customer tracking, rental processing, and necessary reporting for the business. It could also work for a streaming service, but would need changes to monitor streaming data instead of physical rentals. |
|  |

|  |  |
| --- | --- |
| **Now let’s check how well you have understood so far! 😊** | |
| How did you find attempting this exercise? | Choose one (**X**) |
| Grinning Face I’m **very confident** that I have understood and completed this task successfully | **X** |
| Slightly Smiling Face I’m **fairly confident** that I have understood and completed this task successfully |  |
| Thinking Face I think I’m **getting there** but I need to **work on understanding this a little more** |  |
| Grinning Face with Sweat I **got a bit stuck** and I will need to **work on understanding this a lot more** |  |

## Exercise 2

Please answer the below questions based on the following entity relationship diagram:

A screenshot of a computer

Description automatically generated

Figure 5 Example ERD 2

1. Is this kind of ERD a logical, conceptual, or physical ERD model of a database system? Justify your answer.
2. What are the entities that are represented within this system design?
3. What attributes are represented within this system design?
4. Write out some [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statements in SQL which would insert some example tuples/records into Customer, Order, Order\_Product, Employee, and Product (aim for at least 2 records for each) – you do not need to create the database beforehand! Just think about what records you would like to add and what SQL code you could write to insert those records.
5. What is the meaning of the key symbol next to ‘ID’ on Customer?
6. What is the meaning of the ‘salesRepEmployeeNum’ on Customer?
7. What kind of relationship cardinality is there between Customer and Order? Why might this be?
8. Why are Product and Order not connected directly to each other, what benefit does Order\_Product provide?
9. What is the meaning of Productlineid on Product?
10. What kind of software system do you believe this partial database design might be targeted towards?

|  |
| --- |
| CookiesThinkingIdea**Food for thought** |
| Let’s deepen our understanding by thinking a little deeper about this exercise and discussing our thoughts in class. |
| What limitations does this current database design present about the kind of software system which it could underpin? Think about the kind of data that would be able to be represented within the system based on the entities, attributes and relationships present. |

|  |
| --- |
| **My solution** Idea |
| Record your notes about how you attempted to answer the question / solve the exercise in this section (just edit the text box 😉)   1. This ERD is a logical ERD as it shows the database structure with a fair amount of detail. It is more detailed than a conceptual ERD because it specifies data types for each attribute (like varchar, integer, numeric) and identifies primary keys clearly. The relationships between entities are clearly defined, including cardinality, such as one-to-many relationships. However, it is not a physical ERD since it does not include specific implementation details for a database, like index definitions, table spaces, or constraints needed by a specific database management system. 2. The Entities represented in the system design are: Productline, Product, Order, Order\_Product, Customer, Employee, Office, Payment. 3. Each entity has the following attributes:  * Productline: ID, DescInText, DescInHTML, Image. * Product: Code, ProductlineID, Name, Scale, Vendor, PdtDescription, QtyInStock, BuyPrice, MSRP. * Order: ID, CustomerID, OrderDate, RequiredDate, ShippedDate, Status, Comments. * Order\_Product: ID, OrderID, ProductCode, Qty, PriceEach. * Customer: ID, salesRepEmployeeNum, Name, LastName, FirstName, Phone, Address1, Address2, City, State, PostalCode, Country, CreditLimit. * Employee: ID, OfficeCode, reportsTo, LastName, FirstName, Extension, Email, JobTitle. * Office: Code, City, Phone, Address1, Address2, State, Country, PostalCode, Territory. * Payment: CheckNum, CustomerID, PaymentDate, Amount.  1. For Customer,  * INSERT INTO Customer (ID, salesRepEmployeeNum, Name, LastName, FirstName, Phone, Address1, City, State, PostalCode, Country, CreditLimit)   VALUES (301, 401, 'Pashupati Traders', 'Oli', 'KP', '9841234567', 'Baluwatar', 'Kathmandu', 'Bagmati', '44600', 'Nepal', 250000.00),  (302, 402, 'Bhrukuti Stores', 'Prachanda', 'Pushpa Kamal', '9849876543', 'Khumaltar', 'Lalitpur', 'Bagmati', '44700', 'Nepal', 150000.00);  For Order,   * INSERT INTO `Order` (ID, CustomerID, OrderDate, RequiredDate, ShippedDate, Status, Comments)   VALUES (101, 101, '2024-03-15', '2024-03-22', '2024-03-20', 1, 'Bat-Signal delivery'), (102, 102, '2024-03-20', '2024-03-29', '2024-03-25', 2, 'Repulsor-powered delivery expected.');  For Order\_Product,   * INSERT INTO Order\_Product (ID, OrderID, ProductCode, Qty, PriceEach)   VALUES (1, 1, 101, 5, 50.00), (2, 2, 102, 10, 30.00);  For Employee   * INSERT INTO Employee (ID, OfficeCode, reportsTo, LastName, FirstName, Extension, Email, JobTitle)   VALUES (1001, 1, NULL, 'Brown', 'Alice', 'x1001', 'alice.brown@gmail.com', 'Sales Manager'), (1002, 2, 1001, 'White', 'Bob', 'x1002', 'bob.white@gmail.com', 'Sales Rep');  For Product   * INSERT INTO Product (Code, ProductlineID, Name, Scale, Vendor, PdtDescription, QtyInStock, BuyPrice, MSRP)   VALUES (101, 1, 'Product A', '1:10', 'Vendor A', 'High-quality product A', 100, 20.00, 50.00),  (102, 2, 'Product B', '1:20', 'Vendor B', 'High-quality product B', 200, 15.00, 40.00);   1. The key symbol next to ID in the Customer table indicates that it is the primary key. It uniquely identifies each record in the Customer table, ensuring no two customers have the same ID. 2. The salesRepEmployeeNum attribute in the Customer table is a foreign key that points to the ID in the Employee table. This connection helps to track and manage customer accounts by assigning each customer to a specific employee. 3. Customer and Order have a one-to-many cardinal relationship. This implies that although a single consumer may place several orders, only one client is linked to each order. Real-world situations where customers might make several transactions over time are reflected in this design. 4. Due to the many-to-many relationship, the product and the order are not directly related. Multiple products may be included in a single order, and a single product may be included in several orders. For each product in an order, the Order\_Product table serves as a junction table to manage this relationship effectively and permits extra information like Qty (quantity) and PriceEach (price per unit). This offers normalisation and flexibility. 5. The Order\_Product table is crucial for handling the many-to-many relationship between Product and Order. It helps avoid duplication and keeps data accurate. This table stores important details about products in each order, such as quantity and price, which would be hard to manage if the two tables were linked directly. It also makes it easy to add, remove, or change product information in an order without altering the main Product or Order tables, and it simplifies the process of retrieving information about specific product-order pairs. 6. This database design seems aimed at a sales and inventory management system. A company could use it for managing product lines, tracking orders, linking customers with sales representatives, processing payments, and handling office tasks. Businesses that depend a lot on product sales and order management would benefit greatly from this system. |

|  |  |
| --- | --- |
| **Now let’s check how well you have understood so far! 😊** | |
| How did you find attempting this exercise? | Choose one (**X**) |
| Grinning Face I’m **very confident** that I have understood and completed this task successfully |  |
| Slightly Smiling Face I’m **fairly confident** that I have understood and completed this task successfully |  |
| Thinking Face I think I’m **getting there** but I need to **work on understanding this a little more** |  |
| Grinning Face with Sweat I **got a bit stuck** and I will need to **work on understanding this a lot more** |  |

## Exercise 3

Please answer the below questions based on the following entity relationship diagram:

A diagram of a work flow

Description automatically generated

Figure 6 Example ERD 3

1. Is this kind of ERD a logical, conceptual, or physical ERD model of a database system? Justify your answer.
2. What are the entities that are represented within this system design?
3. What attributes are represented within this system design?
4. Why are Students not directly connected to Module, what purpose does Module\_Student serve?
5. Write out some [INSERT INTO](https://www.w3schools.com/sql/sql_insert.asp) statements in SQL which would insert some example tuples/records into Student, Module\_Student, Module, Department and Instructor (aim for at least 2 records for each) – you do not need to create the database beforehand! Just think about what records you would like to add and what SQL code you could write to insert those records.
6. Which SQL DDL statement would allow you to create the data structure which will be used to store Student data?
7. Which SQL DDL statement would allow you to create the data structure which will be used to store Module\_Student data?
8. What kind of software system do you believe this partial database design might be targeted towards? What limitations does the current design present?

|  |
| --- |
| CookiesThinkingIdea**Food for thought** |
| Let’s deepen our understanding by thinking a little deeper about this exercise and discussing our thoughts in class. |
| What limitations does this current database design present about the kind of software system which it could underpin? Think about the kind of data that would be able to be represented within the system based on the entities, attributes and relationships present. |

|  |
| --- |
| **My solution** Idea |
| Record your notes about how you attempted to answer the question / solve the exercise in this section (just edit the text box 😉)   1. The provided ERD diagram is a logical model for a database system. It outlines the columns in each table along with their data types. Key features of a physical ERD model include the representation of these data types and the clear identification of primary keys in each table. 2. The Entities represented in the system design are: Department, Instructor, Module, Student, Module\_Student. 3. Each entity has the following attributes:  * Department: Name, Location, headedBy. * Instructor: ID, DepartmentName, FirstName, LastName, Phone. * Module: ID, DepartmentName, InstructorID, Duration, Name. * Student: ID, FirstName, LastName, Phone. * Module\_Student: ModuleID, StudentID, Grade  1. The relationship between students and modules is characterized as many-to-many, as an individual student may enroll in several modules, and a single module can accommodate multiple students. This complex relationship is facilitated by the Module\_Student table, which acts as a junction table. Additionally, this table allows for the inclusion of supplementary information, such as grades, which track each student's performance in specific modules. Without this intermediary table, it would be challenging to accurately represent and oversee these relationships. 2. For Department  * INSERT INTO Department (Name, Location, headedBy)   VALUES ('Computer Science', 'Pashupati Campus', 101),  ('Mathematics', 'Tribhuvan University Building', 102);  For Instructor,   * INSERT INTO Instructor (ID, DepartmentName, FirstName, LastName, Phone)   VALUES (101, 'Computer Science', 'Aarati', 'Shrestha', '9841234567'),  (102, 'Mathematics', 'Bishal', 'Poudel', '9812345678');  For Module,   * INSERT INTO Module (ID, DepartmentName, InstructorID, Duration, Name)   VALUES (1, 'Computer Science', 101, 10, 'Database Systems'),  (2, 'Mathematics', 102, 8, 'Linear Algebra');  For Student,   * INSERT INTO Student (ID, FirstName, LastName, Phone)   VALUES (201, 'Ankit', 'Mahato', '9821248422'),  (202, 'Ariana', 'Ghimire', '6543210987');  For Module\_Student,   * INSERT INTO Module\_Student (ModuleID, StudentID, Grade)   VALUES (1, 201, 85),  (2, 202, 90);   1. The DDL statement to create a data structure which will be used to store student data is:   CREATE TABLE Student (  ID INTEGER(10) PRIMARY KEY,  FirstName VARCHAR(255) NOT NULL,  LastName VARCHAR(255) NOT NULL,  Phone VARCHAR(255)  );   1. The DDL statement to create a data structure which will be used to store Module student data is:   CREATE TABLE Module\_Student (  ModuleID INTEGER(10),  StudentID INTEGER(10),  Grade INTEGER(3),  PRIMARY KEY (ModuleID, StudentID),  FOREIGN KEY (ModuleID) REFERENCES Module(ID),  FOREIGN KEY (StudentID) REFERENCES Student(ID)  );   1. It looks like this database design is aimed at a college or university academic management system. Essential information including departments and their locations, teachers and the departments they are assigned to, students and the modules they are enrolled in, grades, and course lengths could all be managed by the system. It offers a framework for methodically monitoring academic relationships and performance.   The design has several drawbacks. It lacks time-specific details like semesters or academic years, making it difficult to track module enrolments over time. It also assumes each module is taught by a single instructor, ignoring cases with multiple instructors or groups. Missing details like instructor qualifications, module schedules, and student enrolment dates limit its functionality. Additionally, it does not support cross-departmental modules or electives, which are essential in larger institutions. Addressing these issues would make the system more scalable and practical |

|  |  |
| --- | --- |
| **Now let’s check how well you have understood so far! 😊** | |
| How did you find attempting this exercise? | Choose one (**X**) |
| Grinning Face I’m **very confident** that I have understood and completed this task successfully |  |
| Slightly Smiling Face I’m **fairly confident** that I have understood and completed this task successfully |  |
| Thinking Face I think I’m **getting there** but I need to **work on understanding this a little more** |  |
| Grinning Face with Sweat I **got a bit stuck** and I will need to **work on understanding this a lot more** |  |

## Exercise 4

The sample database found on Moodle in Week 3 ([mysqlsampledatabase](https://moodle.bcu.ac.uk/mod/resource/view.php?id=7729923)) contains a SQL script called “mysqlsampledatabase.sql”, read through the script, and answer the following questions:

1. What statement is used to create the database within the script?
2. What is the name of the database created within the script?
3. Which statement does the SQL script use to select the database it will run further queries on?
4. Which block of SQL code produces the `customers` table? Explain the SQL code by picking it apart.
5. Which statement is used to put some example records into the `customers` table?
6. Using draw.io or some equivalent diagramming software, draw a diagram which represents just the `customers` table
7. Which block of SQL code produces the `employees` table? Explain the SQL code by picking it apart.
8. Using draw.io or some equivalent diagramming software, draw a diagram which represents just the `employees` table
9. Which statement is used to put some example records into the `employees` table?
10. Which block of SQL code produces the `offices` table? Explain the SQL code by picking it apart.
11. Using draw.io or some equivalent diagramming software, draw a diagram which represents just the `offices` table
12. Which statement is used to put some example records into the `offices` table?
13. Using draw.io or some equivalent diagramming software, draw a diagram containing the `customers`, `employees` and `offices` entities
14. Modify the above diagram to establish the relationship between these entities, transforming it from a diagram that just displays those entities on their own to one that shows entities and their relationships between each other (and ERD)

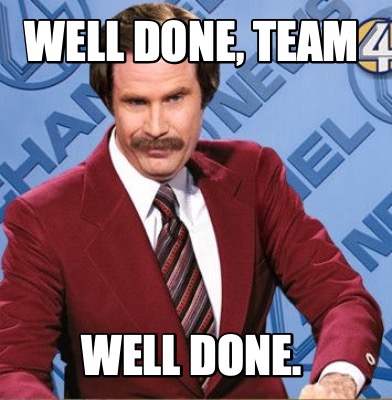
|  |
| --- |
| CookiesThinkingIdea**Food for thought** |
| Let’s deepen our understanding by thinking a little deeper about this exercise and discussing our thoughts in class. |
| In the previous exercises we asked you to pick apart some example ERD **designs** and think about what information they are conveying and what practical software systems could be built from them. In this exercise we have a sample database implementation, and we are asking you to pick this apart before thinking about an equivalent ERD design which would represent that implementation.  Why would we ask you to focus on solving these ERD problems in either direction? Namely what benefits could understanding the following give you in real world scenarios:   * Going from an abstracted Design to Implementation details? * Going from Implementation details to an abstracted Design? |

|  |
| --- |
| **My solution** Idea |
| Record your notes about how you attempted to answer the question / solve the exercise in this section (just edit the text box 😉)   1. The statement used to create the database is:   CREATE DATABASE /\*!32312 IF NOT EXISTS\*/`classicmodels` /\*!40100 DEFAULT CHARACTER SET latin1 \*/;   1. The name of the database is classicmodels. 2. The statement used to select the database is:   USE `classicmodels`;   1. The block of SQL code to create the customers table is:   CREATE TABLE `customers` (  `customerNumber` int(11) NOT NULL,  `customerName` varchar(50) NOT NULL,  `contactLastName` varchar(50) NOT NULL,  `contactFirstName` varchar(50) NOT NULL,  `phone` varchar(50) NOT NULL,  `addressLine1` varchar(50) NOT NULL,  `addressLine2` varchar(50) DEFAULT NULL,  `city` varchar(50) NOT NULL,  `state` varchar(50) DEFAULT NULL,  `postalCode` varchar(15) DEFAULT NULL,  `country` varchar(50) NOT NULL,  `salesRepEmployeeNumber` int(11) DEFAULT NULL,  `creditLimit` double DEFAULT NULL,  PRIMARY KEY (`customerNumber`),  KEY `salesRepEmployeeNumber` (`salesRepEmployeeNumber`),  CONSTRAINT `customers\_ibfk\_1` FOREIGN KEY (`salesRepEmployeeNumber`) REFERENCES `employees` (`employeeNumber`)  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;   * customerNumber: Primary key, ensures each customer has a unique identifier. * customerName, contactLastName, contactFirstName, and other columns: Store customer details like name, phone number, address, and contact details. Fields marked as NOT NULL are mandatory. * salesRepEmployeeNumber: A foreign key that links the customers table to the employees table via employeeNumber. * PRIMARY KEY: Ensures customerNumber is unique. * FOREIGN KEY: Enforces referential integrity with the employees table.  1. The script uses the INSERT INTO statement. An example is:   INSERT INTO `customers`(`customerNumber`, `customerName`, `contactLastName`, `contactFirstName`, `phone`, `addressLine1`, `addressLine2`, `city`, `state`, `postalCode`, `country`, `salesRepEmployeeNumber`, `creditLimit`)  VALUES  (103, 'Atelier graphique', 'Schmitt', 'Carine', '40.32.2555', '54 rue Royale', NULL, 'Nantes', NULL, '44000', 'France', 1370, 21000.00),  (112, 'Signal Gift Stores', 'King', 'Jean', '7025551838', '8489 Strong St.', 'Suite 45', 'Las Vegas', 'NV', '83030', 'USA', 1166, 71800.00);     1. The block for the employees table is:   CREATE TABLE `employees` (  `employeeNumber` int(11) NOT NULL,  `lastName` varchar(50) NOT NULL,  `firstName` varchar(50) NOT NULL,  `extension` varchar(10) NOT NULL,  `email` varchar(100) NOT NULL,  `officeCode` varchar(10) NOT NULL,  `reportsTo` int(11) DEFAULT NULL,  `jobTitle` varchar(50) NOT NULL,  PRIMARY KEY (`employeeNumber`),  KEY `reportsTo` (`reportsTo`),  CONSTRAINT `employees\_ibfk\_1` FOREIGN KEY (`reportsTo`) REFERENCES `employees` (`employeeNumber`)  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;   * employeeNumber: Primary key, uniquely identifies employees. * lastName and firstName: Store employee names. * officeCode: Links to the offices table. * reportsTo: Self-referential foreign key for manager relationships. * jobTitle: Represents the role of the employee.      1. The script uses INSERT INTO statements like:   INSERT INTO `employees` (`employeeNumber`, `lastName`, `firstName`, `extension`, `email`, `officeCode`, `reportsTo`, `jobTitle`)  VALUES  (1002, 'Murphy', 'Diane', 'x5800', 'dmurphy@classicmodelcars.com', '1', NULL, 'President'),  (1056, 'Patterson', 'Mary', 'x4611', 'mpatterso@classicmodelcars.com', '1', 1002, 'VP Sales');   1. The block for the offices table is:   CREATE TABLE `offices` (  `officeCode` varchar(10) NOT NULL,  `city` varchar(50) NOT NULL,  `phone` varchar(50) NOT NULL,  `addressLine1` varchar(50) NOT NULL,  `addressLine2` varchar(50) DEFAULT NULL,  `state` varchar(50) DEFAULT NULL,  `country` varchar(50) NOT NULL,  `postalCode` varchar(15) NOT NULL,  `territory` varchar(10) NOT NULL,  PRIMARY KEY (`officeCode`)  ) ENGINE=InnoDB DEFAULT CHARSET=latin1;   * officeCode: Primary key, uniquely identifies offices. * Other fields like city, phone, addressLine1 store details about office locations.      1. An example:  * INSERT INTO `offices` (`officeCode`, `city`, `phone`, `addressLine1`, `addressLine2`, `state`, `country`, `postalCode`, `territory`)   VALUES  ('1', 'San Francisco', '+1 650 219 4782', '100 Market Street', 'Suite 300', 'CA', 'USA', '94080', 'NA'),  ('2', 'Boston', '+1 215 837 0825', '1550 Court Place', 'Suite 102', 'MA', 'USA', '02107', 'NA');     1. Modified ERD |

|  |  |
| --- | --- |
| **Now let’s check how well you have understood so far! 😊** | |
| How did you find attempting this exercise? | Choose one (**X**) |
| Grinning Face I’m **very confident** that I have understood and completed this task successfully |  |
| Slightly Smiling Face I’m **fairly confident** that I have understood and completed this task successfully |  |
| Thinking Face I think I’m **getting there** but I need to **work on understanding this a little more** |  |
| Grinning Face with Sweat I **got a bit stuck** and I will need to **work on understanding this a lot more** |  |

# Additional Notes

**Extension task**



Finished early? Excellent! 

Use this opportunity to discuss with your lab tutor what you have understood so far and ask them to assign some tasks that you could attempt to further refine your skills and knowledge.

1. This is related to a concept in Engineering called [“Reverse Engineering”](https://en.wikipedia.org/wiki/Reverse_engineering) the process of trying to recreate a design for a system which is as close as possible to the original by understanding how an implementation of that system works [↑](#footnote-ref-2)
2. So we can keep track of whether the answer is primarily derived through your own existing understanding, or based on reviewing the work of someone else – this is not a negative at all when you are starting to learn a new tool or concept! Experienced developers started building their knowledge from somewhere, but you want to clearly reflect on how you are building those skills and knowledge and where you are getting it from [↑](#footnote-ref-3)
3. Referencing is very important, but a simple url, etc is fine for the lab content [↑](#footnote-ref-4)
4. If you need to work on it further then please show us what you have done so far and then continue to work on it further, this allows us to ‘temperature check’ your progress as a group. [↑](#footnote-ref-5)
5. The earlier you tell us, the better 😉 Get our attention and discuss what you need help with so we can advise you [↑](#footnote-ref-6)
6. That’s where the **fun** stuff (and the **money)** is when it comes to Software Development 😉 [↑](#footnote-ref-7)