

**School of Built Environment, Engineering and Computing**

**Level 4 Fundamentals of Databases Work Book**

You can log in to APEX remotely by pasting the following URL link

<https://aet-oracle3.aet.leedsbeckett.ac.uk:8443/ords/> **into your web browser**

**Semester 1 – 2019-20**

**Level 4 Fundamentals of Databases Work Book**

**CONTENTS**

[Session 1 Entity Relationship Diagramming 3](#_Toc19101947)

[Session 2 Relationships between Entities 13](#_Toc19101948)

[Session 3 Logical Database Design 29](#_Toc19101949)

[Session 4 Normalisation 39](#_Toc19101950)

[Session 5 Introduction to the Oracle APEX environment 57](#_Toc19101951)

[Session 6 Inserting data and the implications of foreign keys 71](#_Toc19101952)

[Session 7 Selecting data from a table 75](#_Toc19101953)

[Session 9 More GROUP BY HAVING and Subqueries 83](#_Toc19101954)

[Session 10 Creating Reports using APEX Application Builder 87](#_Toc19101955)

[Session 11 Creating master detail forms using APEX Application Builder 90](#_Toc19101956)

[Appendix A - Film System Specification (ERD and related Tables) 93](#_Toc18940433)

# Session 1 Entity Relationship Diagramming

On completion of this session and its associated activities you should be able to:

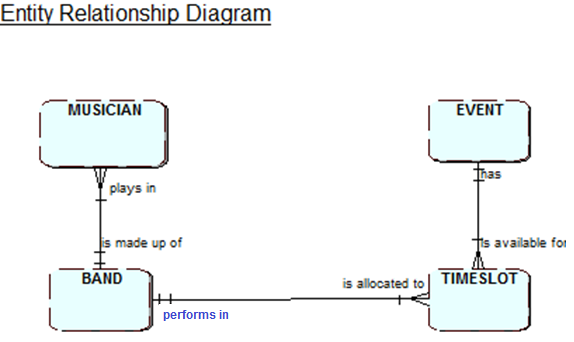
* Understand the purpose of data modelling and its terminology
* Understand what is involved in developing an Entity Relationship Diagram

**1.1 Introduction**

In this module we are going to analyse the information systems in organisations from a data-driven perspective, that is undertaking data modelling in order to produce an Entity Relationship Diagram (ERD). This models the structure of the data used in the processes and stored in the system. We shall use this modelling technique to determine how this data is structured and the relationships between the different types of data.

Data modelling is an important systems analysis technique. It forms the basis for the design of database systems.

An example of an Entity Relationship Diagram is given below, this models entities at a Music events and the relationships between them.

s

The Entities are MUSICIAN, EVENT, BAND and TIMESLOT and are represented as rectangles, the links between them are the relationships which are named in both directions.

**1.2 What is an entity?**

**“An entity is a thing of interest to a system about which data is kept.”** Lejk and Deeks (2002).

In the formal language of database design an “entity” should be called an “entity type”. However, it is usual simply to use the term entity.

Examples of possible entity types are:

In an order processing system:

* customer
* product

In an academic library system:

* student (or borrower)
* book--> B.Name, B.AuthorN, B.ISBN,
* author

Primary key and Identifying attributes --> are same

In a hospital information system:

* ward
* patient
* doctor

In a travel agent’s system:

* booking
* flight
* hotel

Anything about which organisations typically need to hold data could be an entity.

They can be:

**Tangible objects** e.g. **VEHICLE, HOUSE, BOOK**

**Roles played** e.g. **EMPLOYEE, CUSTOMER, TUTOR, STUDENT**

**Customer 🡪**

**Condition:**

* **Important system**

**> Occurance**

**Seld Attributes not verified.**

**Attributes -- Yes**

**C.Name C.ID C.Addrress C.ProductName ----> Attributes**

**1. Anmol 1 Biratnagar mobile**

**2. Netra 2. Banke Ipad**

**3. Jesis**

**4. Sahil**

**5. Ankit**

**Sites or locations** e.g. **OFFICE, WAREHOUSE, ROOM**

**Organisational units** e.g. **DEPARTMENT, TEAM**

**Events or transactions** e.g. **ORDER, PAYMENT, BOOKING**

Note: The objects that occupies a physical space in real life.

**1.4 How do we represent the entity on the ERD?**

Use the Graphical Modelling platform called SemanticPad to model your ERDs - <https://semanticpad.com/Models>

Each **entity type** is modelled on the entity relationship diagram as a named **rectangle** labelled in **uppercase.** As entities are more formally entity types, the name of the entity type is written in the **singular** e.g.

**MUSICIAN not MUSICIANS**

**BAND not BANDS**

Relation and Relationship --> different

**1.4 Entity occurrences**

So far, we have been discussing the identification of entity types. Let us now consider specific **instances** of entity types. For this we use the term **entity occurrence**.

Some examples will help to distinguish between the two terms.

**Entity type Occurrences**

**CUSTOMER** Mr **Netra Kc**, Mrs **Mayalu** etc

**PRODUCT** **Heinz Baked Beans**, **Coca Cola**, etc

**Activity 1**

Which of the following are likely to be **entity types** and which are likely to be **occurrences**?

**BOOK, COURSE**, VENICE, BOB JARVIS, **EXAMINATION, CUSTOMER**

If the item is an entity type, suggest a possible occurrence. If you think it is an occurrence, suggest a suitable name for the entity type. Fill in the table below with your answers.

**Ans:**

**ENTITY TYPE ENTITY OCCURRENCE**

**Book -------------------------------->** Starting out with java

**Course------------------------------>** Bsc.IT

City**----------------------------------> Venice**

Name**-------------------------------> Bob Javis**

**Examination---------------------->** 1st, 2nd term

**Customer-------------------------->** Netra, Jesis

**1.5 Attributes**

The data that is stored for an occurrence will have a structure i.e. it will have a number of attributes. We define an attribute as follows:

**“An attribute is an item of data held about an entity.” (Lejk and Deeks, 2002)**

For example, for the entity type **BAND**, the attributes may include:

* **Band name**
* **Date formed**
* **Genre**

These are examples of the attributes associated with the entity type.

Each entity occurrence will have its value for each attribute, although some attributes may not be required for some occurrences e.g. you would expect all bands to have a name and a date it was formed but some may not fit into a single genre.

Let’s use the example of the entity type **BAND** to examine how all these concepts fit together.

**ENTITY TYPE ENTITY OCCURRENCE**

BAND ACDC

**ATTRIBUTES ATTRIBUTE VALUES**

Band name ACDC

Date formed 1976

Genre Rock

**Activity 2**

Which of the following are likely to be attributes and which are likely to be entity types? Could any be both? **Tick your choice.**

|  |  |  |
| --- | --- | --- |
|  | **ATTRIBUTE** | **ENTITY TYPE** |
| CUSTOMER |  | Yes |
| DATE | Yes |  |
| NAME | Yes |  |
| DESTINATION |  | Yes |
| COLOUR | Yes | Yes |
| VENUE |  | Yes |

**1.6 Four tests of an entity type:**

When a database is being designed, the first decision is which entities to include. The following four tests will help us to decide whether something being considered is actually an entity type that should be included on the ERD.

|  |  |
| --- | --- |
| **Test one** | The object should be of importance in the system being studied |
| **Test two** | There should be data attributes that can be associated with the entity |
| **Test three** | There should be more than one occurrence of this entity type |
| **Test four** | It should not be an attribute of an entity |

**1.7 Modelling entities – the key steps:**

**One initial approach for analysing a system might be to consider identifying candidate entity types**

A “candidate entity” is used to describe something that we might consider as an entity type in the database system being developed.

Taking the system description we can start by **underlining or drawing a box** around all nouns (as these might be things data is needed to be stored about). These are the **Candidate** **entities**.

We put them in a list and then eliminate any that do not pass the **four tests** described above.

**Activity 3**

Circle all the candidate entities in the following brief system description. Then cross out any of the candidates that fail one or more of the four tests of an entity.

*“Students on full-time courses at ~~Leeds Beckett~~ take some core and some elective modules.*

*Most students have a ~~mobile phone~~ and therefore a ~~mobile phone number.”~~*

**Ans: The words on red colour and underline margin entity type.**

**Identify some attributes for each identified entity type**

At this stage we may not be able to identify every attribute for each entity. But there is one particular type of attribute that we need to ensure is included in the list of attributes for an entity. This is the **entity identifier**.

Each entity type in an ERD should be assigned an identifier. This is an important attribute as it is used to **uniquely identify** **each occurrence of the entity type**.

For example:

**A CAR will have a car registration number.**

Sometimes, during the identification of entity types, a natural identifying attribute will be obvious. However, if there isn’t a natural candidate entity, we can introduce an **artificial one** e.g.., a student id for a STUDENT entity. Occasionally there may more than one possible identifier e.g. for an EMPLOYEE

**ATTRIBUTES**

**Employee No**

Emp Name

**Nat Ins No**

Salary

both **Employee No** and **Nat Ins No** are candidate identifiers.

**Activity 4**

Which candidate identifier from the above would you chose and why?

**Employee No is the candidate identifier because every employee has different employee number which represent them and helps to identify.**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

To note the attribute which is the entity identifier, underlining should be used e.g.

**ATTBIBUTES**

Employee No

Emp Name

Nat Ins No

Salary

**Activity 5**

Underline what you think would be the likely identifying attribute in each of the following two examples of entity types where some attributes have been identified?

|  |  |  |
| --- | --- | --- |
| **Entity** | **SCHOOL** | **PRODUCT** |
| **Attributes** | Headteacher | Make |
|  | School\_Name | Model |
|  | Address | Size |
|  | Education Authority | Weight |
|  | Key\_Stages | **Product No** |
|  | **School Id** | Colour |

Remember wherever possible use an existing attribute for the entity identifier and always make sure that it will allow you to uniquely identify every occurrence of an entity type.

**Activity 6**

An estate agent’s system includes a **HOUSE** entity type and an **OFFER** entity type (to record the fact that a definite offer has been made for a house). For each of these two entity types:

* Suggest some suitable attributes
* Define an identifying attribute for the entity type using underlining, remember if there is no natural entity identifier you should create one
* Give an occurrence of the entity type (by suggesting a value for each of the attributes you have suggested)

**Entity: HOUSE**

Attributes: **H.Name,H.Number,H.Location,H.Size,H.Color,H.Owner**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Occurrence: **Flat, Apartment, Guesthouse, Hotels,**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Entity: OFFER**

Attributes: **O.Time, O.Type,** **Applying No. For Order**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­

Occurrence: **New Year, Dashai, Tihar, Holi,**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1.8 Entity:**

To help us to clearly understand and communicate the meaning of the entities we have modelled, a short, meaningful description is given of it. This is the start a “**data dictionary**” which can be shared with other system developers and with clients.

**[Note: model and data dictionary are same.]**

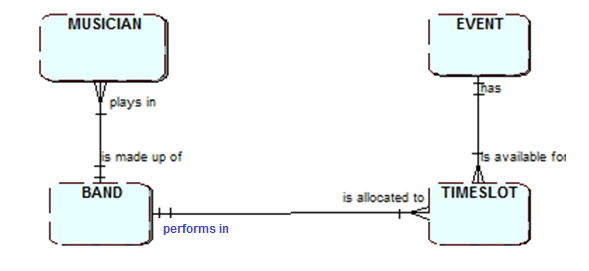
Here are some examples of the full definition of entity types, note the identifying attribute has been underlined.

|  |  |  |
| --- | --- | --- |
| **Entity type** | **Attributes** | **Entity description** |
| STUDENT | **Student Id,** Name, Address, Post code | A student registered at the university |
| COURSE | **Course code,** Course Name | An approved course of study at the university |
| MODULE | **Module code,** Module title, Semester running | A module available as part of an approved course of study |

**[Note: The data dictionary is normally held within a CASE tool. This provides a central reference point and allows for easy searching, amendment and reporting.]**

**1.9 Modelling of relationships between entity types:**

The second component in Entity Relationship Diagrams is the modelling of relationships between entity types.



This ERD has 3 relationships, **one between the entities MUSICIAN and BAND**, **one between BAND** **and** **TIMESLOT** and **one between TIMESLOT and EVENT.**

We will explore this in more detail next session.

**Session 1 Tutorial Exercises**

For the following three scenarios:

* Highlight the candidate entities then eliminate any which are not entities
* Underline the identifying attribute
* For each surviving entity, complete the table and underline the identifying attribute

**Scenario 1 – University structure**

A university is comprised of a number of schools and each school has a unique name, a Dean and a Secretary. Each school is divided into 2 or 3 Academic Groups each of which have a unique Academic Group name and a Group Head. Each university Lecturer is based in an Academic Group and their name, their telephone extension and email address are recorded.

|  |  |  |
| --- | --- | --- |
| **Entity type** | **Attributes** | **Entity description** |
|  |  |  |
|  |  |  |
|  |  |  |

**Scenario 2 – Sports hire**

A sports centre allows its members to make bookings for use of its sun beds. When a booking is made, a particular sun bed is reserved for the use of the member at the requested date and time. The sports centre also records, for reasons of fire safety, which member is using which sun bed at the current moment (as this can occasionally be different from the one that has been booked).

|  |  |  |
| --- | --- | --- |
| **Entity type** | **Attributes** | **Entity description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Scenario 3 – Pop Idol**

The Pop Idol production team runs a national contest for aspiring pop stars. They organise auditions on advertised dates at venues around the UK. Approximately 20,000 hopeful contestants in total attend the contest, usually going along to one of the auditions organised at the nearest venue. A panel of judges is assembled to adjudicate at the audition. There are four judges currently available but, given their recording commitments at home and overseas, they are not able to be part of the panel at every audition.

|  |  |  |
| --- | --- | --- |
| **Entity type** | **Attributes** | **Entity description** |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**Check the boxes when you have completed the following Session 1 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 1 Reflection

|  |
| --- |
|  |

* Session 1 Entity Relationship Activity Leeds Laser Clinic

|  |
| --- |
|  |

* Do Session 1 quiz – Entities and Attributes

Session 2 Relationships between Entities:

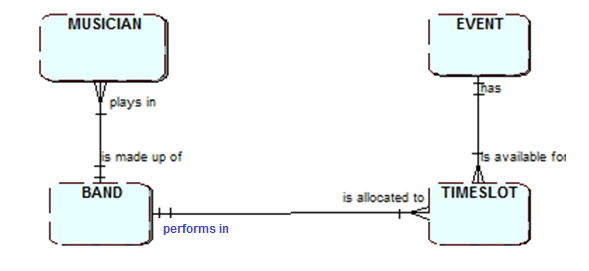
On completion of this session and its associated activities you should be able to:

* Understand what is meant by a relationship between entity types
* Apply the entity relationship modelling notation in example situations
* Name each relationship and determine its cardinality and optionality
* Draw ERDs using **SemanticPad** for the entities and relationships in a scenario

**2.1 Introduction**

In the last session we started to analyse information systems from a data-driven perspective using Entity Relationship Diagrams (**ERDs**). You should now be able to identify **entity types**, **attributes**, decide on an **identifying attribute**, identify **occurrences** of entity types and be aware of **relationships between entity types**.

In this session we shall concentrate on the modelling of relationships between entity types.



This ERD has 4 entities with 3 relationships, one between the entities MUSICIAN and BAND, one between BAND and TIMESLOT and one between TIMESLOT and EVENT.

**2.2 Relationships between entities**

What do we mean by “relationships” in this context? A MUSICIAN plays in a BAND. A TIMESLOT is allocated to a BAND etc. Strictly speaking there is a relationship between one occurrence of an entity with an occurrence of another entity e.g. Angus Young (an occurrence of MUSICIAN) plays in ACDC (an occurrence of BAND).

Let’s take another example and consider the entity types TUTOR, SCHOOL, MODULE and STUDENT.

We can identify a few relationships between these entity types. Some examples are:

* A TUTOR is a member of a SCHOOL
* A TUTOR teaches on MODULES
* A STUDENT studies on MODULES.

Let us look at the entities TUTOR and MODULE in an Entity Occurrence Diagram.

One occurrence of TUTOR Lesley Earle has a relationship with many occurrences of MODULE

**TUTOR MODULE**

**Teaches on**

**Project**

**Lesley Earle Databases**

**Human Computer Interaction**

**Meg Soosay**

**Databases**

**Project**

In this particular example we have one occurrence of the TUTOR entity, Lesley Earle, who teaches on 2 occurrences of MODULE and another occurrence of TUTOR, Meg SOOSAY, who teaches on 3 occurrences of MODULE. We would say a TUTOR Teaches on one or many MODULES.

Here we have looked at the relationship between the two entities from the perspective of the TUTOR. We must now consider the relationship from the viewpoint of the other entity, the MODULE. We might name the relationship as follows: A MODULE ‘is taught by’ TUTORS.

This can be illustrated at the occurrence level by the following diagram:

**MODULE TUTOR**

**is taught by**

**Project Lesley Earle**

**Meg Soosay**

**Databases Lesley Earle**

**Meg Soosay**

**Human Computer Interaction Meg Soosay**

We would say a MODULE is taught by one or many TUTORS.

**Activity 1**

Which of the following are relationships between entity types and which are relationships between occurrences?

**RELATIONSHIPS BETWEEN: (Tick appropriate column)**

|  |  |  |
| --- | --- | --- |
|  | ENTITY TYPE | OCCURRENCE |
| A). EMPLOYEE claims EXPENSES |  |  |
| B). ASDA sells MILK |  |  |
| C). HAZEL SMITH teaches EPD |  |  |
| D). CUSTOMER buys PRODUCT |  |  |

**Activity 2**

1. The following table shows an extract from the records of films system. It shows the DIRECTOR and FILM(s) that they direct.

i) From the perspective of the DIRECTOR we have:

**DIRECTOR directs FILM**

Robert RedfordLions for Lambs

A River Runs through it

Steven SpielbergJAWS

ET the Extra Terrestrial

Guy RitchieSnatch

How is each occurrence of DIRECTOR related to occurrence(s) of FILM?

A DIRECTOR directs …..……….. FILM(s)

ii) From the perspective of the FILM we have:

**FILM is directed by DIRECTOR**

Lions for Lambs Robert Redford

A River Runs through itRobert Redford

JAWSSteven Spielberg

ET the Extra TerrestrialSteven Spielberg

SnatchGuy Ritchie

How is each occurrence of FILM related to occurrence(s) of DIRECTOR?

A FILM is directed by ………….. DIRECTOR(s)

The relationship has relevant verbs to describe the association between the two entities in each direction. The relationships of a DIRECTOR directs one or many FILM(s) and a FILM is directed by one DIRECTOR are represented in the following way:



The relationship must be described and labelled in both directions.

**Activity 3**

Look again at the relationships between the entity types discussed in Activity 1. These relationships are named from the viewpoint of the first entity. Now name the relationship from the standpoint of the second entity.

EXPENSES EMPLOYEE

PRODUCT CUSTOMER

**2.3 Cardinality of relationships**

The number of occurrences one entity is related to in another entity, is called Cardinality. There are 3 main cardinalities 1:M, M:N and 1:1.

**1:M Relationships**

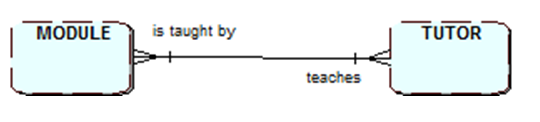
There is said to be a one to many relationship between FILM and DIRECTOR which is normally represented by the shorthand notation 1:M.



Note a “crow’s foot” drawn at the “many” end of the relationship on an ERD.

**M:N Relationships**

Now consider the relationship between TUTOR and MODULE. If we examine this from the perspective of a single TUTOR, we can state that a TUTOR teaches on one or more MODULES. Then, looking at the relationship from the perspective of a single MODULE, we can also state that a MODULE is taught by one or more TUTORS. This is an example of a second type of cardinality the many to many or M:N (sometimes written as M:M).

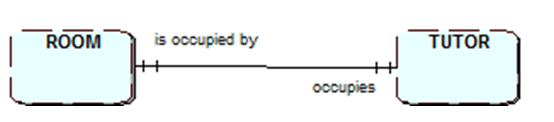


**A many to many (M:N) relationship**

This time there is a crow’s foot at both ends because they are both “many” ends.

**1:1 Relationships**

Finally, there is one last type of cardinality we need to examine. This is the one to one relationship. If, in this system, a tutor occupies just one room and a room is occupied by one, and only one, tutor, we can show this as follows on the ERD:



**A one to one (1:1) relationship**

**Activity 4**

Draw an Entity Relationship Diagram showing the four entities and the three relationships for the following:

a). An ARTIST is featured on one or more TRACKS

A TRACK features one or more ARTISTS

b). A TRACK appears on an ALBUM

An ALBUM contains one or more TRACKS

c). An ALBUM is publicised by a COVER

A COVER publicises an ALBUM

**2.4 Optionality of relationships**

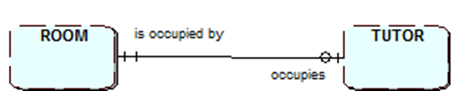
**Optionality for one to one relationships**

Membership of a relationship is not always mandatory, it might be optional. In the previous example, we modelled the situation in which a ROOM is always occupied by a TUTOR. But it may be that some rooms are not staff offices. Consequently, there will be at least one occurrence of the entity ROOM that is not related to an occurrence of the entity TUTOR.

Can the same be said of the relationship from the perspective of the TUTOR?

No, its membership of the relationship is mandatory. All occurrences of TUTOR occupy a ROOM.

The fact that a relationship is optional is denoted on the ERD by including a small circle on the relationship.



**A one to one (1:1) relationship with optionality**

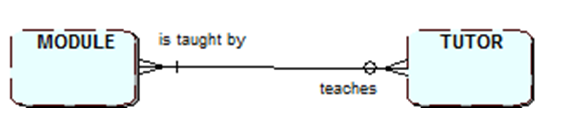
The circle might be considered as the letter O for optional but is best considered as a zero.

The diagram shows that a ROOM is occupied by zero or one TUTOR. From the TUTOR end of the relationship, this diagram shows that a TUTOR (always) occupies one ROOM.

**Optionality for many to many relationships**

Now look again at the relationship between TUTOR and MODULE, a many to many relationship: a TUTOR teaches on one or more MODULES and a MODULE is taught by one or more TUTORS. Could optionality be present in this relationship?

To decide on the answer to this we may be given the information that the University has new modules that have not yet been assigned to TUTOR(s). Consequently, at any time, there may be at least one occurrence of MODULE that has zero TUTOR(s) attached to it.

****

**A many to many (M:N) relationship with optionality**

We read the relationship as follows:

One MODULE is taught by zero, one or many TUTORS

One TUTOR teaches on one or many MODULES.

**Activity 5**

Let’s return to the Entity Relationship Diagram you produced in Activity 4. The relationships have now been amended to include optionality in some areas. Show how this affects your ERD.

a). An ARTIST is featured on zero, one or more TRACKS

(Justification: The artist is placed on the record company’s books prior to recording any material.)

A TRACK features one or more ARTISTS.

b). A TRACK appears on zero or one ALBUM

(Justification: some tracks are recorded but never make it onto an album).

An ALBUM contains one or more tracks

c). An ALBUM is publicised by a COVER

A COVER publicises zero or one ALBUM

(Note a cover may be commissioned by the record company but not be used for the album).

**Use SEMANTICPAD for producing your ERDs.**

**Activity 6**

Use SEMANTICPAD to create ERDs for the following:

1. A university comprises of a few faculties. Each faculty is divided into schools. Each university lecturer is based in a school.
2. A sports centre allows its members to make bookings for use of its sun beds. When a booking is made, a particular sun bed is reserved for the use of the member at the requested date and time. The sports centre also records, for reasons of fire safety, who is using which sun bed at the current moment (as this can occasionally be different from the one that has been booked).
3. The Pop Idol production team runs a national contest for aspiring pop stars. They organise auditions on advertised dates at venues around the UK. Approximately 20,000 hopeful contestants in total attend the contest, usually going along to one of the auditions organised at the nearest venue. A panel of judges is assembled to adjudicate at the audition. There are four judges currently available but, given their recording commitments at home and overseas, they are not able to be part of the panel at every audition.

**2.5 Many-to-Many Relationships that have data associated with them**

In the ERD containing the three entities TUTOR, MODULE and STUDENT we would have a relationship between TUTOR and MODULE as described above and one between MODULE and STUDENT which is also M:N and is mandatory both ways. It is likely that this, type of relationship will have data associated with the relationship itself that will need to be captured. This is data that cannot be recorded as attributes for either of the entities involved in the relationship.



**The many to many (M:N) relationship between STUDENT and MODULE**

**Activity 7**

Let’s consider one occurrence of STUDENT ‘you’ and ‘Fundamentals of Databases’ as one occurrence of MODULE. What data would we be likely to need to record about your study of the database module?

­­­­­­

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

As the relationship currently stands above, it is hiding the data we’ve just identified behind the many-to-many relationship. We need to alter the ERD so that this data can be modelled. We do this by resolving the many-to-many relationship and capturing the data in a third entity type. This entity type connects together a single occurrence of each of the other two entities, one module for each student.

There is a standard rule for resolving a many-to-many relationship under these circumstances.



**A many to many relationship between A and B**

If we have identified that there is extra data that is represented by the M:N relationship we resolve this by introducing a third entity, C, which has a one-to-many relationship with each of the other two entities.

The ERD now becomes:



**Activity 8**

Use this approach in SEMANTICPAD to resolve the many-to-many relationship between STUDENT and MODULE. Make sure that you show the cardinality of the relationships correctly. You need to choose a suitable name for the new entity.

**Activity 9**

A situation that we often meet when modelling the data involved in sales order processing is as follows:

1. A CUSTOMER ORDER is comprised of one or more PRODUCTS and a PRODUCT is included in zero, one or more CUSTOMER ORDERS. The relationship between CUSTOMER ORDER and PRODUCT is therefore:



**The many to many relationship between**

**CUSTOMER ORDER and PRODUCT**

Resolve this many-to-many relationship by introducing a third entity type. Name this new entity as ORDER ITEM.

b) In a Film system an ACTOR can work on one or many FILMS, a FILM can have many actors working on it. There is data associated with this relationship, it is the CONTRACT data.

i)Draw a diagram which resolves the many to many relationship between ACTOR and FILM by including the entity CONTRACT.

ii). Suggest some suitable attributes for ACTOR and for FILM.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

iii). Suggest some suitable attributes for CONTRACT. (Note: this new entity is the link between a specific occurrence of ACTOR and a specific occurrence of FILM).

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**2.7 Assigning the new entity an identifier**

All entities on an ERD must have a unique identifying attribute, as we discovered in session 1.

What will be the identifier of the new entity?

The entity identifiers that we have met so far have comprised of a single attribute. The Identifier for a **MODULE** is its **module code**. The identifier for **STUDENT** is its **student ID**. It is unlikely that this new link entity will have a single, natural candidate attribute for this. It is more likely that there will be no obvious identifier. In this situation, the identifier can usually be formed from the identifiers of the original pair of entities combined together.

Let’s apply this to the Student Records system that we modelled in Activity 2. The identifier for STUDENT is **Student ID**. The identifier for MODULE is **Module Code**. Consequently, the identifier for STUDY is **Student ID and Module Code**. It is the combination of these two attributes that uniquely identifies a single occurrence of the entity STUDY.

Now, if we examine some occurrences of these entities, we can clearly show that the combined identifier is needed to uniquely locate a specific STUDY of a particular MODULE by a named STUDENT.

What issue might arise relating to the identifier if a student could study a module more than once?

**Activity 10**

Consider the following situation:

The Beckett Park Playhouse contains two theatres: the main theatre and a studio theatre. It allows theatregoers to make telephone bookings for performances of its current productions. The caller is asked for the performance date and the theatre in which it is to take place. No more than one performance per day is staged at each theatre. A booking reference number is quoted to the caller. Theatregoers also enquire about current productions. In response to their enquiries, information about the actors appearing in them, the running time, etc is supplied.

The entities in this case study are THEATRE, THEATREGOER, BOOKING, PERFORMANCE, PRODUCTION and ACTOR.

a). Suggest TWO possible attributes for each entity. (One of these should be its identifier).

THEATREGOER \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

BOOKING \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PERFORMANCE \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

PRODUCTION \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

ACTOR \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b). Suggest relationships between these entity types. (Remember to name the relationship from both entities involved, and also to include the cardinality and any potential optionality). Draw the ERD for this case study using SEMANTICPAD.

c). There is a one many-to-many-relationship.

Resolve this by introducing an appropriate new link entity.

**Session 2 Tutorial Exercises**

Please use SEMANTICPAD for these exercises.

1. In a telephone banking call centre, an operator deals with many different customers and, over time, a customer speaks to many different operators. This is modelled on an ERD as follows:



a) Resolve this many-to-many relationship, by redrawing the diagram in SEMANTICPAD to include an appropriate link entity.

b) Identify THREE attributes of this new entity, including the identifier.

2. A fleet of vehicles is owned and operated by a large company and the company wishes to hold data about its vehicle operations. There are many different classes of vehicle, and the class of vehicle determines how often it is serviced, who may drive it, etc. An employee is permitted to drive certain classes of company vehicle, depending on what qualifications he or she holds (HGV, for example). Some of the drivers supervise other drivers but not all drivers are supervised.

Vehicles are regularly serviced and, at each service, details of the mileage, the type of service, etc are recorded. Occasionally an accident occurs involving one of these vehicles. This may result in an insurance claim being made. The driver may subsequently be barred from driving some or all classes of vehicle.

An analysis of the entity types required has taken place, resulting in the following being chosen.

DRIVER

VEHICLE

CLASS OF VEHICLE

SERVICE

ACCIDENT

INSURANCE CLAIM

a) Identify the relationships between these entities as described in the above definition of the system requirements

b) Draw an entity relationship diagram for the system in SEMANTICPAD which includes appropriate cardinality, optionality and naming of relationships.

**Challenging Activities for Session 2**

1. **Modelling entities in which occurrences are related to each other**

We have concentrated on identifying and modelling relationships between pairs of entity types. Most of these relationships will be one-to-many, a few might be many-to-many and, occasionally, some might be one-to-one. We have also discovered how to resolve many-to-many relationships that contain data which is of interest in the situation being modelled.

We will also encounter entities that are related to themselves. To be more precise, occurrences of the entity type are related to other occurrences of the **same** entity type. This is called a **recursive relationship**.

Consider the entity type STAFF in this university. There are over 3000 staff employed at Leeds Beckett, giving us over 3000 occurrences of the entity. The Vice-Chancellor manages the Deans. Each Dean manages several Group Heads. The Dean of CCTE, for example, is line manager to two Group Heads, who, in turn, manage the teaching staff.

We can observe a hierarchy relationship emerging within this single entity type. This can be represented pictorially using a hierarchy diagram, as follows:



**The “manages” relationship between occurrences of STAFF**

This hierarchy diagram clearly shows us that an occurrence of the entity STAFF, say Vice-Chancellor, manages one or more other occurrences of STAFF. Another occurrence of the entity, the Dean of CCTE, also manages one or more other occurrences of the same entity.

How do we model this recursive relationship on an ERD? We simply draw a relationship line both starting and finishing at the entity, as follows:



**Modelling a recursive relationship**

Reading this relationship from the ‘one’ end, we can state that a member of STAFF manages one or more other members of STAFF. Reading the relationship from the other end, we can state that a member of STAFF is managed by one and only one other member of STAFF.

Does this accurately model the ‘manages’ relationship between occurrences of staff in this university? Not quite because optional participation in the relationship is also present. The Vice-chancellor holds the chief academic officer responsibility. He is not managed by another member of staff. (His activity is scrutinised by the Board of Governors, but the Governors are not occurrences of the entity STAFF). Similarly, there are numerous individual lecturing staff who do not manage any other staff. Consequently, we need to represent this optionality on the ERD:



**Recursive relationship with optionality**

This version of the ERD covers all possible variations of the ‘manages’ relationship. A member of STAFF manages zero, one or more other STAFF. A member of STAFF is managed by zero or one other member of STAFF.

**Exercise**

The Business Information Systems and Business Information Technology courses are two of the courses provided by the University of Hotton. Each course is comprised of a few modules that are studied by students. Some modules, at one level of study, are pre-requisites for studying particular modules at the next level of study. For example, students wishing to study Systems Methods at level 2 must have passed Systems Modelling at level 1. Systems Modelling is a pre-requisite for taking Systems Methods. It is also a pre-requisite for the Database Development and Implementation module. Some modules may have no pre-requisite requirements.

State any assumptions you have made in building your ERD.

1. **Sub Types**

There is sometimes a need to show sub types of an entity this is usually the case when the sub types contain different sets of data attributes. The main entity (Super type) contains attributes that are common to all of the entity’s sub types. Note you can have sub types of a subtype but not a good idea to have too many.

The use of sub types also provides the opportunity to show relationships that only relate to a specific sub type rather than the super type e.g. as shown below if a STUDENT is taught by a TEACHING academic but not a RESEARCH academic.

e.g. An employee at the university could be modelled using sub types to show the different categories of employee.



1. **Exclusive Relationships**

Sometimes two or more relationships are mutually exclusive e.g. a holiday may use a coach or a flight but not both



**Check the boxes when you have completed the following Session 2 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 2 Reflection

|  |
| --- |
|  |

* Session 2 Activities – 3 questions to complete on SEMANTICPAD

|  |
| --- |
|  |

* Entity Relationship Modelling of Door Step Sandwiches

|  |
| --- |
|  |

* Do Session 2 quiz – Relationships between Entities

Session 3 Logical Database Design

On completion of this session and its associated activities you should be able to:

* Convert an ERD into a logical database design
* Set Primary and Foreign keys
* Provide a logical design for relationships of different cardinalities

**3.1 Introduction**

In this session we shall look at how we can turn a data model - in the form of an Entity Relationship Diagram - into a design suitable for implementation. We will produce a design for a relational database such as Oracle™. A relational database is a set of tables containing data in attributes. Each table (called a relation in logical design) contains one or more attributes. Each row contains a unique instance of data for the defined attributes.

**3.2 Recap**

***Entities and relationships***

|  |  |
| --- | --- |
| Entity | Something that data is to be stored about |
| Attribute | A property of an entity |
| Occurrences | Specific instances of entities |
| Identifying attribute | An attribute that uniquely identifies an entity occurrence, it may have a combination of attributes. We shall now also call this a **key** |
| Relationship | An association between two entities |
| Cardinality | The number of occurrences in one entity that can have a relationship with the occurrences in another entity |
| Optionality | Participation in a relationship |

***Terminology about the cardinality of a relationship***

|  |  |
| --- | --- |
| one to one | 1:1 |
| one to many | 1:M or 1:m or M:1 or m:1 |
| many to many | M:N or m:n (or M:M) |

**3.3 Keys**

The following terminology is used in association with keys in relational databases:

|  |  |
| --- | --- |
| **Key** | **Description** |
| Primary Key | Uniquely identifies an entity occurrence in a relation/table |
| Candidate key | A ‘possible’ primary key (several candidate keys may exist for a relation) |
| Compound key | A key consisting of two or more attributes |
| Foreign key | An attribute - or combination of attributes - of one relation which exists as a primary key in some other relation |

NOTE: When implementing the database, all occurrences MUST have a Primary key value and where there is a Foreign key defined it MUST also have a value for every occurrence.

**3.3 Database Design Stages**

There are three stages in designing a database:

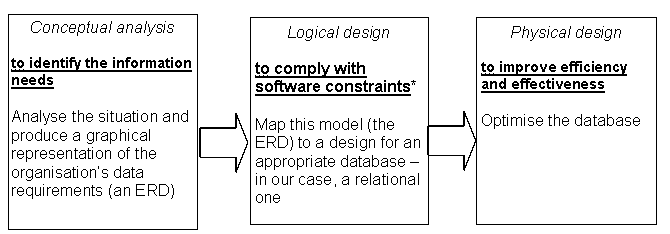
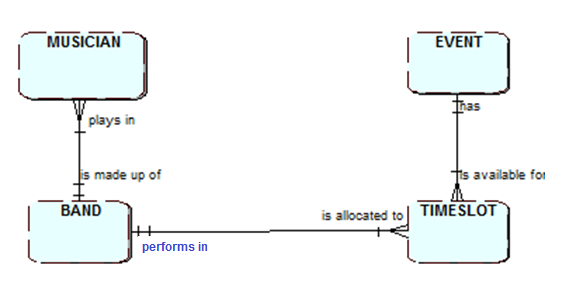


DIAGRAM WRITTEN LIST SQL CODE

Logical design takes an **ERD** and through a mapping process produces **Relations** which then become **tables** when implemented e.g. in ORACLE.

**3.4 Relational Databases**

The name relational database derives from the word relation. A relational database stores all its data inside tables. All operations on data are done on the tables themselves. The standard user and application program interface to a relational database is the structured query language (SQL). **NOTE**: A relation is NOT the same as a relationship.



**Step 1**

The mapping process of Logical Design initially takes each **Entity** and represents it as a **Relation**. A definition of a relation is the ‘logical’ representation of an entity in terms of its attributes and keys which is shown in text form with the unique identifier being underlined but now referred to as the **primary key** e.g.

MUSICIAN ( Musician\_Id, Name, Address, DateOfBirth, Mobile......... )

BAND (Band\_No, Band-name, Date\_formed, Genre,….)

EVENT (Event\_No, Title, Date, Location, …..)

TIMESLOT (Timeslot#, Start\_time, End\_time,….)

**Step 2**

So far we have only captured the entities from the ERD, we now need to capture the relationships. There are a set of rules to follow depending on the cardinality and optionality of the relationship, they use foreign keys and sometimes result in a new relation being needed.

**Foreign Key definition:**

1. A Foreign Key is an attribute which is a Primary Key in another relation or table.
2. When implemented in a Relational Database a foreign key cannot take a NULL (empty) value.

**1:M Relationships**

For the one-to-many relationship between MUSICIAN and BAND through step 1 we have:

BAND(Band\_No, Band\_name, Date\_formed, Genre,….)

MUSICIAN (Musician\_Id, Name, Address, DateOfBirth, Mobile......... )

Step 2 needs the implementation of the 1:M relationship between these two relations, to do this we copy the primary key of the one end, Band\_No in BAND into the relation at the many end MUSICIAN as a foreign key (represented by italics): **Band\_No becomes a foreign key in MUSICIAN**, foreign keys are identified by being in italics**.**

BAND(Band\_No, Band\_name, Date\_formed, Genre, ….)

MUSICIAN(Musician\_Id, Name, Address, DateOfBirth, Mobile, *Band\_No*......... )

In handwritten text you use a dotted underline for a foreign key. A key that is both a primary key and a foreign key can be underlined twice once with dotted and once with normal underline.

**Activity 1**

Can you think of the reason we must put the foreign key into the MUSICIAN relation and not the other way around?

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Activity 2**

Now implement the same approach for the other two 1:M relationships in the Band example.

NOTE: You will need 2 foreign keys in the same table, don’t forget to use italics or a dotted underline to indicate the foreign keys.

BAND(Band\_No, Band-name, Date\_formed, Genre, ….)

MUSICIAN(Musician\_Id, Name, Address, DateOfBirth, Mobile, *Band\_No*.. )

EVENT(Event\_No, Title, Date, Location ,…..)

TIMESLOT(Timeslot#, Start\_time, End\_time, , ,….)

**M:N Relationships**

In session 2 we discussed the fact that a relationship of type M:N may have data associated with it. We showed how this could be dealt with by inserting a link entity. In the logical design for any M:N relationship, whatever the optionality a new relation will need to be created.

For example, the following ERD:



We will just use **skeletal relations (containing primary and foreign keys only):**

**Step 1** – Each entity becomes a relation

TEACHER (Teacher#, .......)

PUPIL (Pupil#, ........)

**Step 2 –** Implement the many to many relationship by introducing a new relation, this must have a compound key taken from the combination of the keys in the relations on each side of the relationship

TEACHER (Teacher#, .......)

PUPIL (Pupil#, ........)

TUITION (*Teacher#* , *Pupil#* )

Note the compound primary keys are also both foreign keys. Both are needed together to identify uniquely occurrences that will exist in the resulting relationship table.

**Activity 2**

a) Consider the following ERD:



What would you choose for the following:

Primary key for entity type STUDENT: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Primary key for entity type MODULE: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Primary key for entity type MARK: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

b) Which of these applies to the key for MARK? a) Compound Key b) Candidate Key

**1:M relationships with and without optionality**

Many to one or one to many relationships **without optionality** such as



require only two relations, as has already been explained.

The primary key of the “one” end is posted into the table at the “many” end as a foreign key:

WARD ( Ward#, .... )

PATIENT ( Patient#, ........, *Ward#,* .... )

You can still do this if there is optionality at the “many” end.

If, however, there is **optionality at the “one” end** this will not work!

Consider for example the following ERD:



If we used the simple approach above and we had an out-patient who was not assigned to a ward, we would have no value to give to Ward# in the PATIENT table. The value would be what is referred to as **NULL**. This goes against the definition of foreign keys as seen earlier. To resolve this, we create an additional relation in the logical design:

WARD ( Ward#, .... )

PATIENT ( Patient#, .... )

PATIENT\_WARD ( *Patient#,* *Ward#* )

Note that Patient# should be used as the primary key for the new relation because a patient can only be allocated to one ward. Ward# and Patient# are both, separate, foreign keys. With this arrangement an entry is only made in the PATIENT\_WARD table when a patient is allocated to a ward. Other patients not on a ward e.g. out-patients will only have data entered into the PATIENT table.

**1:1 relationships without optionality**

We may have a one to one (1:1) relationship which is **mandatory at each end** for example



This is a special case and will only need one table in the final database. At the logical design stage, the two entities are effectively merged into one relation with just a single primary key:

EMPLOYEE (Employee No,...... , *Car\_No*, ...... ) OR CAR (Car\_No,….., *Employee\_No,* ..)

Which we choose is decided upon by considering the focus of the system.

* Is it Human Resources? If so, choose the first alternative.
* Is it vehicle inventory or maintenance? Choose the second.

**1:1 relationships with optionality**

If we have a one to one relationship with **optionality at one end**, for example



then a relation must be created for each of the entities. The key of the entity at the non-optional end is posted into the relation for the optional entity as a foreign key:

EMPLOYEE ( Employee No,..... )

CAR ( Car\_No, ..... , *Employee No*, ...... )

**1:1 relationships with optionality at both ends**

If, however, we have a one to one relationship with **optionality at both ends**, for example



then 3 relations are needed, one for each entity and one to express the relationship between the cars and their users:

EMPLOYEE ( Employee No,..... )

CAR ( Car\_No, ..... )

CAR\_ALLOCATION ( *Employee\_No*, *Car\_No* )

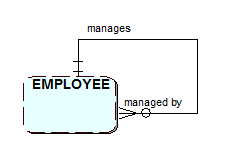
Note that either of the original identifiers could have been used as the primary key for the third relation.

**Challenging Activities for Session 3**

**Recursive Relationships**

When we have an entity type which can have occurrences that relate to other occurrences of the same entity type.

For a 1:m with or without optionality at the “many” end of the relationship, include a foreign key attribute to hold the primary key value of the one end record. i.e the employees manager’s emp\_no.

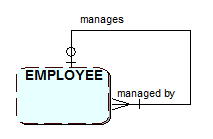


EMPLOYEE (Emp\_no, emp\_name)

Becomes

EMPLOYEE (Emp\_no, emp\_name,*mgr\_emp\_no*)

For a 1:m with optionality at the one end we need to introduce a new relation to avoid holding a NULL foreign key value. In this example not all employees have a manager. Therefore, we introduce a manager relation which will hold the employee’s emp\_no(Primary key) and their manager’s emp\_no (Foreign key) for each employee that does have a manager.

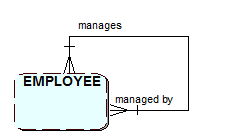


EMPLOYEE (Emp\_no, emp\_name)

And

MANAGER (*Emp\_no, mgr\_emp\_no*)

Where there is a m:m relationship a new relation is used to store the employee’s emp\_no and their manager’s emp\_no. Both Foreign Keys are needed to form a compound Primary Key



EMPLOYEE (Emp\_no, emp\_name)

And

MANAGER (*Emp\_no, mgr\_emp\_no*)

**Exclusive relationships**

In a mutually exclusive relationship, which is shown as an exclusive arc, the optional entities e.g. Flight and Coach have a Foreign Key (holiday \_code). This avoids the Null key problem.

HOLIDAY (**holiday\_code,…)**

FLIGHT ( **flight\_id**, *holiday\_code*,…)

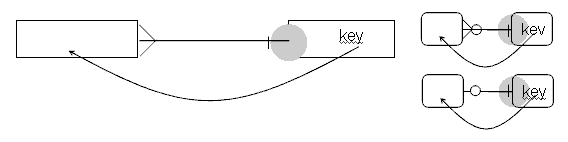
COACH (**coach\_id**, *holiday\_code*,…)



**3.5 Summary**

An ERD can be converted into a logical design suitable for a relational database by defining a set of **relations.** Each entity becomes a relation and each relationship implemented as a foreign key or a new relation depending on the type of relationship and whether there is optionality.

**1:M Relationships** require posting the primary key of the one end as a foreign key at the many end:



**Many to many relationships** require an extra relation which has two foreign keys, one from each side.

**Optionality at a one end of a relationship** requires a new relation unless it is a one to one relationship.

Note when producing your Logical Design, you will need to work from the ERD checking each relationship in turn. Depending on the model relationships you may end up with some relations having a few foreign keys.

**Session 3 Tutorial Exercises**

1. Produce the logical design for the following ER- Diagram:



It should be noted that this diagram assumes that the system will only deal with modules that are running and therefore must have students studying them (i.e. modules in development would be ignored).

2. Produce the logical design for the following ER- Diagram:



It should be noted that this diagram has been drawn for a system that only needs to store information on current loans (i.e. who has a book out on loan now).

3. Produce a logical design for the ERD you produced in Session 2 for the Beckett Park Playhouse system.

**Additional tasks for Challenging Activities**

4. Sub type task

Identify some sub types for the university library model above and redraw the model.

5. Exclusive arc task

A student studies either an undergraduate course or a postgraduate course.

For an example of a more realistic case study to model, please refer to the Door Step Sandwiches in the Session 1 & 2 folder.

**Check the boxes when you have completed the following Session 3 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 3 Reflection

|  |
| --- |
|  |

* Logical Design of Door Step Sandwiches

|  |
| --- |
|  |

* Do Session 3 quiz – Logical Design

Session 4 Normalisation

On completion of this session and its associated activities you should be able to:

* Understand the differences between normalised and un-normalised data
* Convert un-normalised data to first normal form
* Convert data in first normal form to second normal form
* Convert data in second normal form to third normal form

**4.1 Introduction**

There are two ways of approaching logical database design:

* **“top-down” approach** – starting with entity-relationship modelling before converting to a set of relations
* **“bottom-up” approach** - looking at the data in forms and reports which already exist in a current system or which are required in a new system, and building relations based on them

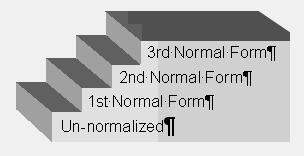
In the previous session we followed the top-down approach.

In this session we will introduce an appropriate ‘bottom-up’ approach. In this you will become familiar with the general idea of “normal forms” of data and the process of **normalisation.**

**4.2 Normalisation - an overview**

**What is it?**

At the logical design stage relations can be in one of four states or categories, depending on whether certain conditions are satisfied. These states are:



Each state imposes further conditions on those required by the previous one.

**Normalisation** is the process of transforming a relation or a set of relations which are the starting point of a logical design process into one of these normal forms; typically, the aim is to bring the logical design into the third normal form which is generally referred to as “normalised”.

**Why Normalisation?**

**We use normalisation to produce a logical design which will lead to an efficient and effective database.**

**The aim is to:**

* **Avoid unnecessary wasting of storage space from data redundancy (the appearance of the same data (attributes) in more than one place)**
* **Avoid data anomalies from insertion, deletion or amendment of redundant data**
* **Produce easier to use table structures**

If redundancies were allowed to exist then any insertion, deletion or amendment of data would have to be carried out for all tables which included that data and if inconsistently done then anomalies would occur in the database.

It can be noted that if the ERD approach has been used correctly then the resulting logical design will already be normalised to a degree. In this case, the normalisation process is regarded as being a check on the definitions of the relations and a way of “fine-tuning” them.

If the database is developed from forms and reports, then the initial logical design is very unlikely to be already normalised and the normalisation process will inevitably cause changes in the definitions of the relations.

There are additional materials on Normalisation available on VLE including a template which should be used to assist with the normalisation stages.

When you have normalised all the individual data sets then you should check to see if any new relations / tables have been identified, if so they should be added to the original ERD to form a complete (composite) model of the data requirements.

**Normalisation up to 3rd Normal Form (3NF)**

Normalisation up to 3rd Normal Form involves 4 states with transformative steps between them.

Step 1 UNF – List all the attributes from the document or report.

Identify a key attribute and underline it.

Identify any repeating groups and note with \*

Step 2 1NF - Copy the none repeating group data and its key attribute as a relation

Break out the repeating group to form another relation

Add the primary key from the UNF to the new relation as part of the key and also

include at least one other attribute with in that relation to make a compound key

Step 3 2NF – Copy any relations already in 2NF

Identify part key dependencies and form them into a new relation, then make the part key attribute they were dependent on the key for the new relation finally make the part key attribute in the relation they came from a Foreign Key (FK).

Step 4 3NF - Copy any relations already in 3NF

Identify any non-key dependencies and form them into new relations, then make the non key attribute they were dependent on the key for the new relation finally make the non key attribute in the relation they came from a Foreign Key (FK).

**4.3 Un-normalised data and its conversion to First Normal Form**

The following is a representation of a typical order form.

TO **Global Supplies** ORDER NO: 10345

DATE: September 19th 2019

Love to live Road

Leeds

LS6 4SA

Caedmon Hall

Headingley Campus

Leeds Metropolitan University

LS3 3QS Tel: 0113 2832600

FROM Customer number C101

**Invoice Name/Address** **Delivery Name/Address**

Mrs. Flowers c/o J. Law

The Shrubbery Clink Lane

Seldom Southwark

Wilts London

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ITEM REF** | **QTY** | **DESCRIPTION** | | **UNIT PRICE** | **AMOUNT** |
| EPSS020093 | 3 | Stylus Colour 400 Black Ink Cartridge | | 13.80 | 41.40 |
| IOM10091 | 5 | Zip Disk PC Format 100Mb | | 10.47 | 52.35 |
| MS109PS2 | 1 | White Mouse | | 6.95 | 6.95 |
|  | | | **SUBTOTAL** | | 100.71 |
| **VAT @ 17.5%** | | 17.62 |
| **POSTAGE & PACKING** | | 5.00 |
| **TOTAL DUE** | | 123.33 |

From this form we can identify the following list of attributes that will need to exist in a database system designed to handle orders:

Order\_No

Date

Customer\_No

Invoice\_Name

Invoice\_Address

Delivery\_Name

Delivery\_Address

*Item\_Ref*

*Qty*

*Description*

*Unit\_Price*

*Amount*

Subtotal

VAT

Postage&Packing

Total

Note that the name, address and telephone number for Global Supplies are not included on the list because the system is to be used by them and therefore they do not need their own data to be stored in the database.

We are now in a position to consider whether or not this list of attributes is in an un-normalised form and if it is to see how it can be normalised.

***Recognising UNF***

In order to understand why this data is un-normalised it is useful to consider what would happen if an attempt was made to create just one database table with all of these attributes included.

An un-normalised table can be considered in various different but equivalent ways as one in which the attribute values are not atomic (single) resulting in embedded tables or repeating groups of data.

For each primary key value there should be one associated value for each attribute. The following shows that for several of the columns the data is not atomic – the cells contain more than one attribute value so has an embedded table or repeating group.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Order  \_No | Date | Etc. |  | | Delivery\_  Address | Item\_Ref | Qty | | | Description | | Unit  Price | | Amount | | P\_and\_P |
| 🞈 | 🞈 | 🞈 | | 🞈 | 🞈 | 🞈  🞈  **🞈** | | | 🞈  🞈  🞈 | 🞈  🞈  🞈 | | 🞈  🞈  🞈 | | 🞈  🞈  🞈 | | 🞈 |
|  |  |  | |  |  |  | |  | |  |  | |  | |  | |

**Activity 1**

In the Order table below identify the embedded table or repeating group?

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Order  \_No | Delivery\_  Address | Item\_Ref | Qty | Description | Unit\_  Price | Amount | P\_and\_P |
| 10345 | c/o J. Law  Clink Lane  Southwark  London | EPSS020093 | 3 | Stylus Colour 400 Black Ink Cartridge |  |  | 5.00 |
| IOM10091 | 5 | Zip Disk PC Format 100Mb |  |  |  |
| MS109PS2 | 1 | White Mouse |  |  |  |
| 10447 |  |  |  |  |  |  |  |

. .

***Conversion to 1NF***

* Copy the non-repeating group data and its key attribute as a relation
* Break out the repeating group to form another relation
* Add the primary key from the UNF to the new relation as part of the key and also include at least one other attribute with in that relation to make a compound key

The following tabular method is recommended for carrying out the conversion. To help you with the process you should use the **Normalisation Template** available in Session Materials, Session 4 on the VLE. The template also includes the steps to follow for each normal form.

Relation with non-repeating group data and its the key attribute

Relation for repeating group with a compound key from the UNF key and at least one other attribute

|  |  |
| --- | --- |
| **Un-normalised form** | **First normal form** |
| **Order\_No** | **Order\_No** |
| Date | Date |
| Customer\_No | Customer\_No |
| Invoice\_Name | Invoice\_Name |
| Invoice\_Address | Invoice\_Address |
| Delivery\_Name | Delivery\_Name |
| Delivery\_Address | Delivery\_Address |
| Item\_Ref\* | Subtotal |
| Qty\* | VAT |
| Description\* | Postage&Packing |
| Unit\_Price\* | Total |
| Amount\* |  |
| Subtotal | ***Order\_No*** |
| VAT | **Item\_Ref** |
| Postage&Packing | Qty |
| Total | Description |
|  | Unit\_Price |
|  | Amount |

To do this we started with the original unnormalised list. We identified the repeating group items shown with \*. From the non-repeating group items, we chose Order\_No as the primary key. We placed all of the non-repeating group items together in one relation. We repeated the primary key at the head of the list of repeating group items in another relation. Finally, we identified all of the keys appropriately.

Note that we have separated one list of attributes into two separate lists therefore one foreign key has had to be identified. The resulting attribute lists are relations in every sense except that we have yet to identify names for them.

To see the effect of conversion to 1NF we can look at how the database tables would be created based on these relations. We would create an ORDER table (showing two orders):

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Order  \_No | Customer  \_No | Etc….. | Delivery  \_Name | Delivery  \_Address | P\_and\_P | Etc….. |
| 10345 | C101 |  | J. Law | Clink Lane  Southwark  London | 5.00 |  |
| 10447 | Etc |  |  |  |  |  |

and an ORDER-ITEM table:

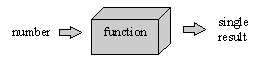
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Order  \_No | Item\_Ref | Qty | Description | Unit\_  Price | Amount |
| 10345 | EPSS020093 | 3 | Stylus Colour 400 Black Ink Cartridge | 13.80 | 41.40 |
| 10345 | IOM10091 | 5 | Zip Disk PC Format 100Mb | 10.47 | 52.35 |
| 10345 | MS109PS2 | 1 | White Mouse | 6.95 | 6.95 |
| 10447 | Etc… |  |  |  |  |

We can see that for order number 10345 there is one row in the ORDER table and three in the ORDER-ITEM table. There are no repeating groups and no empty cells.

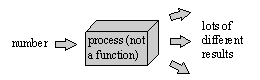
***Formal definition of 1NF***

The formal definition uses the phrase “functionally dependent” so we must first define what that means.

Mathematicians talk about functions, like the square (or sine or log) of something, when they mean that, given a number, you can feed it into the function and a single unique result will come out.



If several results could arise, then the process cannot be called a function.



**Functional dependence (or dependency)**

An attribute B is functionally dependent on an attribute A if for each value of A there is only one value of B associated with it. So, a relation is in First Normal Form if all attributes are functionally dependent on the primary key. In other words, for each value of the primary key there is only one value for each attribute in the relation.

**Activity 2**

A manufacturer of soft toys obtains the materials and other items which will go to make up the toys, such as fur fabric, eyes and squeak mechanisms, from a variety of different suppliers. She currently stores the information about the suppliers of materials and what they can supply in a table, as shown below.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Material No | Description | Unit Of Ordering | Supplier No | Supplier Name | Supplier Address | Minimum Order Quantity | Delivery Time | Price |
| M501 | Fur fabric | Metre | S455 | J. Smith plc | 12 May St  Bootle | 1000 | 3 weeks | 4.45 |
|  |  |  | S126 | Fabulike | 2 Green St  Leeds | 1500 | 2 weeks | 4.50 |
|  |  |  | S312 | F. Binns Ltd | Oddy Way  Grimsby | 500 | 2 weeks | 4.67 |
| P377 | Eyes 50mm | Each | S007 | Partsco | Axo Works  Back Lane  Redcar | 3000 | 1 week | 0.01 |
|  |  |  | S402 | Glasso plc | 31 Cog Rd  Leeds | 2000 | 1 week | 0.01 |
| P402 | Growl mechanism | Each | S007 | Partsco | Axo Works  Back Lane  Redcar | 1000 | 2 weeks | 0.37 |

1. Mark the column range for the embedded table(s) in the MATERIAL table above.
2. List the attributes in tabular form (UNF) using the Normalisation Template.
3. Convert the data from UNF to 1NF on the Normalisation Template.
4. Draw the conceptual data model for the two resulting relations.

**4.4 Second Normal Form**

***Conversion from 1NF to 2NF***

We are only concerned with relations that have a compound key, effectively, any relation with a simple (single attribute) key is already in 2NF if it is in 1NF.

The process involves checking whether there are attributes (or not) that depend only on **one part** of the compound key. You carry out the following:

* Copy any relations already in 2NF
* Identify part key dependencies and form them into a new relation, then make the part key attribute they were dependent on the key for the new relation finally make the part key attribute in the relation they came from a Foreign Key (FK).

NB Don’t forget to ensure that the original relation retains the compound key.

Order Item has a compound key so may have partial key dependancies. Are any attributes dependant on part of the key? YES Description and Unit\_Price are dependant on only Item\_Ref, so it is not in 2NF.

Order MUST be in 2NF as it has a simple key, there can be no partial key dependencies. It is in 2NF.

|  |  |
| --- | --- |
| **Un-normalised form** | **First normal form** |
| **Order\_No** | **Order\_No** |
| Date | Date |
| Customer\_No | Customer\_No |
| Invoice\_Name | Invoice\_Name |
| Invoice\_Address | Invoice\_Address |
| Delivery\_Name | Delivery\_Name |
| Delivery\_Address | Delivery\_Address |
| Item\_Ref\* | Subtotal |
| Qty\* | VAT |
| Description\* | Postage&Packing |
| Unit\_Price\* | Total |
| Amount\* |  |
| Subtotal | ***Order\_No*** |
| VAT | **Item\_Ref** |
| Postage&Packing | Qty |
| Total | Description |
|  | Unit\_Price |
|  | Amount |

Once again, to undertake the conversion we use the tabular approach. In this example, the result is as follows:

|  |  |  |
| --- | --- | --- |
| **Un-normalised form** | **First normal form** | **Second normal form** |
| Order\_No | Order\_No | Order\_No |
| Date | Date | Date |
| Customer\_No | Customer\_No | Customer\_No |
| Invoice\_Name | Invoice\_Name | Invoice\_Name |
| Invoice\_Address | Invoice\_Address | Invoice\_Address |
| Delivery\_Name | Delivery\_Name | Delivery\_Name |
| Delivery\_Address | Delivery\_Address | Delivery\_Address |
| (Item\_Ref | Subtotal | Subtotal |
| Qty | VAT | VAT |
| Description | Postage&Packing | Postage&Packing |
| Unit\_Price | Total | Total |
| Amount)\* |  |  |
| Subtotal |  |  |
| VAT |  | *Order\_No* |
| Postage&Packing |  | *Item\_Ref* |
| Total | *Order\_No* | Qty |
|  | Item\_Ref | Amount |
|  | Qty |  |
|  | Description | Item\_Ref |
|  | Unit\_Price | Description |
|  | Amount | Unit\_Price |

Simple key so no change from 1NF

Description and Unit\_Price are now in a separate relation with the appropriate key

Quality and Amount dependant on the full compound key. Item\_ref now a FK here

Note that we now have two foreign keys defined (*Order\_No* and *Item\_Ref*).

If the 2NF relations were implemented as tables the ORDER table would be as before,

the ORDER-ITEM table would have less columns than before:

|  |  |  |  |
| --- | --- | --- | --- |
| Order\_No | Item\_Ref | Qty | Amount |
| 10345 | EPSS020093 | 3 | 41.40 |
| 10345 | IOM10091 | 5 | 52.35 |
| 10345 | MS109PS2 | 1 | 6.95 |
| 10447 | Etc… |  |  |

There would be a new table which would sensibly be named PRODUCT:

|  |  |  |
| --- | --- | --- |
| Item\_Ref | Description | Unit\_Price |
| EPSS020093 | Stylus Colour 400 Black Ink Cartridge | 13.80 |
| IOM10091 | Zip Disk PC Format 100Mb | 10.47 |
| MS109PS2 | White Mouse | 6.95 |
| Etc… |  |  |

The advantage of having data in 2NF is that we have removed redundancy. In 1NF we would, for instance, have to provide the value for the description attribute in the ORDER-ITEM table every time a new order came in which had that particular product on it. In 2NF the value is provided once only in the PRODUCT table.

***Formal definition of 2NF***

We can extend the definition of functional dependency (which means single-valued) to the idea of full functional dependency (which means dependent on the whole not part).

**Full Functional dependence (or dependency)**

If attribute B is functionally dependent on attribute A and is not functionally dependent on any subset of attribute A, then attribute B has full functional dependency on attribute A. (Note that the term “attribute A” here must be on the understanding that A consists of a set of attributes taken together to form a compound attribute (if A were a single attribute it can have no subset!)

**Second Normal Form**

A relation is in Second Normal Form if

* it is already in 1NF and
* all non-key attributes are fully functionally dependent on the key.

**Activity 3**

Using the Normalisation Template from Activity 2 convert the relations into 2NF.

**Activity 4**

a) Suggest a reason why the following relations are unlikely to be in 1NF. Show how each relation could be converted into 1NF.

CHILDBENEFIT ( CB\_Reference#, Payee, ChildID, ChildName, ChildDateofBirth, ...)

Note that Child benefit is paid to families as a total sum for all their children under 16.

CUSTOMER (Account#, Address, Newspaper Name, Newspaper Type, . . . )

Note that the customers of a newsagent have newspaper(s) delivered to them; a newspaper is either of type daily or of type Sunday.

b) What is needed for a relation to be in 2NF? Why is 2NF a desirable state for relations?

c) Suggest a reason why the following relation (which records the regular teaching done at a music college) is unlikely to be in 2NF. Show how the relation can be converted to 2NF.

MUSICLESSON (Teacher#, Pupil#, DayofWeek, Time, Duration, PupilName, PupilAge, Instrument, TeacherName, Fee, ….).

Note that no pupil has two sets of lessons with the same teacher.

**4.6 Informal definition of 3NF**

The Third Normal Form is very similar to the Second Normal Form. However, instead of considering whether some attributes in our table are dependent only on part of a compound key, we look to see whether they are dependent on attributes which are **not** part of the key. In a very similar way to 2NF, we remove the attributes which depend on this (non-key) attribute into a different table.

As an example of a table being in 2NF but not in 3NF consider the following: -

A popular band of musicians plays at many different clubs and pubs, at festivals and at private functions. Data about each gig is stored in the following table

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| RefNo | Date | Time | Fee | BookerTel | VenueName | VenueAddress | VenueTel |
| 00/56 | 13.03.06 | 20.00 | 400.00 | 0114 9998888 | Sheffield Arena | Sheffield S3 4KK | 0114 7777777 |
| 00/61 | 15.03.06 | 19.30 | 450.00 | 0113 9988899 | Majestyk | City Square Leeds  LS1 1PP | 0113 8888888 |
| 00/73 | 25.03.06 | 21.00 | 480.00 | 0113  7899789 | Evolution | Warner Village Leeds  LS4 2GG | 0113 6666666 |
| 00/57 | 20.03.06 | 21.00 | 500.00 | 0114 7897899 | Sheffield Arena | Sheffield S3 4KK | 0114 7777777 |

The design of this table can be represented by the relation:

CONCERT ( RefNo, Date, Time, Fee, BookerTel, VenueName, VenueAddress, VenueTel)

We can note that both the relation and the table:

is already in First Normal Form, because there are no repeating groups

is already in Second Normal Form, because the primary key is not compound

has some attributes that depend on an attribute other than the primary key:

VenueAddress and VenueTel depend only on VenueName (assuming this is unique) and not on the primary key.

The problem is that the telephone number of a particular venue can appear several times in different rows of the table. If it changes it will need to be altered several times over, rather than just once.

We need therefore to create a new table for the venue details and make VenueName a foreign key in the original table.

**4.7 Conversion to 3NF**

This is a very similar process to the one for 2NF, but now we operate on an attribute is not part of a key.

In the above example Venue Name would be this attribute, which is not a key attribute but one on which other attributes depend. We move the attributes that depend on Venue Name into a new table. To do this we carry out the following:

* Copy any relations already in 3NF
* Identify any **non-key dependencies** and form them into new relations, then make the non key attribute they were dependent on the key for the new relation finally make the non key attribute in the relation they came from a Foreign Key (FK).

In the above example, this would create the following two relations:

VENUE ( VenueName, VenueAddress, VenueTelNo)

CONCERT ( RefNo, Date, Time, Fee, BookerTelNo, *VenueName* )

The corresponding tables would be:

VENUE

|  |  |  |
| --- | --- | --- |
| VenueName | VenueAddress | VenueTelNo |
| Sheffield Arena | Sheffield S3 4KK | 0114 7777777 |
| Majestyk | City Square Leeds  LS1 1PP | 0113 8888888 |
| Evolution | Warner Village Leeds  LS4 2GG | 0113 6666666 |

CONCERT

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Ref\_No | | Date | Time | Fee | Booker\_Tel\_No | Venue\_Name |
| 00/56 | 13.03.06 | 20.00 | 400.00 | 0114 9998888 | Sheffield Arena |
| 00/61 | 15.03.06 | 19.30 | 450.00 | 0113 9988899 | Majestyk |
| 00/73 | 25.03.06 | 21.00 | 480.00 | 0113 7899789 | Evolution |
| 00/57 | 20.03.06 | 21.00 | 500.00 | 0114 7897899 | Sheffield Arena |

Note that the conversion to 3NF has removed redundancy from the data stored in the tables.

**Activity 5**

Let’s return to the order example.

The normalisation has so far produced the following 2NF:

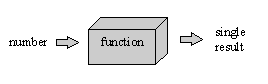
|  |  |  |  |
| --- | --- | --- | --- |
| **Un-Normalised Form UNF** | **First**  **Normal Form 1NF** | **Second**  **Normal Form 2NF** | **Third**  **Normal Form 3NF** |
| Order\_No | Order\_No | Order\_No |  |
| Date | Date | Date |  |
| Customer\_No | Customer\_No | Customer\_No |  |
| Invoice\_Name | Invoice\_Name | Invoice\_Name |  |
| Invoice\_Address | Invoice\_Address | Invoice\_Address |  |
| Delivery\_Name | Delivery\_Name | Delivery\_Name |  |
| Delivery\_Address | Delivery\_Address | Delivery\_Address |  |
| (Item\_Ref | Postage&Packing | Postage&Packing |  |
| Qty | Subtotal | Subtotal |  |
| Description | VAT | VAT |  |
| Unit\_Price | Postage&Packing | Postage&Packing |  |
| Amount)\* | Total | Total |  |
| Subtotal |  |  |  |
| VAT | *Order\_No* | *Order\_No* |  |
| Postage&Packing | Item\_Ref | *Item\_Ref* |  |
| Total | Qty | Qty |  |
|  | Description | Amount |  |
|  | Unit\_Price |  |  |
|  | Amount | Item\_Ref |  |
|  |  | Description |  |
|  |  | Unit\_Price |  |

a) Complete this to **3NF**.

b) Now draw an entity model corresponding to the 3NF relations.

**4.8 Formal definitions**

Going back to the idea of functional dependency, we call the number in the diagram below a determinant of the result (it determines the result).



**Determinant**

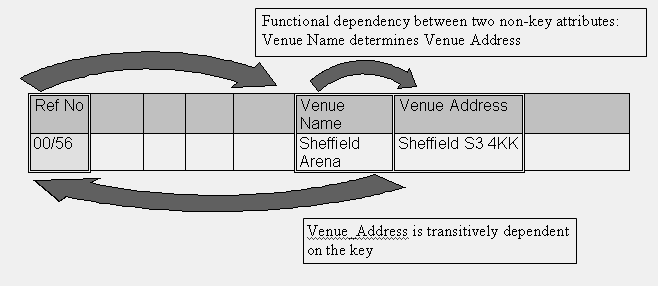
If attribute B is functionally dependent on attribute A, then attribute A is a determinant of attribute B.

Is StudentId a determinant of Student Name?

Is Student Name a determinant of StudentId?

**Transitive Dependency**

If attribute A is a determinant of attribute B and B is a determinant of attribute C then attribute C has a transitive dependency on attribute A. The following diagram illustrates this for an example where Ref No is attribute A, Venue Name is attribute B and Venue Address is attribute C.



**Third Normal Form**

A relation is in Third Normal Form if -

* it is already in 2NF
* there are no functional dependencies between any pair of non-key attributes (i.e. there are no transitive dependencies)

**4.9 Summary**

The relations of our logical design can be further improved in terms of elimination of redundancy by putting them into Third Normal Form. Here is a summary of the normal forms:

* First Normal Form - achieved by removing repeating group attributes into a new relation and adding a link to it – tables implemented at this stage would have single entries for all attribute values
* Second Normal Form – achieved by removing non-key attributes not dependent on the whole key into a new relation and adding a link to it – tables implemented at this stage would have some redundancy removed
* Third Normal Form – achieved by removing non-key attributes with functional dependency – tables implemented at this stage have all redundancies removed
* When tables are in 3NF we tend to say they are normalised, though there are other forms

**To help you remember the 3 normal forms:**

The attributes in a table should depend on

**1NF the key**

**2NF the whole key**

**3NF nothing but the key**

**Activity 6**

A Recording Company wishes to set up a database to record data about the artists (musicians) whose recordings it produces. For each artist it needs to hold their surname, one forename, their address, their telephone number and the name and address of their agent: each artist and each agent is allotted a unique identifying number. It cannot be assumed that any given agent will represent only one artist, though it can be assumed that any given artist will only have one agent. The Company also wishes to hold details of the various recordings that each artist has made. Each recording can be identified by a unique reference number. A first draft un-normalised relation (i.e. with no repeating group identified is:

ARTIST (Artist#, Surname, Forename, Artist Address, Artist Phone#, Recording#, Recording Title, Recording Type, Agent#, Agent Name, Agent Address)

Convert this to 3NF using the tabular method. You may wish to carry this out electronically using Excel so that you can by cut and past the attributes between the normal forms. Alternatively use the Normalisation template available on the VLE as this has the rules at the top of each column to aid you. A starting point is given in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Un-normalised form** | **First**  **normal form** | **Second**  **normal form** | **Third**  **normal form** |
| Artist # |  |  |  |
| Surname |  |  |  |
| Forename |  |  |  |
| Artist\_address |  |  |  |
| Artist\_phone# |  |  |  |
| Recording# |  |  |  |
| Recording\_title |  |  |  |
| Recording\_type |  |  |  |
| Agent# |  |  |  |
| Agent\_name |  |  |  |
| Agent\_address |  |  |  |

**Activity 6**

A travel agency records information whenever a customer books a holiday, including the customer's name and address, the date of booking the holiday, the date of travel, the duration of the holiday (for example 7 nights), the (unique) flight reference number, etc. A booking identification number is allocated to each new booking, and a customer number to each customer.

The booking also records the reference number, name and grade of the customer's hotel, as well as the name of the airline on which the customer will be flying, and the airport of departure.

A first draft relation (no repeating groups identified) defining this data with Booking# as its key, would be:

BOOKING (**Booking#**, Customer#, Customer Name, Customer Address, Booking Date, Travel Date, Duration, Flight#, Hotel#, Hotel Name, Hotel Grade, Airline, Departure Airport )

a) Please answer the following questions:

|  |  |  |
| --- | --- | --- |
|  |  | ***Why?*** |
| Are there any repeating groups? | Yes  No |  |
| Is the relation (table) already in 1NF? | Yes  No |  |
| Does the table have a compound key? | Yes  No |  |
| Is the relation already in 2NF? | Yes  No |  |
| Are there dependencies between  non-key attributes? | Yes  No |  |
| Is the relation already in 3NF? | Yes  No |  |

b) Why do we need to create more relations?

c) Please indicate the four relations needed with all their attributes

1. BOOKING ( **Booking#**

2

3

4

When you have obtained a set of normalised relations you are ready to implement them as tables using a Relational Database Management System e.g. ORACLE.

**Session 4 Tutorial Exercises**

1. The following relation records the details of a house, including the offers made for it by potential buyers, in an estate agent’s system. A buyer may put in offers for two or more houses at the same time.

PROPERTY (Property Code, Property Type, Number Of Bedrooms, Address, Vendor ID, [Buyer ID, Buyer Name, Buyer Address, Offer Date, Amount Offered]\* , Asking Price)

Convert it to 3NF using the Normalisation Template

2. For the following three relations please decide which of the following keys shown applies.

a) Simple primary key

b) Compound primary key

c) Foreign key

RELATION 1 ( A, B, C, D, )

RELATION 2 ( P, Q , R, S, )

RELATION 3 ( *T, U*, V, W, )

3. For each of the following, please decide if the statement given is applicable for the

**First** Normal Form, **Second** Normal Form or **Third** Normal Form.

|  |  |  |
| --- | --- | --- |
| 1) | The non-key attributes depend on the whole (compound) key, not just part of it | Normal Form |
| 2) | For each row of the table, there is a single entry in each column | Normal Form |
| 3) | In a table in this form all non-key attributes are fully functionally dependent on the key | Normal Form |

1. Suggest a reason why the following relation is unlikely to be in 3NF. Show how the relation can be altered to 3NF.

LECTURER (Lecturer Name, Address, Age, Department, Faculty, …)

5. Examine the following relations. For each relation state whether it is in UNF, 1NF, 2NF or 3NF.

a. PUPIL (Pupil Ref#, Pupil\_Name, FACULTY\_Ref#, FACULTY\_Name)

b. PROJECT (Project#, Project\_Name, Project\_Leader#)

c. HORSE RACE (Race\_Id, Horse\_Id, Position\_Achieved, Horse\_Name, Race\_Length, Jockey, Start\_Time)

**Check the boxes when you have completed the following Session 4 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 4 Reflection

|  |
| --- |
|  |

* Do Session 4 quiz – Normalisation

|  |
| --- |
|  |

* Start the General Data Protection Regulation (GDPR) Materials

Session 5 Introduction to the Oracle APEX environment

**Aims**

* log into and out of ORACLE
* reset your password
* open, save and edit an SQL Script file
* using DESCRIBE
* create tables with simple and compound PRIMARY KEYS
* create tables with foreign keys
* Insert Data

**Introduction – READ THIS**

ORACLE Application Express (APEX) provides you with a set of tools to work with the database and develop web enabled applications. At the time this document was prepared: it is an **Oracle 19c database running Apex Application Express 19.1.0.00.15.**

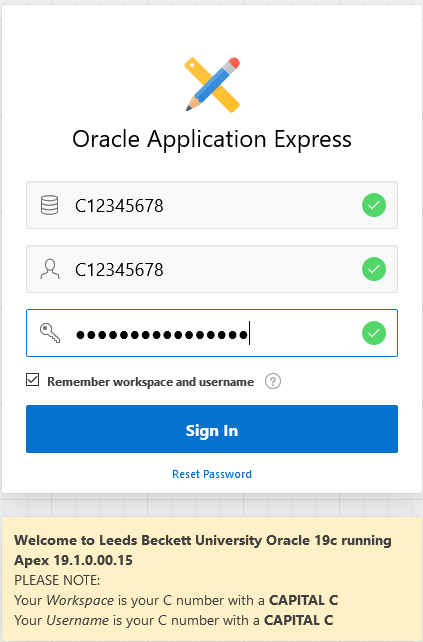
This workbook will introduce you to the main APEX Tools and SQL Commands, be very careful that you don’t miss out any steps, as following tasks assume all previous tasks have been completed.

A full SQL reference is available by clicking on the Help button on the APEX Home page.

**Tasks - DO THIS**

1. Log in to the VLE (Fundamentals of Databases module) and look in the Session 5 activities folder for the file **practical1.sql**, copy it into one of your directories. Make sure it keeps the **.sql** extension or ORACLE will not recognise it.
2. To Log on to ORACLE APEX click on the **APEX icon** on the desktop. Then enter your student id (**with a capital C**) from your student card for both the **Workspace**, **Username and password** initially as your password will have been set to your original password so you need to reset it. Now log in.

**WARNING - if you enter an incorrect password more than 5 times you will be locked out from APEX. Reset Password before this happens. If you have any problems logging in to APEX please contact the Helpdesk in JG212 directly. Bring along your student card!**



Type in your student id number with an uppercase C for all of Workspace, Username AND password as this will trigger Reset password

Password

1. **You can also log in to APEX remotely by pasting the following URL link** [**https://aet-oracle3.aet.leedsbeckett.ac.uk:8443/ords/**](https://aet-oracle3.aet.leedsbeckett.ac.uk:8443/ords/) **into your web browser.**

**Script files – READ THIS**

You can enter SQL commands directly using the APEX SQL Command Tool. However, when developing applications, it is usual practice to create **script files** to store any SQL commands with comments, which may need to be run more than once e.g. CREATE TABLE commands or a regularly used SQL query. A script file allows you to quickly retrieve, edit and rerun SQL commands. A script file is a simple text file and can be created outside of APEX using any text editor such as notepad. Script files should be saved with a suffix of **.sql**

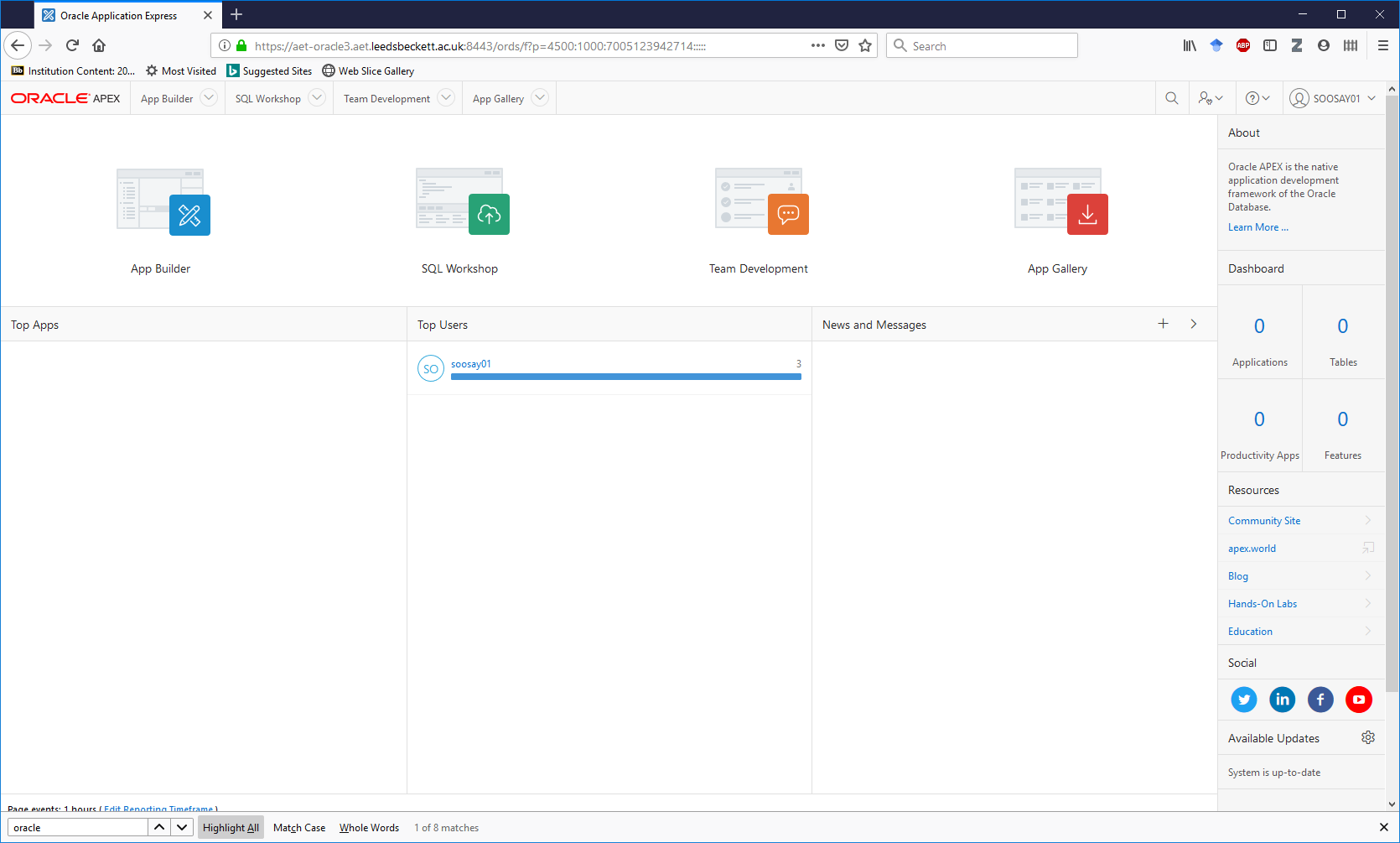
**IMPORTANT do not confuse a script file with a database table, even though they could both have the same name.**

The following screens show you how to run a script file called **Practical1.sql**. This file contains SQL commands that will create some tables in your database workspace.

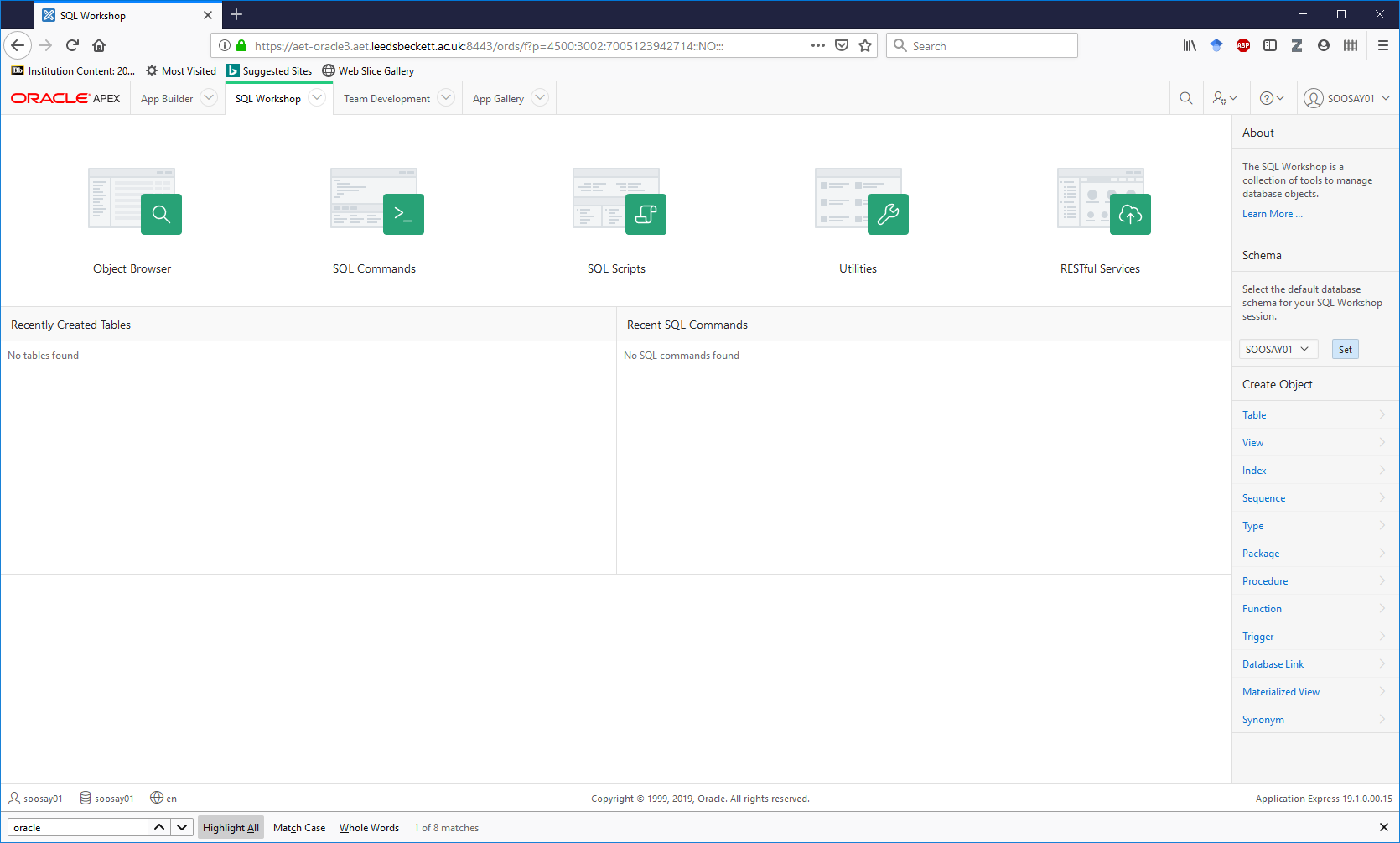
**Tasks – DO THIS**

1. Once logged in to APEX, on the Home page click on the **SQL Workshop** icon.

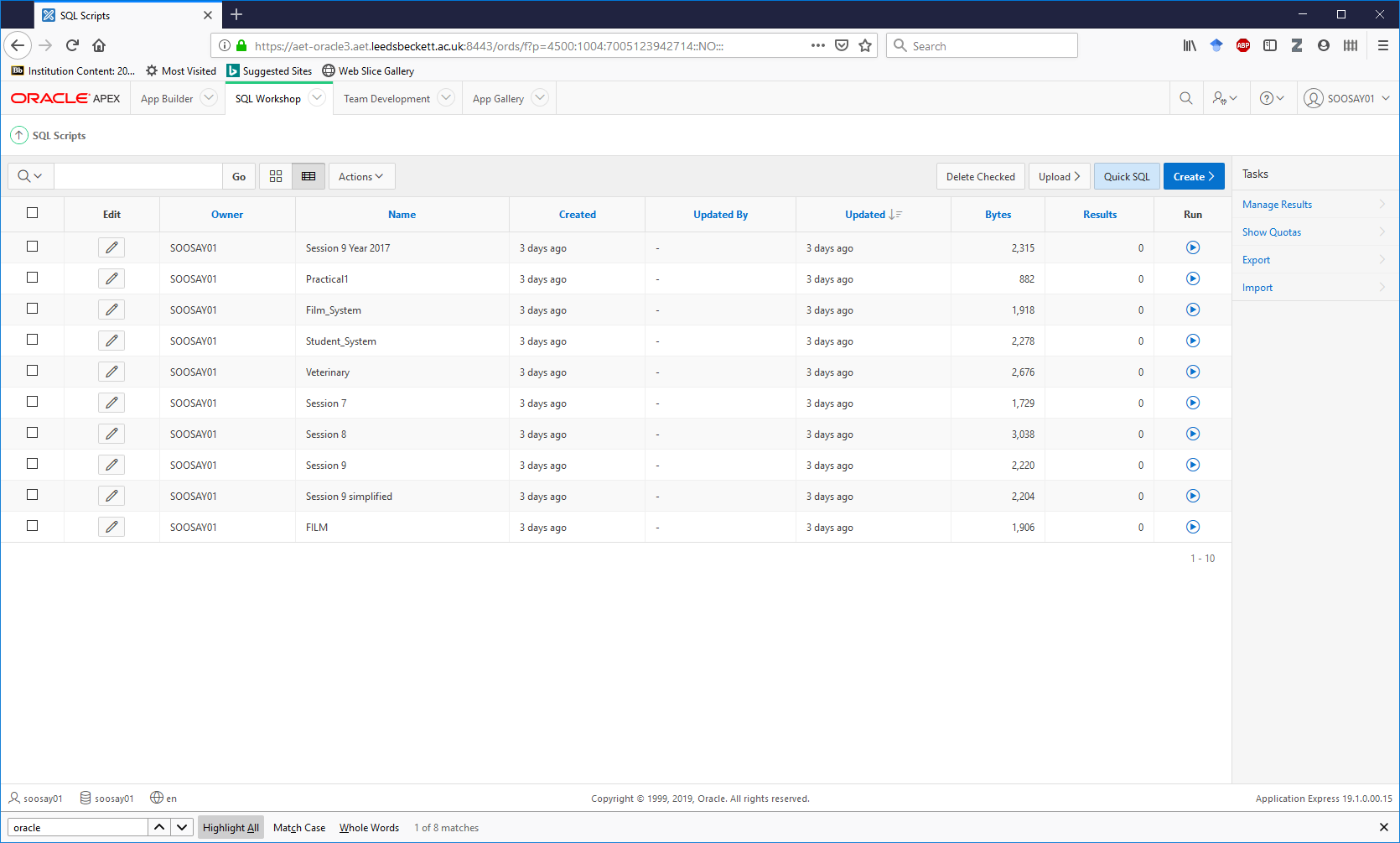
The ORACLE Apex Home Page



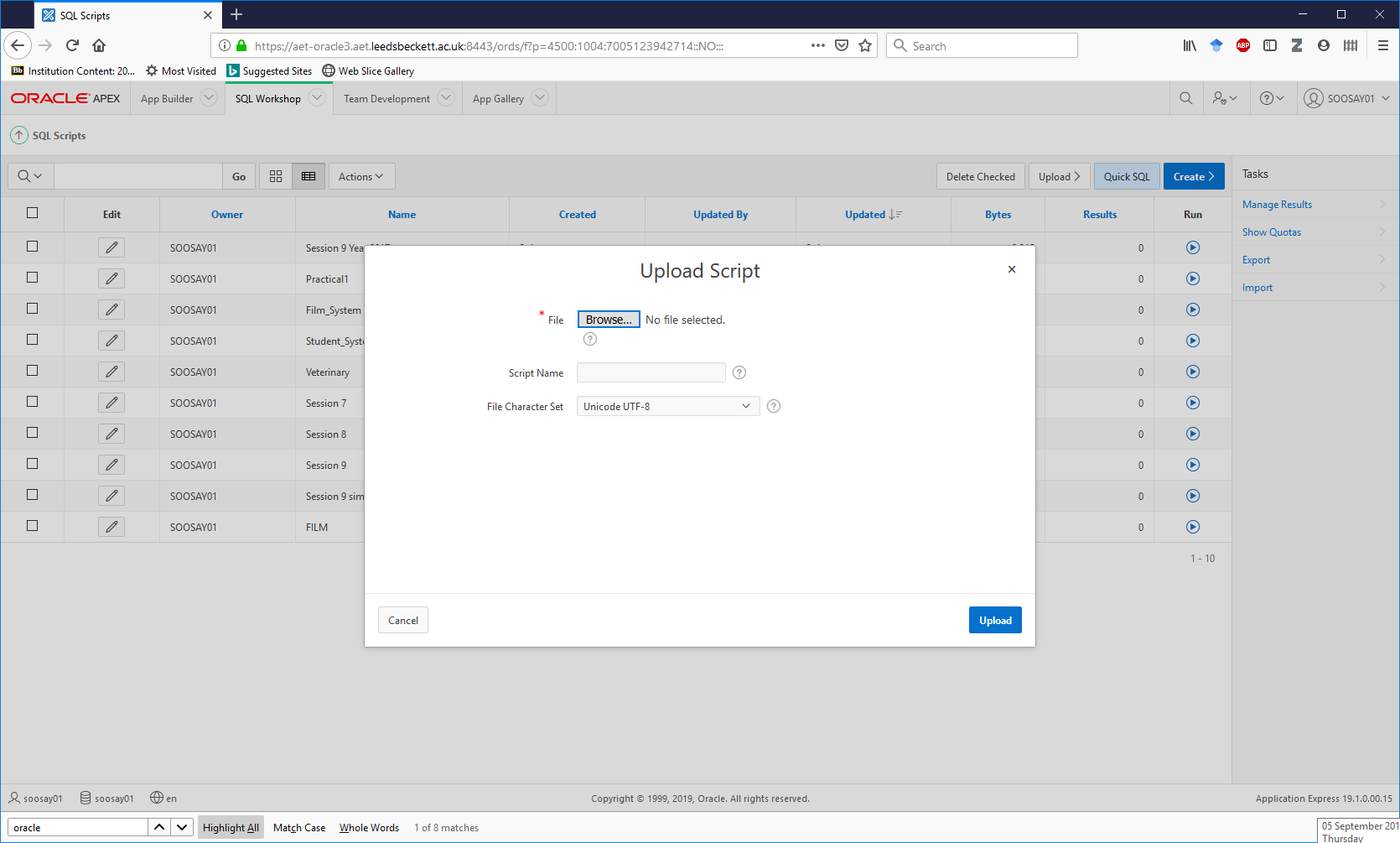
Then click on **SQL Scripts**.



1. Then click **Upload>**.



1. Browse for the file you have just saved **Practical1.sql** and enter a name for the uploaded script file e.g. **Practical1**

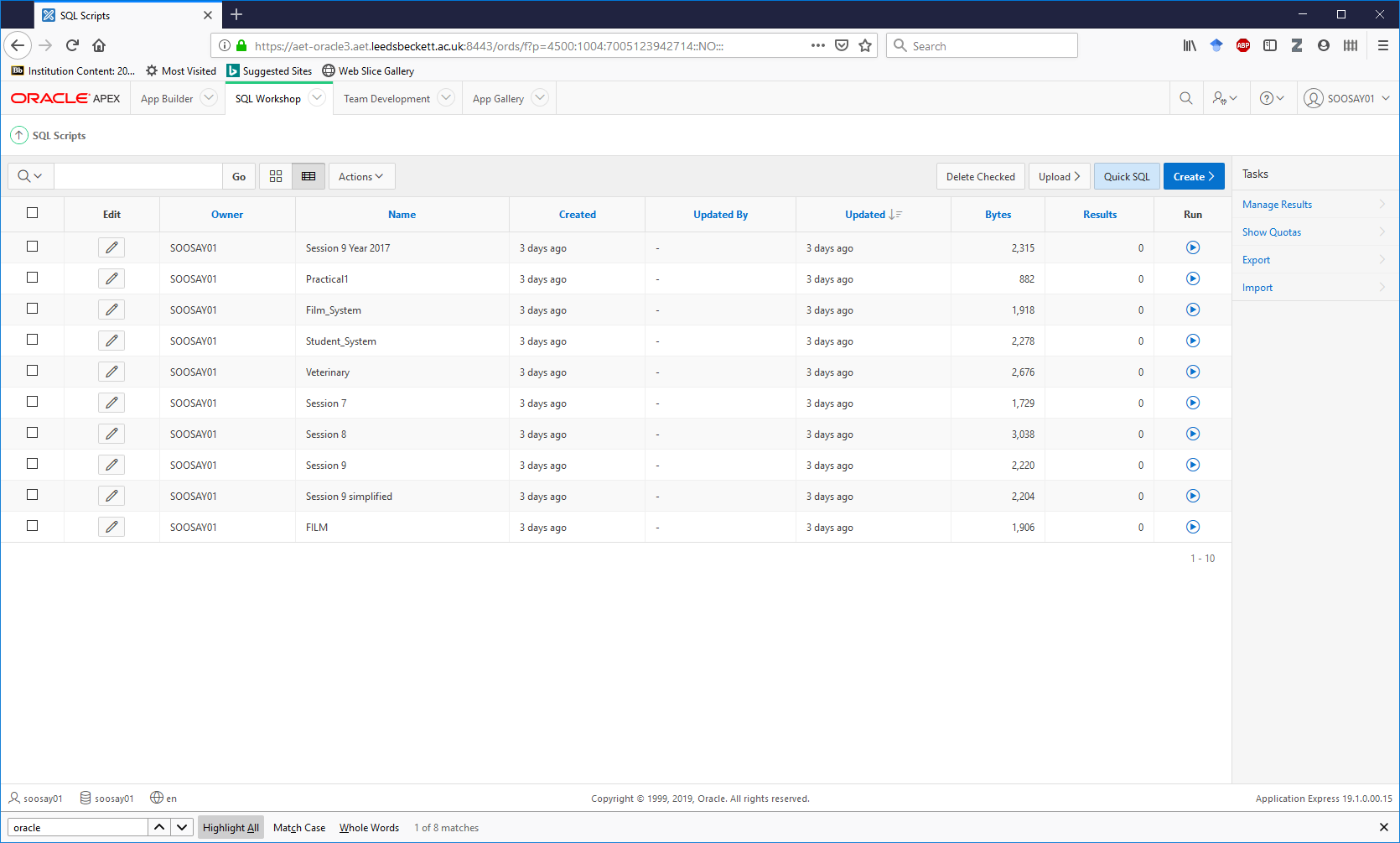


3. Finally, click on **Upload**.

2. Enter a Script Name **Practical1**

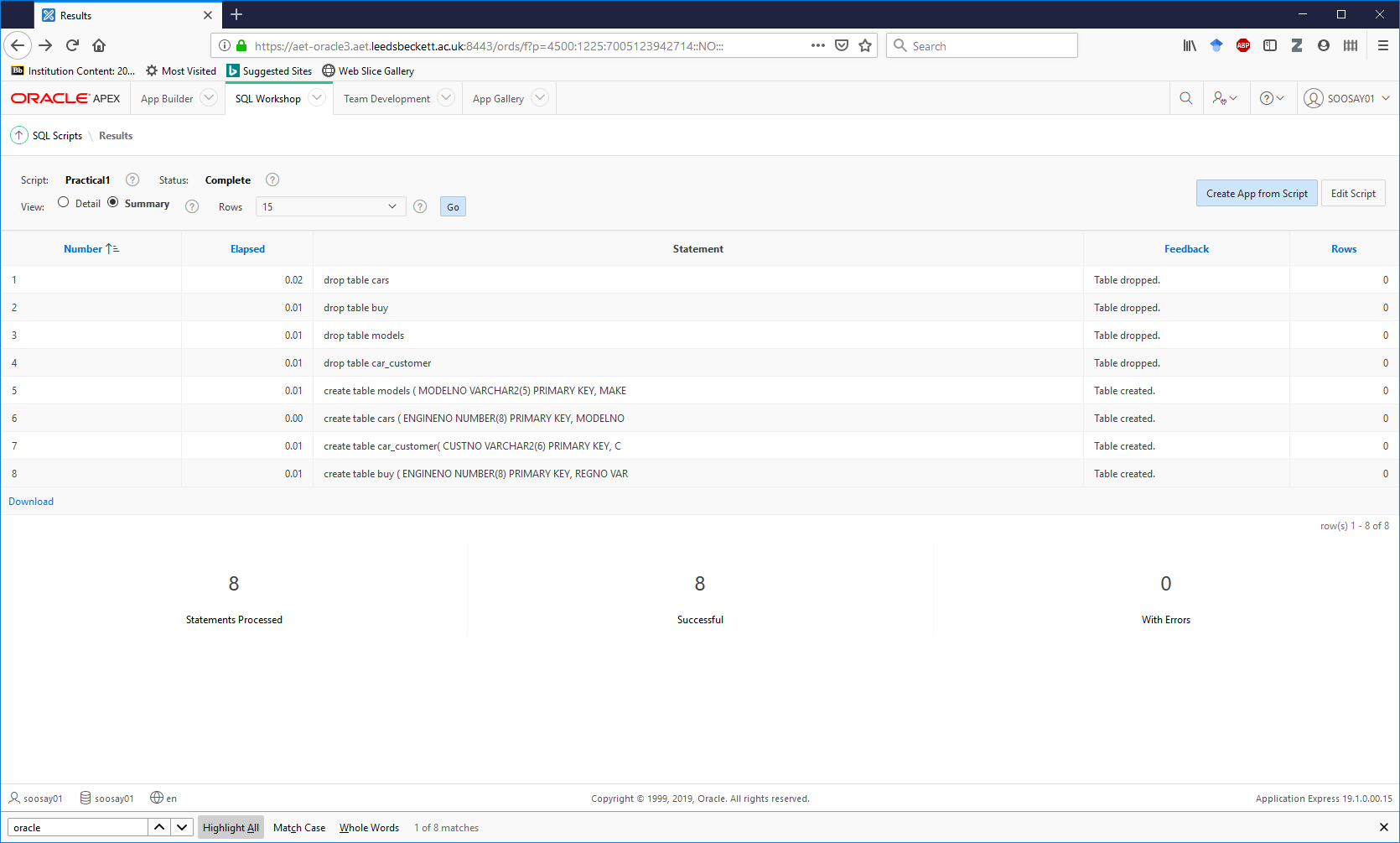
1. Browse for the **Practical1.sql** file

1. **Practical1** should appear in the SQL scripts list. Then, click **Run **.

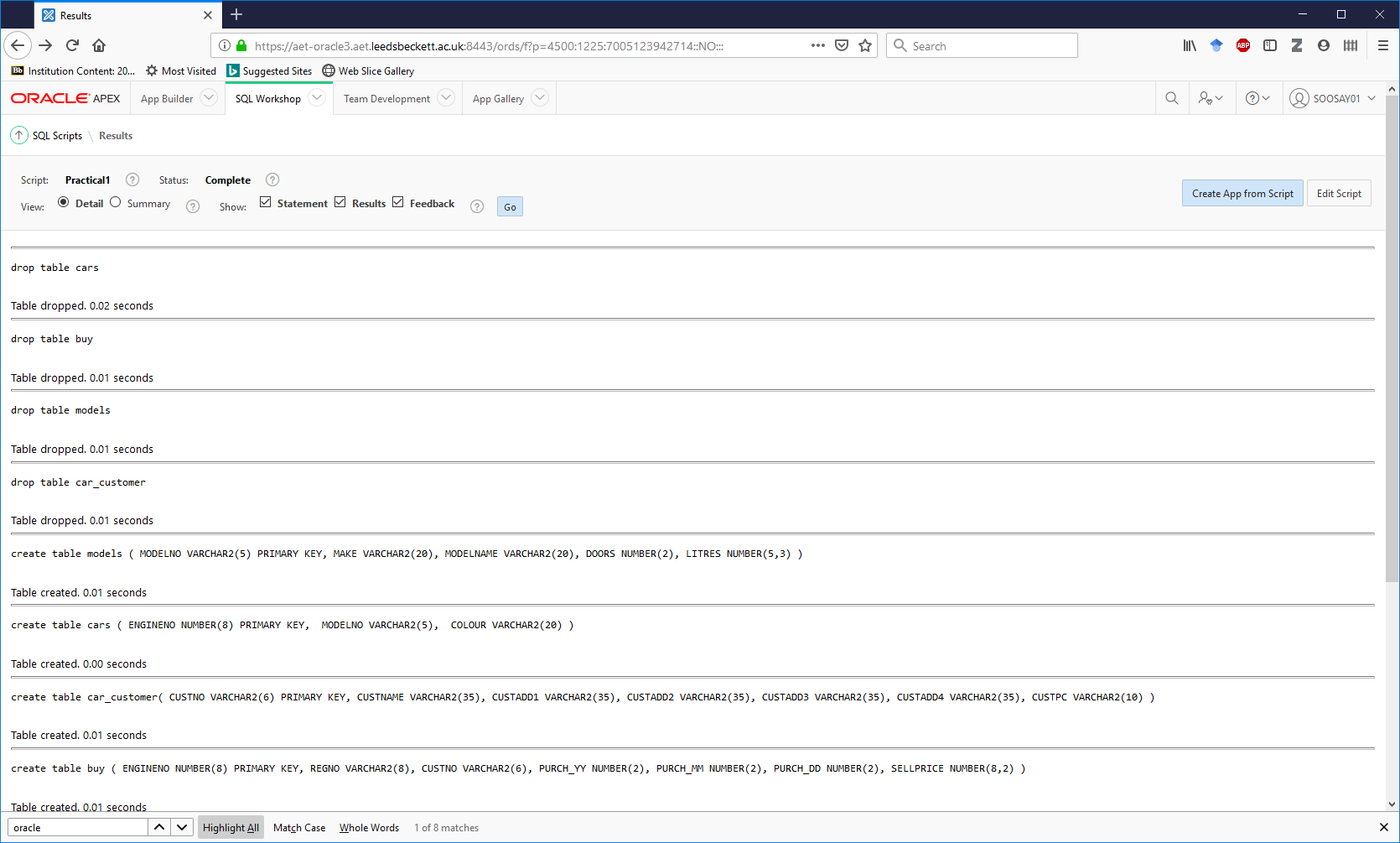


Click on the **Run** for **Practical1** in the list.

1. Then on the Run Script page click on **Run Now.** By default, you will see the Summary view as follows. Click on the **View Detail** button and click on the **Go** button.



**Why use Detail view?** You will usually need to do this each session e.g. each time you log into, so you can clearly see both commands and any run time error messages.The contents of **Practical1** script appear along with the ORACLE runtime messages.



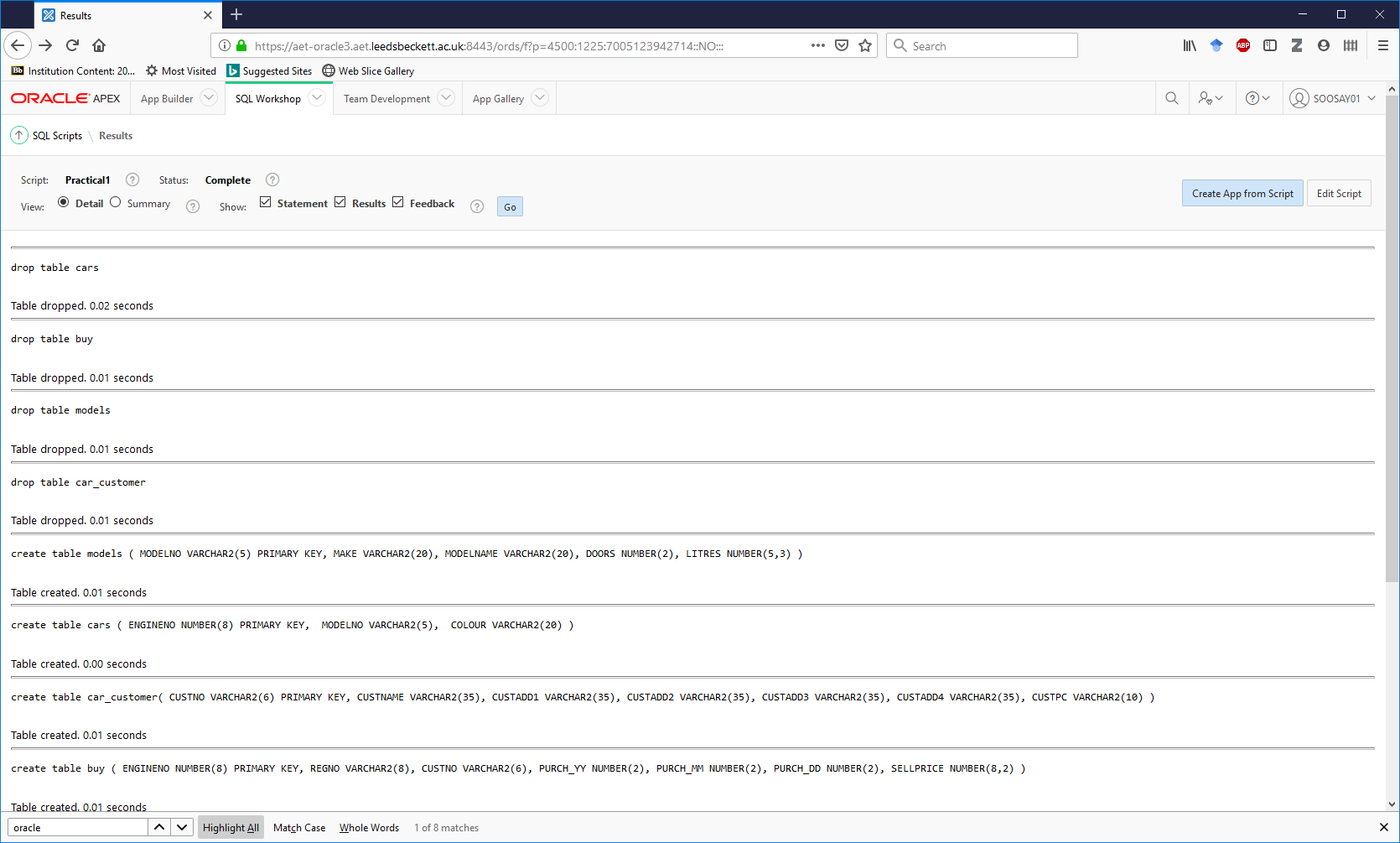
Examine the code - it creates 4 tables. Identify which tables and identify the structure of them. We use **drop table** in script files, as it prevents errors caused by trying to create a table that already exists.

**IMPORTANT – the CREATE TABLE command does NOT overwrite any existing table with the same name. You must DROP the existing table first, before you can recreate it!**

**Note: If you try to DROP a table that doesn’t exist you will get an error, but it doesn’t cause any problems. You can ignore it and go ahead and run the CREATE TABLE command to create your table.**

**Always remember to check the APEX output messages to ensure that your commands have run successfully.**

1. To amend the script **Practical1**, click on the **Edit Scrip**t button.

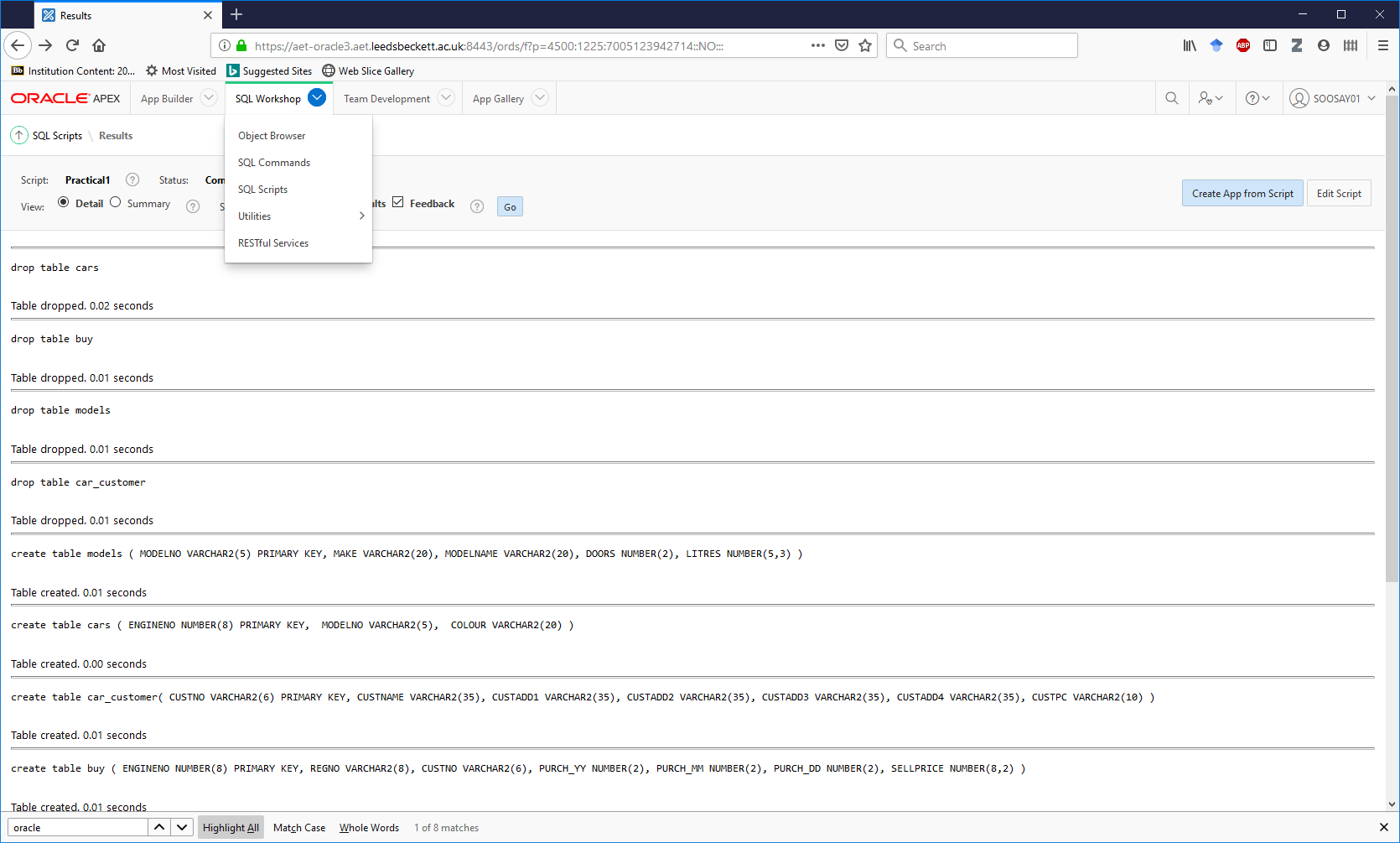


1. Make an amendment to the MODELS table to change the datatype of DOORS to NUMBER(1). This is sensible as you would never need more than one digit to represent the number of doors: you would never have a vehicle with more than 9 doors.
2. Then click the Run button which will both save and run the script click.
3. When the Run Script page appears, click on the **Run Now** button.
4. To examine the results, click on the **View Detail** button, you will see the MODELS table has been created and the datatype for DOORS is now NUMBER(1).

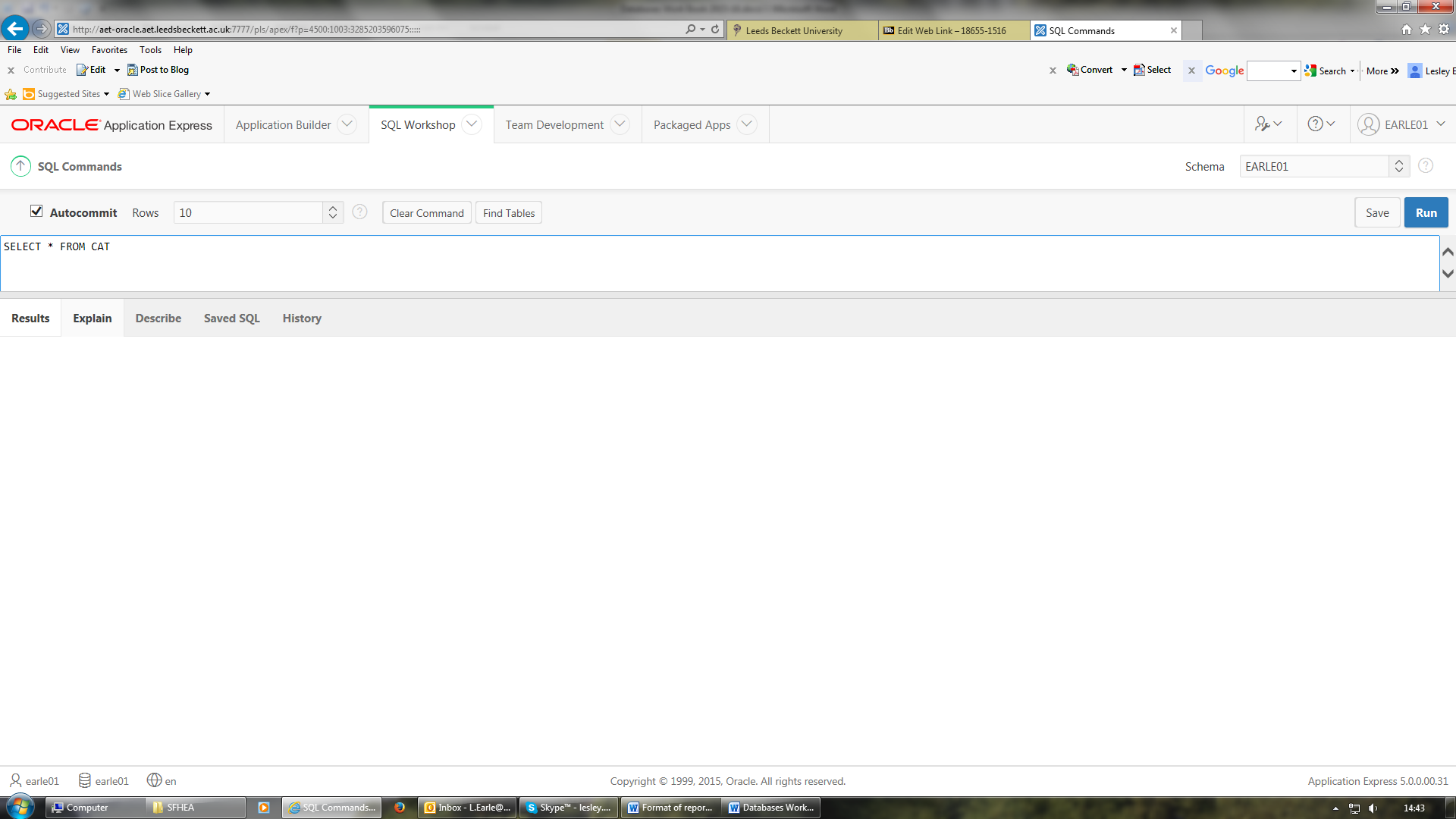
**Always check the output messages to make sure your tables have been created.**

**Note: As this is now the 2nd time you have run the script practical1.sql you will not get drop table errors as you did have the tables in your user area to drop.**

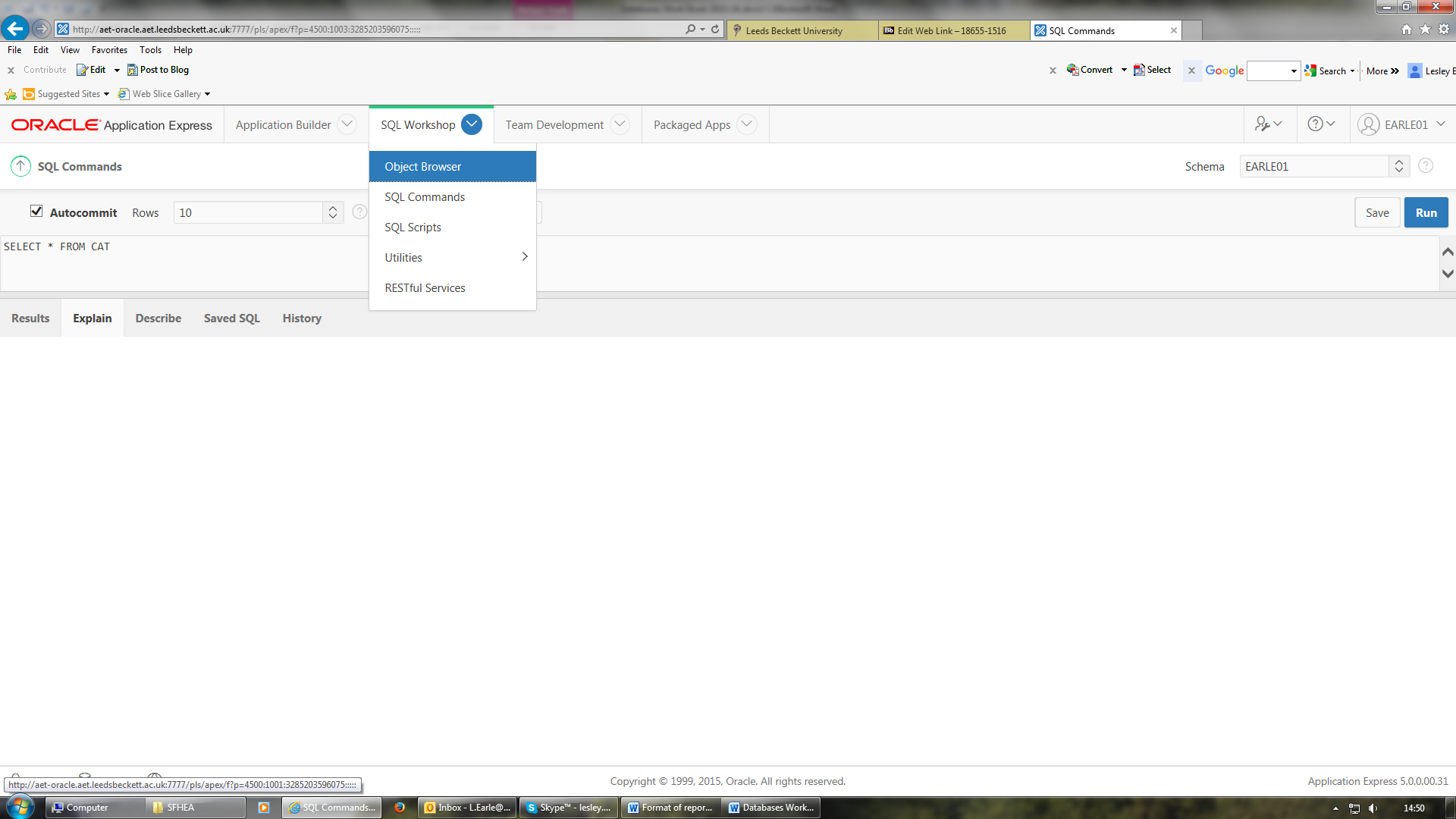
1. To check tables, you have access to you can use the **SQL Workshop, SQL Commands tool** available from the main tool bar at the top of the window.



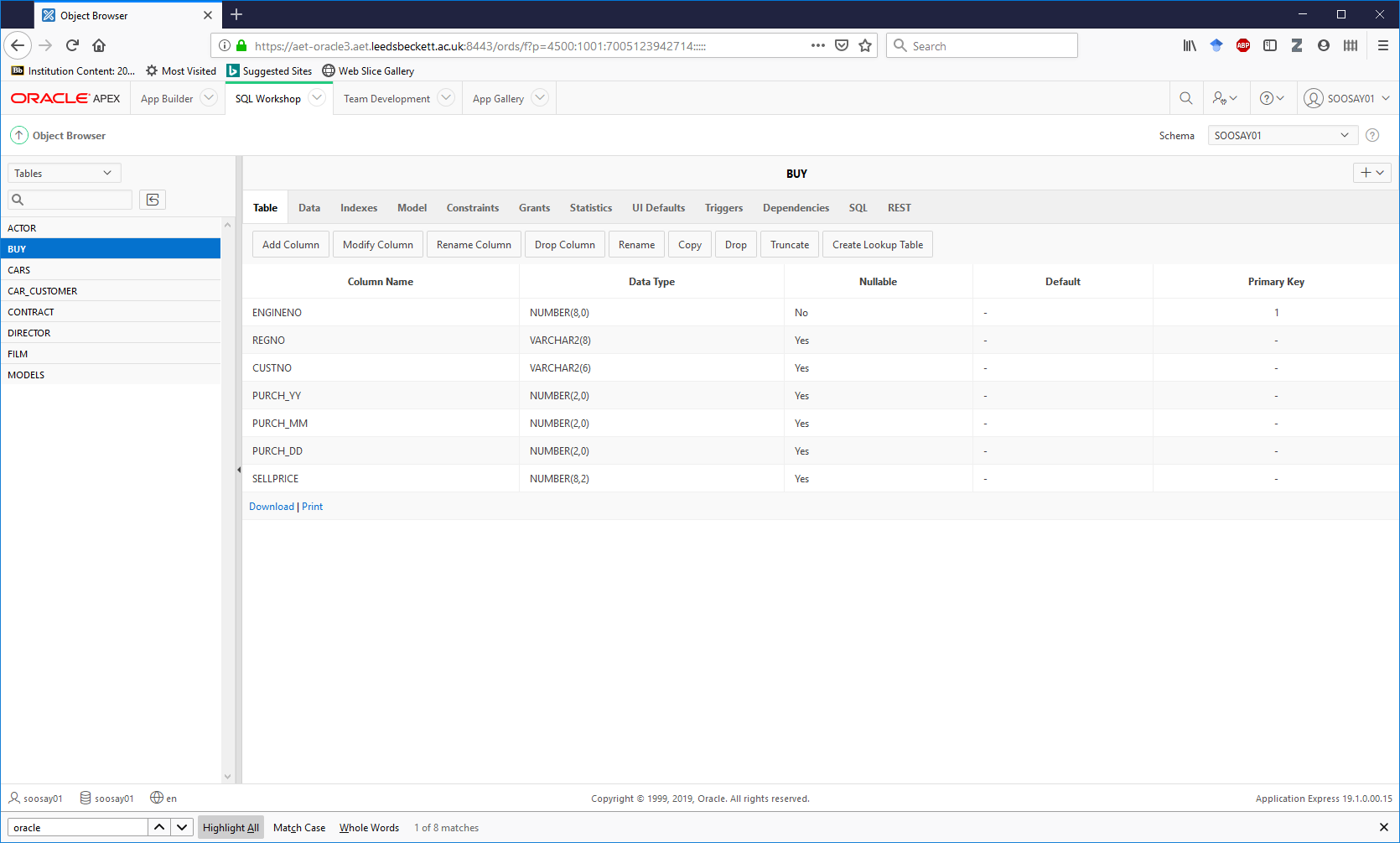
1. Type **Select \* from cat;** into the Command windowthenclick **Run**; this will list all the tables available in your user area (along with any Sequences and other objects) below the command.
2. Type Select in caps: **SELECT**. Does capitalisation matter?



NOTE: Alternatively, you can use the SQL Object Browser and select Tables from the drop-down list. Select **Object Browser**



1. Ensure Tables is selected from the drop-down list. The Object Browser also allows you to view and make changes to tables and data.



1. Go back to SQL Commands and enter then Run the following

**DESCRIBE cars*;***

This command can be shortened to **DESC** e.g. DESCcars*.*

What does the command **DESC** cars do?

You may prefer the Object Browser to using the command DESCRIBE to see what the structure of a table is. It’s up to you!

1. **You are now going to create a set of tables for a film system.** First look at the last page of this booklet to review the structure of the whole system.

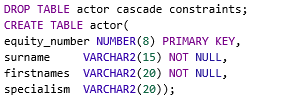
Now click on the **SQL Scripts** icon and then click on **Create** to create a new script file which you should name **film\_tables.** This will be used to hold the scripts for all 4 of the tables in the film system.

The actor table will be created first as it has no foreign keys. It has following table specification. Note the Primary Key is **equity\_number**.

Table Name: **actor**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Optional** | **Key** |
| equity\_number | NUMBER(8) |  | Primary Key |
| surname | VARCHAR2(15) | NOT NULL |  |
| firstnames | VARCHAR2(20) | NOT NULL |  |
| specialism | VARCHAR2(20) | YES |  |

**Then you should type in is below.** There are 2 commands, one to DROP the table and one to create the table. Note also how each of the two statements is terminated by a semi-colon. Commas are used as separators between the attributes. Note how we implement PRIMARY KEY and NOT NULL. Attributes which are optional just have the attribute name, type and length specified.



Click the **Run** button then **Run Now** to run the script **film\_tables** to create the actor table.

Scroll and check for errors until you see the ORACLE message **Table created**. Remember: if you try to DROP a table that doesn’t exist you will get an error. **This usually happens the first time you run a script but that’s ok, remember to ignore this type of error**!

1. You are now going to create another of the tables for the film system, **director**.  
     
   Open the same script file **film\_tables** by clicking on **Edit Script** and after the script for the actor table enter the SQL commands to create the table for **director** to match the following specification.

**Note**: Don’t forget to include the **DROP TABLE** command for this table.

Table Name: **director**

|  |  |  |  |
| --- | --- | --- | --- |
| **Column Name** | **Data Type** | **Optional** | **Key** |
| director\_id | NUMBER(4) |  | Primary Key |
| surname | VARCHAR2(15) | NOT NULL |  |
| firstnames | VARCHAR2(20) | NOT NULL |  |

Once your Drop table and Create Table commands have been typed in, click **RUN** then **RUN NOW**, make sure table creates correctly, if not correct the errors.

1. You are now going tocreate a new script file in order to insert rows of data into your director table.

Createa new script file called **filmdata** and type in the following:





Click **RUN** then, **Run Now** to run the script **filmdata.** Click **View Detail** to check the data has been inserted. If you have been successful the comment ‘**1 row inserted’** will appear for each of the rows inserted. Correct any errors if not.

1. **Use Edit Script** to reopen the script file called **filmdata** and write another **INSERT** command to add another director of your choice.   
    **Run it** and check results in **Detail** view to ensure that the new row has been inserted.

**NOTE** – As the rows for Robert Redford and John Ford have already been inserted into the table you will get an error **‘unique constraint violated’** for each of those rows. **It’s important to understand what this error means.**

Consider the wording of the error, it is telling you that you have violated the unique constraint which checks the primary key is unique for each row, you had already put in rows with director\_id values of 101 and 102 and you have now just tried to put them in again. So, don’t worry, **just remember you can’t have duplicate Primary Key values in a table** as director\_id was specified as the Primary Key. You don’t have to correct every error, understand them and then decide if any action is needed, in this case there is no action to take.

You should see the comment ‘**1 row inserted’** if your new row was inserted correctly. You can also check they are in your table by typing SELECT \* FROM director in SQL commands or use the Object Browser if you prefer.

1. Now **Edit Script** again to reopen the **filmdata** script file and at the end of the file enter **INSERT INTO** commands to insert some datafor 3 actors of your choice.

Make sure you check the structure of the table actor, it is not the same as director. There are 4 attributes for actor, use 001 as the equity number for the 1st actor and 002 for the 2nd, then 003 for the 3rd etc. The specialism data was not specified as NOT NULL so it is optional. For those with a specialism we want to use the following specialisms:

COMEDY, ACTION, HORROR, SCIFI, HISTORICAL

Use ‘’ the empty string, for any actors with no specialism.

1. Ensure you correct all the errors. Each row will not have been created until you see the ORACLE message ‘1 Row inserted’.

**HOW TO CREATE TABLES WITH FOREIGN KEYS**

Both tables you have created so far have had primary keys, as all tables should but no foreign keys. The next tables you will create have foreign keys.

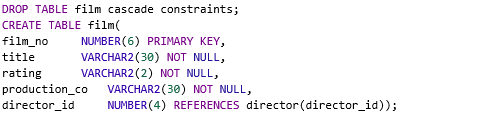
1. Open the **film\_tables** script again, you are going to create a table called **film** table with the following design, including the foreign key on director\_id.

Table Name: **film**

|  |  |  |  |
| --- | --- | --- | --- |
| **Table name:** | **film** |  | **KEYS** |
| **Column name** | **Data type** | **Optional** |  |
| film\_no | NUMBER(6) |  | **Primary Key** |
| title | VARCHAR2 (30) | NOT NULL |  |
| rating | VARCHAR2 (2) | NOT NULL |  |
| production\_co | VARCHAR2(30) | NOT NULL |  |
| director\_id | NUMBER(4) |  | Foreign Key reference to director(directory\_id) |

As the film table is at the many end of the 1:M relationship with director table it will need code to specify the foreign key for that relationship. It must also be created after the director table so go to the bottom of the script file. A foreign key is created by adding the code REFERENCES followed by the table name and primary key of the table it references.

Type in the following code:



**Note – Don’t forget the DROP TABLE command.**

**Make sure you check that the table has been created correctly.**

1. **You are now going to create the contract table which is specified below.**

There are 2 Foreign keys so you also need to include the references for the two foreign keys, in a similar way to the film table. This also means it must be created after the film table, so open the **film\_tables** script and at the end of the script enter the code to implement the following.

|  |  |  |  |
| --- | --- | --- | --- |
| **Table name:** | **contract** |  |  |
| **Column name** | **Data type** | **Optional** | **Keys** |
| contract\_no | NUMBER(6) |  | **Primary Key** |
| film\_no | NUMBER(6) |  | **Foreign** **Key** references film(film\_no) |
| equity\_number | NUMBER(8) |  | **Foreign** **Key** references actor(equity\_number) |
| contract\_date | DATE | NOT NULL |  |
| fee | NUMBER(8) | NOT NULL |  |

Run the script **film\_tables**

Ensure you correct all the errors. Your table will not have been created until you see the ORACLE message **Table created.**

You should now have all four tables created, each has the primary key defined and 2 tables have foreign keys.

**Check you have correct structure for the tables.** In the SQL commands tooltype in the following 4 commands:

**desc film;**

**desc actor;**

**desc director;**

**desc contract;**

Or you can look in the object browser but make sure you check your tables match the specifications at the end of this booklet, talk to your tutor if you are unsure whether they are correct.

1. It’s important to be able to understand ORACLE error messages and be able to identify and correct mistakes in your code quickly. Please refer to the **Common ORACLE Errors document** on the VLE in sessions 6 & 7 folders.

Now let’s look at some student system script files with errors.

Open the **Week 5 Find the errors in the code** document in the session 5 and 6 tutorial session materials folder and see if you can identify the errors in the CREATE TABLE statements and correct them.

Then, have a look at the file **Week 5 Find the errors in the code Solution,** to check if you have found them all.

1. There are a number of on-line resources available for ORACLE and SQL look at the <http://www.w3schools.com/sql/default.asp> which gives tutorials and quizzes on SQL.

Note there may be differences in SQL commands depending on which database product you are using.

1. Check out this useful resource <https://www.oracle.com/technical-resources/>. Search for one of the SQL commands you have used today e.g. DROP and look at the result of the search.
2. Further information on ORACLE APEX can be found at <http://www.oracle.com/technology/products/database/application_express/index.html>

**Check the boxes when you have completed the following Session 5 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 5 Reflection

|  |
| --- |
|  |

* Find the errors in table creation and inserting data

|  |
| --- |
|  |

* Do Session 5 quiz – table creation and data entry

|  |
| --- |
|  |

* Complete the GDPR materials

# Session 6 Inserting data and the implications of foreign keys

**Aims**

* Inserting data into tables with foreign keys.
* Implications of foreign keys when deleting data

**Background – READ THIS**

**In ORACLE we refer to PRIMARY and FOREIGN keys as constraints. A FOREIGN Key is a referential integrity constraint. Remember without correct keys we can’t link tables and extract the data that is required.**

From last week you have the 4 tables:- **actor, director, film** and **contract** which currently do not contain any data, as you have just recreated them in the last few tasks last week. We are now going to insert data into all the tables. However, this needs careful thought as there are some foreign keys in the system. The ERD below indicates where the foreign keys should be:

**FILM**

**DIRECTOR**

**CONTRACT**

**ACTOR**

directs

has

is a

The MANY end of a relationship will contain the FOREIGN KEY. Any data inserted into an attribute which has a foreign key MUST reference the attribute at the ONE end of the relationship and so make sure the table you are referencing has been created and the data being referenced already exists.

For the 1:M relationship between director and film, any data entered into the attribute director\_id in the film table MUST already exist in the primary key of the director table.

If we wanted to enter the data into the table **film** about the film Horse Whisperer, the director of the film Robert Redford’s data must be already in the table **director** as you are going to include the director\_id in the table **film**. So, we would need the rows to be entered in the following order:

Primary key for director matches with foreign key for film

INSERT INTO director VALUES (‘101’,’Redford’,’Robert’);

INSERT INTO film VALUES (‘1009’,’The Horse Whisperer’,’12’,’Warnar’, ‘101’);

The final attribute in **film** is the foreign key reference to the primary key of the table **director** e.g. 101.

**Tasks - DO THIS**

1. Now open the **filmdata** script you created last week, you should already have 3 directors including Robert Redford, John Ford and one you made up. It should also have the 3 actors you typed in the script. However, as you have not run this script since you last recreated the tables there will be no data yet in your tables.

You should now add the statement below to insert the data on the Horse Whisperer film into the film table. This statement must be typed at the end of the file, or more correctly it must come after the director data it refers to as there is a foreign key in film which refers to the director primary key.

**Type:**

INSERT INTO film VALUES (‘1009’,’The Horse Whisperer’,’12’,’Warnar’, ‘101’);

RUN the **filmdata** script and ensure that all 7 rows are inserted, if not correct the errors.

If you get a **foreign key constraint violated error** message, you have not got matching data in your foreign key.

Add another film of your choice to the bottom of the script, ensuring the director referred to in the last attribute already exists in the director table, RUN the script, correct any errors that need correcting.

1. The **Contact** tablehas two foreign keys so there will be similar references to the primary keys of the **film** and **actor** table.

Open the **filmdata** script and insert 2 rows of your own data into the **contract** table, they need to go at the end of the script as there are foreign keys on both film (you have 2 to choose from) and actor (you have 3 to choose from) and data must explicitly match the primary key values. When entering dates into a DATE column use the default date format MM/DD/YYYY e.g. ’02/12/2016’ or you may encounter problems when you do date comparisons

RUN the **filmdata** script and ensure that these 2 rows are inserted, if not correct the errors.

NOTE when you run the script, the error of **“unique constraint violated”** on the 1st 8 statements is just telling you that these rows are already in the tables, so ignore these errors. If you have been successful, the comment “**1 row created”** will appear for each of the 2 new rows of data inserted. If you have any errors, work through them.

If you get **foreign key constraint violated error** messages check your foreign key values against the primary key they are referencing.

1. Open the script file **filmdata** script again

Add more INSERT INTO commands to add at least two more rows of data of your choice into all the 4 tables: director, film, actor and contract tables. Make sure you plan this well, so you won’t get **“foreign key constraint violated”** errors.

Run the scriptand correct any errors which need correcting.

If you have completed the tasks correctly you will have 5 directors, 4 films, 5 actors and 4 contracts.

You should now implement the following system.



Faculty(**FacultyId**, FacultyName, Dean)

Room(**RoomNo**, Roomsize)

Tutor(**StaffId**, Surname, Firstname, Telno, *FacultyId, RoomNo*)

You will need to decide on the type and length of the attributes and for the data for Roomsize use:

SINGLE, DOUBLE, MULTIPLE SHARED

a) Create a new script file called **Rooms** and create the three tables defined above, don’t forget to include a primary key (in bold) for each table and the two foreign keys (in italics) in the tutor table. Note the order the tables are created in. Remember the table at the one end of a relationship MUST be created before the many end one (Remember you can’t reference a table that doesn’t exist).

b) Run the script file **Rooms** to create the tables.

c) Edit the same script file called **Rooms** and at the end of the script file, write insert commands to insert data for 2 faculties, 3 rooms and 4 tutors.

Don’t forget the constraints on the data in terms of data types and size of data.

Don’t forget the data at the **one** end of a relationship MUST exist before the **many** end, so the tutor data MUST be inserted last.

d) Run the script file to insert the data. Edit the file to correct any errors, starting at the top first.

**Typical Error Messages (Refer to Common ORACLE Errors document on the VLE)**

It’s important to understand Error Messages and their cause so you can avoid them or make appropriate corrections promptly.

e) Go to the **Object Browser** and look at each of these 3 tables, reviewing the tabs for table, data and constraints.

**Check the boxes when you have completed the following Session 6 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 6 Reflection

|  |
| --- |
|  |

* Find the errors in inserting data with foreign keys

|  |
| --- |
|  |

* Do Session 6 quiz – data entry with foreign keys

|  |
| --- |
|  |

* Complete your revision of week 1 to 4 materials ready for your assessment

# Session 7 Selecting data from a table

**Aims**

* Selecting all the data from a table
* Selecting some attributes from a table
* Selecting some rows from a table
* Selecting some rows and some attributes from a table
* Ordering the output from a select statement
* Using group by queries

**Background – READ THIS**

The simplest form of query is to select all the data from a table and the command to select all the data from a table called tutor is:

SELECT \* FROM tutor;

You can have the output ordered by particular attributes if you wish e.g.

SELECT \* FROM tutor

ORDER BY surname;

**Tasks – DO THIS**

1. You should already have some data in the table for the film system but in order to ensure you have all the tables and sufficient data for the queries upload to APEX and run the script called **filmdata18** which you will find on VLE in the weekly activities session 7 folder.

You may not have remembered the structure of the tables so to help you either look back at the last sessions notes or use the describe command. Also refer to the lecture slides for the syntax of the commands used for creating tables and inserting data.

Create a new script file called **session7** to write each of the following select statements.

**SELECTING ALL THE DATA FROM A TABLE**

1. Select all the data from the director table
2. Select all the data from the film table
3. Select all the data from actor table, order by surname
4. Select all the data from contract table, order by contract\_date

Run the script **session7**. Carefully examine the output to ensure you understand how each command has worked.

**SELECTING SOME ATTRIBUTES FROM A TABLE**

**Background – READ THIS**

To select particular attributes from a table, list them after SELECT instead of \* e.g.

SELECT staff\_id, telno

FROM tutor;

HINT: decide which table is needed and then decide which attributes are needed and in which order the columns should appear, if the rows should be ordered use ORDER BY

**Tasks – DO THIS**

Type the commands to implement the following into your **session7** script and Run, you might want to try each query one at a time.

1. Select the titles and rating of the films
2. Select the names of the directors (there is no single attribute called names, check the structure of the table)
3. Select the names of the actors
4. Select the rating, titles and production company of the films order by rating
5. Select the production companies and order by production company

Run the script **session7**

Carefully examine the output to ensure you understand how each command has worked.

Notice that you have some repeated production company names for i). Why is that?

Look at all the data in the table if you are not sure.

Why did it not happen for the other queries?

Note if you don’t want to see a column’s output repeated then you can use the optional DISTINCT clause before the column name.

1. Select the distinct production companies and order by production company

**SELECTING SOME ATTRIBUTES AND SOME ROWS**

**Background – READ THIS**

A WHERE clause is needed to select particular rows.

To select the names of the actors who have the specialism of COMEDY

SELECT surname, firstnames

Will only list the names of the actors with specialism of COMEDY

FROM actor

WHERE specialism = ‘COMEDY’;

Note queries are case sensitive for data so only rows where specialism column value has been entered in upper case will be output.

**Tasks – DO THIS**

k) Select the names of the actors who have a specialism of ACTION, order by surname

l) Select the titles and rating of the films where the rating is PG.

m) Select the titles and rating of the films where the rating is 12, order by title

n) Select the equity numbers of the actors paid over 7000000 for a film (Do you get repeated data? If so, why?)

o) Select the film number, the equity number and fee for the actors paid over 8000000.

p) Select the film number, the equity number and contract date for the actors paid over 8000000.

q) Select the title, director id and rating for films produced by DISNAY.

r) Select the equity numbers and fees for the film number 1015.

s) What is the director number of the director of the film The Horse Whisperer?

t) What is the average fee paid to actors? Use the Function AVG on fee e.g. AVG(fee).

1. Overview questions using the ROOMS system
2. What are the names and telephones of all the tutors?
3. Who is the Dean of each of the faculties?
4. Which room numbers are SINGLE rooms?
5. Which room numbers are MULTIPLE SHARED rooms?
6. What are the sizes of all the rooms ordered by room number?

**SELECTING USING GROUP BY AND FUNCTIONS**

**Background – READ THIS**

A group by clause can be used to group data together, using group by will output one row for **each different value** in the group by attribute specified e.g.

To select a list of all the different room sizes, use:

Will list all the different roomsizes

SELECT roomsize

FROM ROOM

Group by attribute

GROUP BY roomsize;

Better still you can use GROUP BY with functions COUNT, MIN, MAX, AVG, SUM.

To **count** the number of rooms of **each room size**, add a count of each roomsize using:

Will list all the different roomsizes and a count of the number of rooms of each roomsize

SELECT roomsize, COUNT(roomsize)

FROM ROOM

GROUP BY roomsize;

Group by attribute

**Tasks – DO THIS**

1. Using the film system:
2. Using GROUP BY output the different ratings there are.
3. Amend the command for (i) to list the number of films for each rating. Use COUNT.
4. Using the contract table how many films has each actor had contracts for?
5. How many films has each director directed?
6. How many films has each production company made?
7. What is the maximum fee paid to each actor?
8. What is the minimum fee paid to each actor?

**Check the boxes when you have completed the following Session 7 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 7 Reflection

|  |
| --- |
|  |

* Session 7 Find the errors in these simple student queries

|  |
| --- |
|  |

* Do Session 7 quiz – Week 7

**Session 8 Selecting data from more than one table**

**Aims**

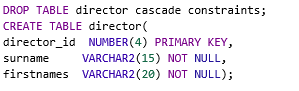
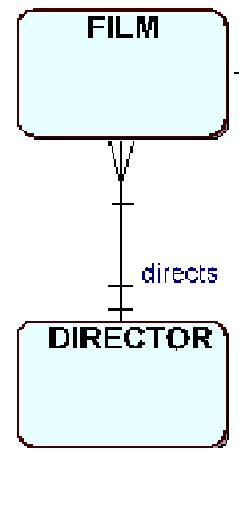
* To select data from more than one table

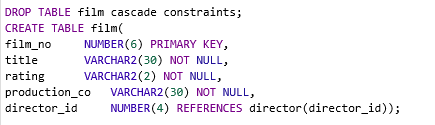
**Background – READ THIS**

**SELECTING DATA FROM MORE THAN ONE TABLE:**

Let’s look at an example, suppose we want to obtain the director ids, names and film titles for the PG rated films.

The director and the film tables are needed, these will be listed in the FROM clause





The attributes from the director table will be surname and firstnames from the director table. The title is from the film table. Note that director\_id is in both tables so dot notation must be used for this attribute and it doesn’t matter which table you take it from e.g. director.director\_id.

The join condition for these two tables is the PK of the one end director\_id from the director table and the appropriate FK director\_id from the many end table film. Include the join conditions after the WHERE keyword. As director\_id is in both tables as the PK/FK link then dot notation needs to be used e.g. film.director\_id to refer to the director\_id in the film table (the FK) and director.director\_id to refer to the director\_id on the director table (the PK).

Add any other conditions are included in the WHERE clause which restrict the rows to be output e.g. films with a PG rating. ORDER BY can be included for the attribute(s) being output if required.

Attributes, columns to be output, director\_id must use dot notation

**SELECT director.director\_id, surname, firstnames, title**

**FROM director, film** Tables

**WHERE director.director\_id = film.director\_id** Join with dot notation

**AND rating = ‘PG’;** Condition to restrict rows output

As required the above query, joins the tables using the FK in the film table and the PK in director table, selects the director\_id, surname and firstnames from the director table, the title from the film table and only the details for the PG films are selected.

**NOTE: if you get the error message ‘Column ambiguously defined’ you have forgotten to use dot notation for an attribute that is in more than one of the tables in the query.**

**Tasks – DO THIS**

1. You should already have some data in the tables for the film system as you will have run the script called **filmdata18** which you will find on VLE in the weekly activities session7 folder.

You may not have remembered the structure of the tables so to help you, look at the Film System Specification in the Appendix at the back of this book or refer to the Object Browser and refer to previous pages or the lecture slides for the syntax of the commands.

Create a new script file called **session8** in which to write and run each select statement.

Write queries which will produce the following. NOTE, you will need to join 2 tables for all these:

1. Select the title, rating and the surname of the director of all the films. Hint: you will need to join the film and the director tables.

Note: you should expect 1 row for every film, as film is at the many end of the relationship. If you get more rows you have forgotten the **join condition**, so every row of the film table is joined to every row of the director table (a Cartesian product), which is nonsense.

1. Select just the surname and first names of the actors who were paid more than 8000000.   
   Hint: use the **actor** and **contract** tables but don’t forget the join condition.

Note: Some actors names may appear more than once if they have contracts for more than one film.

1. Select the surnames of the directors who directed 18 rated films.
2. Select the surnames of the directors who directed films for DISNAY.
3. Select the equity number and contract date of the actors in the film Purassic Park.
4. Select the surnames of all the actors, their contract dates and film number, order by actor surname.
5. Select the equity numbers of all the actors, their contract dates and fees for the films produced by DISNAY, order by equity number.
6. Select the title, equity number and fee of films which had contracted actors who were paid more than 8000000.
7. Select the title and contract number of the films which contracted actors after 10/11/2015.

Note: When using dates ensure you use the default format ‘MM/DD/YYYY’

2. Using the **ROOMS** system

1. What are the surnames, telephone numbers, room numbers and dean of all tutors listed?
2. What are the faculty names, tutor names, telephone numbers and room numbers of all tutors, order by faculty name?
3. List the name of the dean and all their tutors’ names, order by the Dean?
4. How many tutors work in each named faculty? Note: use group by on two tables.

**Background – READ THIS**

**SELECTING DATA FROM MORE THAN ONE TABLE USING ALIAS FOR TABLENAMES**

When querying using 3 or more tables is often useful to use Aliases to save typing. This is when you specify usually a letter to refer to a table in the FROM clause of the query, it can then be used throughout the query.

Alias can be used here

**SELECT d.surname, d.firstnames, f.title**

**FROM director d, film f** Alias defined here

**WHERE d.director\_id = f.director\_id** Join condition using alias

**AND rating = ‘PG’;**

**Tasks – DO THIS**

**3. These require joining 3 tables, remember you will need 2 join conditions**.

1. Select the names of the actors who worked on PG films.
2. Select the titles of the films and the names of the directors and equity numbers of the actors who worked on them.
3. Select the titles of the films and the names of the actors and their contact date.
4. Select the titles of the films and the names of the actors who have contacts dated after 2rd Feb 2015.
5. Select the title of the film and the names of the actors who were contracted for the ‘The Horse Whisperer’ also output their contact date. NOTE data is case sensitive.
6. Select the names of actors and the names of the films and their fees, for those actors who have contracts after December 31st 2015.

**Challenges – you need all tables**

1. Select the titles of the films and the names of the directors and actors who worked on them.
2. Select the names of the directors who directed ACTION actors.

**Check the boxes when you have completed the following Session 8 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 8 Reflection

|  |
| --- |
|  |

* Session - Find the errors week 8

|  |
| --- |
|  |

* Do Session 8 quiz – Week 8

# Session 9 More GROUP BY HAVING and Subqueries

**Using GROUP BY HAVING**

**Background – READ THIS**

Group by outputs one row for every different value of the GROUP BY attribute specified. Group by having restricts this output to those rows which satisfy the HAVING clause. For example:

To output the room numbers and the number of people in each room which has more than one person, use:

SELECT roomno, COUNT(\*)

FROM TUTOR

Group by attribute

GROUP BY roomno

HAVING COUNT(\*) > 1;

Restricts the output to those with more than one tutor

This will output one row for each room with the count of the number of people in each room, but only for those rooms having more than one person.

**Tasks – DO THIS**

**Note these only need one table.**

1. Output the number of actors for each specialism with more than 3 actors, make sure you also output the name of each specialism not just the numbers, or the output will not make sense.
2. Output the number of films which have been directed by each director id who has directed more than one film.
3. Output the number of contracts for each film number having more than 1 contract.
4. Output the sum of the cost of contracts for each film number having less than 4 contracts.
5. Output the lowest and the highest fee for each film number with more than 1 contract.
6. Output the value of the contract with the lowest cost contract for each film, which is greater than 7000000.
7. Output the value of the contract with the highest fee for each film number with more than 1 contract for contracts issued after 1st January 2014. (You will need to use a WHERE condition).

**More Challenging**

1. Output the total value of all the contracts for each film, for contracts with more than 1 contract issued after 1st January 2014, ensuring each column has an appropriate heading (see lecture notes)

**The following use GROUP BY and more than one table so don’t forget the join condition**

1. Output the number of actors contracted for each named film with a rating of 12, give appropriate column headings
2. Output the number of contracts for each named film paying less than 18000000 in total fees.
3. Update Q6 to output the name of each film rather than just the film number (you will need to use 2 tables, don’t forget the join condition)
4. Output the number of films which have been directed by each named director who has directed more than one film (you will need to use 2 tables, don’t forget the join condition).

**SUBQUERIES**

**Background – READ THIS**

When trying to extract data which involves using a number of tables it is sometimes appropriate to use a subquery rather than a select command with lots of joins.

If you wanted to know which contracts paid higher fees than the average, you could have an inner query to find the average fee for contracts and an outer query to find the contracts which paid more than the average.

The query to find the average fee for contracts:

SELECT AVG (fee) FROM contract;

This value can then be used in the outer query as part of the WHERE clause:

SELECT contract\_no, fee

Outer query

FROM contract

WHERE fee > (SELECT AVG (fee) FROM contract);

Subquery in brackets

Notice how they are *nested* into one query.

You might want to change the query above to find the names of the actors paid higher than the average fee, this would need the inclusion of the actor table in the outer query.

SELECT firstnames, surname, contract\_no, fee

Two tables in the outer query

FROM contract, actor

WHERE contract.equity\_number = actor.equity\_number

AND fee > (SELECT AVG (fee) FROM contract);

If you know the inner query will only produce one value you can use =, <, > operators, if there will be potentially more than one value you must us IN. e.g. what are the names of the actors who had contacts issued after 10th October 2015.

Subquery could find more than one value so use IN here

SELECT surname

FROM actor

WHERE equity\_number IN (SELECT equity\_number FROM contract

WHERE contract\_date > ‘10/10/2015’);

**Tasks – DO THIS**

Using your film system, solve the following using **subqueries:**

1. Select the contract number and fee for the contracts with fees lower than the average fee. (The 1st example above should help)
2. Select the actor names, contract number and fee for the contracts with fees lower than the average fee. (The 2nd example above should help)
3. Select the names of the actors who had a contract issued on 1st Jan 2015.
4. Select the surnames of the directors who directed 18 rated films.
5. Select the contract date and fees of the ACTION actors.
6. Select the equity number and contract date of the actors in the film Purassic Park.
7. Select the actor’s names that have contracts for fees of more than 7000000.

**More Challenging examples. – READ THIS**

If you need to select the names of the actors that have worked for DISNAY, instead of a 3 table join, find the equity numbers of actors for DISNAY films (using the contract and film tables) then use the outcome to find the names of the actors were in those films, there could be more than one so use IN.

SELECT actor.surname, actor.firstnames

FROM actor where equity\_number IN

(SELECT equity\_number from contract, film

WHERE contract.film\_no = film.film\_no

AND production\_co = 'DISNAY');

This would also work:

SELECT actor.surname, actor.firstnames

FROM actor, contract where actor.equity\_number = contract.equity\_number AND film\_no IN

(SELECT film\_no from film

WHERE production\_co = 'DISNAY');

1. Select all the directors Johnnie Depp has worked with (i.e. the directors of the films the actor has worked in, just use the surname Depp).
2. Select all the actors who have acted in films which have a rating of PG.
3. Select all the actors who have worked for Universal production company.
4. Select all the actors the director Robert Redford has worked with, just use the surname Redford.

There are many SQL resources available on the web, but do start here: <https://docs.oracle.com/en/database/oracle/application-express/19.1/index.html> and the library has suitable books.

**Check the boxes when you have completed the following Session 9 activities on the VLE. Make sure you complete them before next week’s session.**

|  |
| --- |
|  |
|  |

* Complete all the Activities and Tutorial exercises, check against the answers
* Session 9 Reflection

|  |
| --- |
|  |

* Find the errors week 9

|  |
| --- |
|  |

* Do Session 9 quiz – Week 9

# Session 10 Creating Reports using APEX Application Builder

**Aims**

* **Create an application**
* **Create a classic report**
* **Create an interactive report**

**APEX Reports**

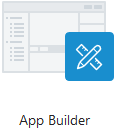
Most applications require reports which display data in an easy to understand format. Essentially your output would be called a report.

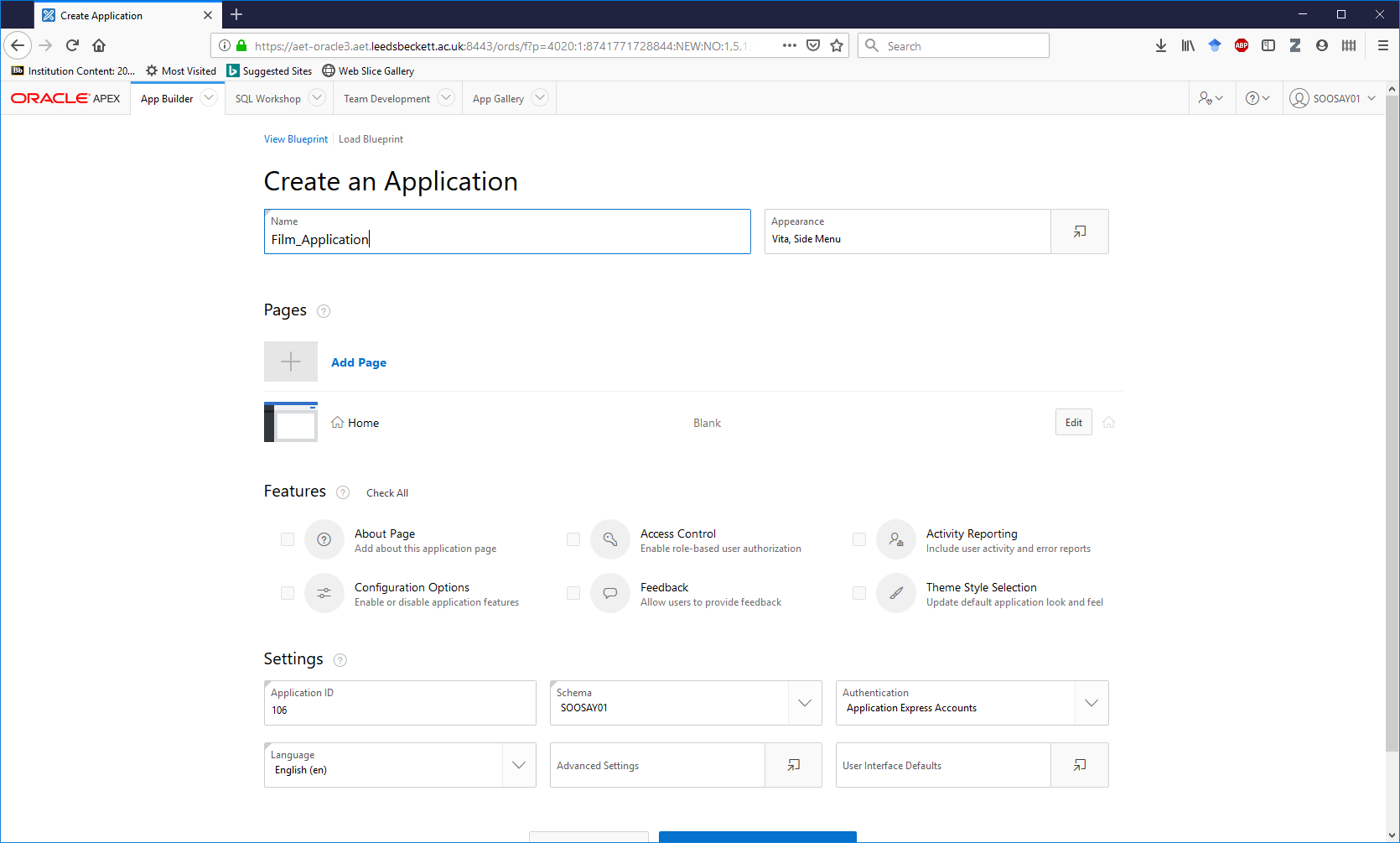
APEX Application Builder is used to produce applications, various types of reports and forms.

APEX has 2 types of reports - **interactive reports** and **classic reports.**

First - you should create an application starting with a home page.

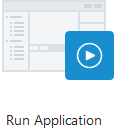
To create an application:

1. On the Workspace home page, click the **App Builder** icon. 
2. Click the **Create** icon.
3. Click **New Application** icon.
   1. For **Name**, enter a name to identify the application**:** **Film Application**
   2. Under **Pages –** Click **Add Page**. Twelve options appear - select **Report**
   3. You have the option generating the report using **Table or View** or **SQL Query.** Format is either as an **Interactive Report** or **Classic Report.**

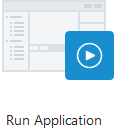


**Start by creating an interactive report on actors in the Film Application:**

* 1. Type in Page Name: **Actor\_Report**
  2. Select **SQL Query** and type a simple SQL: **SELECT \* FROM ACTOR.** Your report is based on this query.
  3. Select **Interactive Report**.
  4. Click **Add Page**.
  5. Check your entries and click **Create** **Application**!

1. Now click on **Run Application** icon  to see the blank page for the application. You will first be asked to log in, use your ORACLE login. You will then see your home page.
2. Click on **Actor\_Report**. The report is generated based on your SQL code and settings.
3. This is an interactive report. Meaning, you can change parameters by clicking on the column name headings. Play around with the interactive features to get familiar.

**Now create a classic report on films in the Film Application:**

1. Click on **Film Application**
2. Click on **Create Page.** Click on **Report** and select **Classic Report** format
3. Type in Page Name: **Film\_Report** and click **Next**
4. On the **Navigation Menu** page select the **second option**: *Create a new navigation menu entry* and choose **HOME** page for this. Click **Next**
5. Select **SQL Query** and type a simple SQL: **SELECT \* FROM FILM.** Your report is based on this query. Click **Create**
6. Now click on **Run Application** icon  to see the blank page for the application. You will first be asked to log in, use your ORACLE login. You will then see your home page.
7. Click on **Film\_Report** if it’s not displayed already. The report is generated based on your SQL code and settings.
8. This is a classic report with limited interaction. Click on the column name headings to sort the order of entries in the column.
9. NOTE – some browsers will show the report immediately others you need to click on the tab at the top of the screen.

**Continue by creating more classic reports in the Film Application:**

1. Create another classic report using a more complex query you have created in Session 8. Call it something appropriate, keep the Home page as the navigation menu entry, then **RUN** and check it.
2. Create another classic report containing the data from both the director and film tables (Don’t forget the join conditions). Call it Classic Film Director Report, keep the Home page as the navigation menu entry, then **RUN** and check it.

**Now practise creating more interactive reports in the Film Application:**

1. Create a new interactive report, named Interactive Actor Contract Report with the following query based on two tables:

SELECT surname, firstnames, actor.equity\_number, contract\_date, fee

FROM actor, contract

WHERE actor.equity\_number = contract.equity\_number

Investigate the actions on this report, particularly the group by actions.

1. Now create a new interactive report with the following query based on 3 tables:

SELECT title, surname, firstnames, actor.equity\_number, contract\_date, fee

FROM actor, contract, film

WHERE actor.equity\_number = contract.equity\_number

AND contract.film\_no = film.film\_no

Investigate the actions on this report, particularly the group by actions.

1. Now create a new interactive report with another query of your choice based on 3 tables

Investigate the actions on this report, particularly the group by actions.

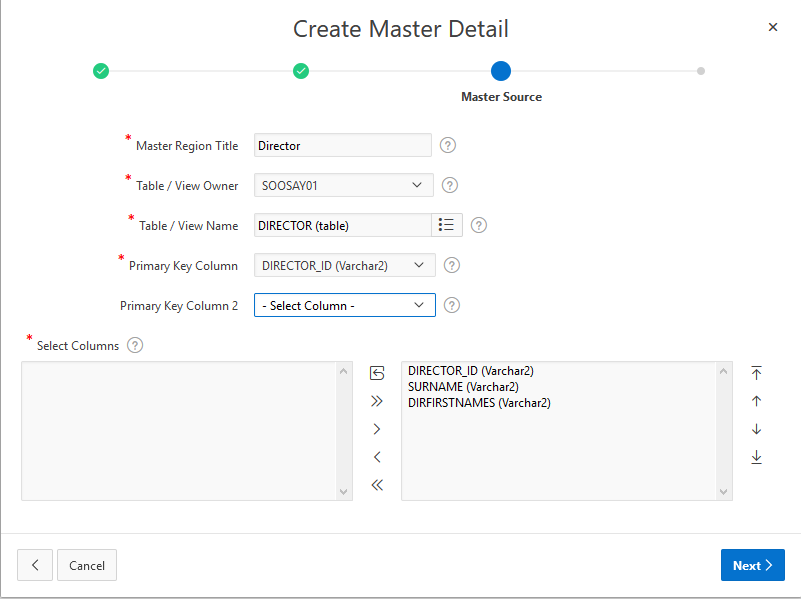
**HINT: in Assessment 2 - Practical Test you will need to have a report based on a 3-table query!**

# Session 11 Creating master detail forms using APEX Application Builder

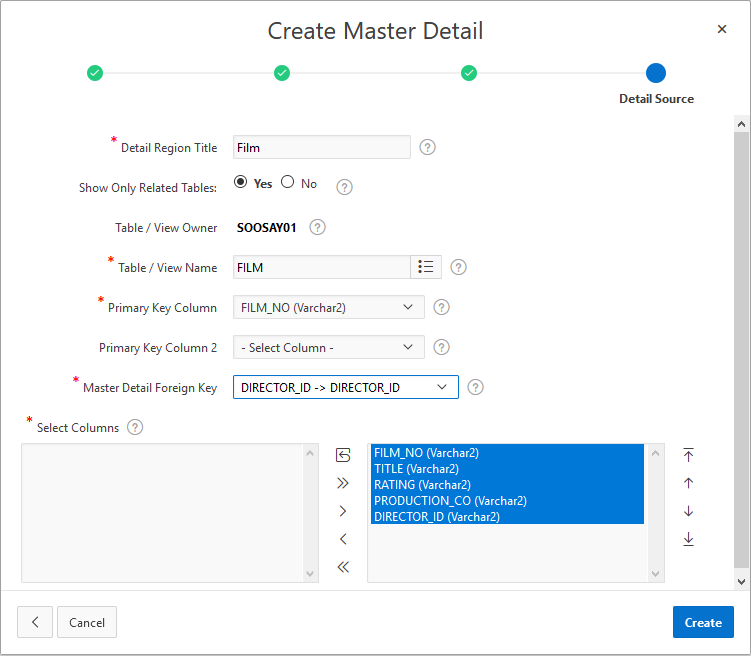
**Background**

FORMS are widely used in applications by users. A form might just be for a single table. **Master detail forms are based on 1:M relationships**. The master is the one side, the detail is the many end.

1. **We will use the FILM database - refer to the ERD on page 93 to view the 1:M relationships.**
2. Go to the Application Builder, click on **Film Application** you worked on in the last session**,** then click **Create Page.**
3. Choose **Master Detail -** Click **Next**.Choose **Stacked -** Click **Next**.Type Page Name as **Dir\_Film\_MD** and click **Next**.
4. Under **Navigation Menu - Create a new navigation menu** entry and under **Parent Navigation Menu Entry,** choose **Home** page for this. Click **Next**
5. Under **Create Master Detail – Master Source**, enter the following and click **Next**:

**

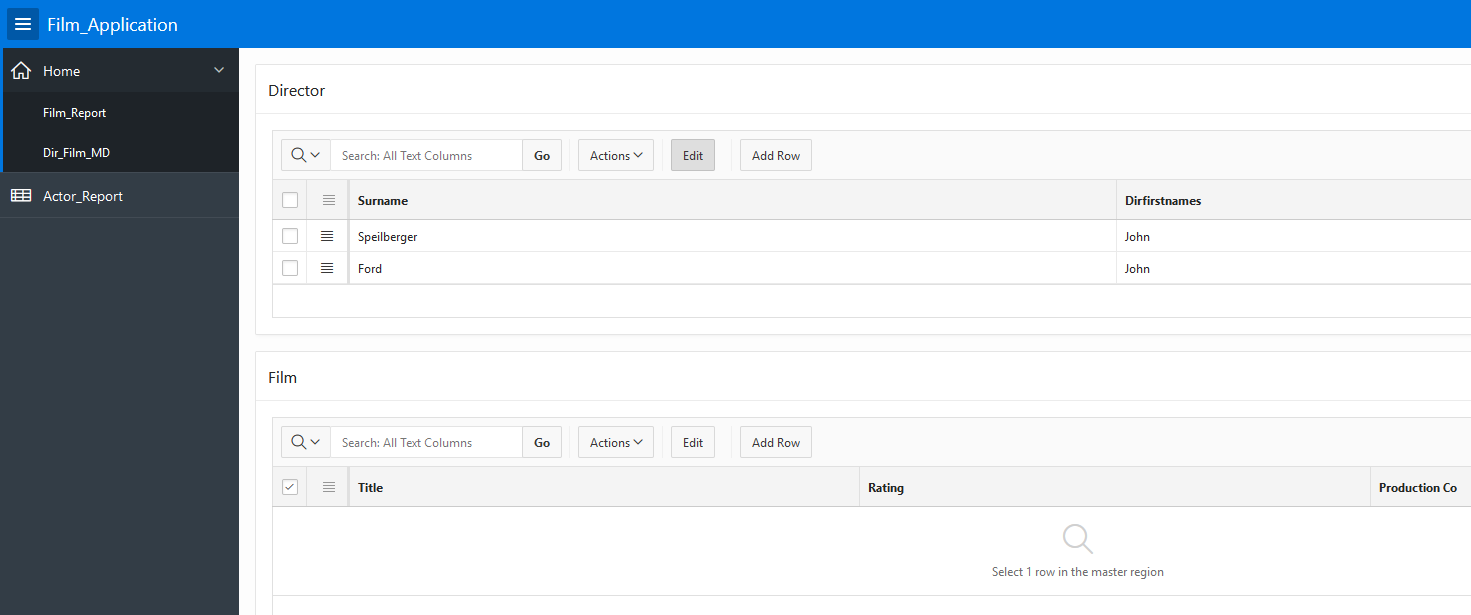
1. Under Create **Master Detail – Detail Source**, enter the following and click **Create**:

****

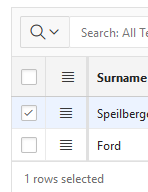
1. Click **RUN** . You will see the 2 tables data stacked on top of each other. You have the option of interacting with either the master or the detail without leaving the page.

**REMEMBER: A master is the table of the one end, the detail is the table of the many end.**

**Therefore, the Master-Detail forms show a 1:M relationship between tables.**



1. Now, tick on one of the master records to reveal the related detail record.



1. Practise:
   1. sorting rows,
   2. adding rows,
   3. deleting rows etc.
2. Continue creating other formats of Master Detail forms such as **Side by Side** or **Drill Down!**

**Remainder of the module**

You will have an opportunity to attempt the practice test in week 12, this will be one of your most important sessions this semester, make sure you do not miss it. Experience shows that students who do the practice test do significantly better in the final test.

We hope you have enjoyed studying this module which prepares you for further study next year. Try not to forget what you have learned and **keep this workbook for future reference**.

# Appendix A - Film System Specification (ERD and related Tables)

**FILM**

**DIRECTOR**

**CONTRACT**

**ACTOR**

directs

has

is a

**Table Name:** director

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints, Keys** |
| director\_id | NUMBER(4) | Primary Key |
| surname | VARCHAR2(15) | NOT NULL |
| firstnames | VARCHAR2(20) | NOT NULL |

**Table Name:** actor

|  |  |  |
| --- | --- | --- |
| **Column Name** | **Data Type** | **Constraints, Keys** |
| equity\_number | NUMBER(8) | Primary Key |
| surname | VARCHAR2(15) | NOT NULL |
| firstnames | VARCHAR2(20) | NOT NULL |
| specialism | VARCHAR2(20) |  |

|  |  |  |
| --- | --- | --- |
| **Table name:** | film |  |
| **Column name** | Data type | Constraints, Keys |
| **film\_no** | NUMBER(6) | Primary Key |
| **title** | VARCHAR2 (30) | NOT NULL |
| **rating** | VARCHAR2 (2) | NOT NULL |
| **production\_co** | VARCHAR2(30) | NOT NULL |
| **director\_id** | NUMBER(4) | Foreign key to director(director\_id) |

|  |  |  |
| --- | --- | --- |
| Table name: | contract |  |
| **Column name** | **Data type** | **Constraints, Keys** |
| contract\_no | NUMBER(6) | Primary Key |
| film\_no | VARCHAR2 (6) | Foreign key to film(film\_no) |
| equity\_number | VARCHAR2 (8) | Foreign key to actor(equity\_number) |
| contract\_date | DATE | NOT NULL |
| fee | NUMBER(8) | NOT NULL |