H DDD Clydeview

**Tasks and Answers**

Tasks

[Task 1 Design: Cardinality 1 1-1](#_Toc183811575)

[Task 2 Design: Cardinality 2 2-1](#_Toc183811576)

[Task 3 Design: Cardinality 3 3-1](#_Toc183811577)

[Task 4 Design: Entity Occurrence Diagrams 1 4-1](#_Toc183811578)

[Task 5 Design: Entity Occurrence Diagrams 2 5-1](#_Toc183811579)

[Task 6 Design: Compound Keys 6-1](#_Toc183811580)

[Task 7 Design: Entity Relationship Diagrams 1 7-1](#_Toc183811581)

[Task 8 Design: Entity Relationship Diagrams 2 8-1](#_Toc183811582)

[Task 9 Design: Data Dictionary 1 9-1](#_Toc183811583)

[Task 10 Design: Data Dictionary 2 10-1](#_Toc183811584)

[Task 11 Design: Queries that use another query 11-1](#_Toc183811585)

[Task 12 Design: Queries that use another query 12-1](#_Toc183811586)

[Task 13 Implementation: SQL Wildcards 13-1](#_Toc183811587)

[Task 14 Implementation: SQL Wildcards 14-1](#_Toc183811588)

[Task 15 Implementation: SQL Wildcards 15-1](#_Toc183811589)

[Task 16 Implementation: Calculations and Computed Fields 16-1](#_Toc183811590)

[Task 17 Implementation: Calculations and Computed Fields 17-1](#_Toc183811591)

[Task 18 Implementation: Computed Fields and Aliases 18-1](#_Toc183811592)

[Task 19 Implementation: Computed Fields and Aliases 19-1](#_Toc183811593)

[Task 20 Implementation: Computed Fields and Aliases 20-1](#_Toc183811594)

[Task 21 Implementation: Computed Fields and Aliases 21-1](#_Toc183811595)

[Task 22 Implementation: Aggregate Functions 22-1](#_Toc183811596)

[Task 23 Implementation: Aggregate Functions 23-1](#_Toc183811597)

[Task 24 Implementation: Aggregate Functions 24-1](#_Toc183811598)

[Task 25 Implementation: Aggregate Functions 25-1](#_Toc183811599)

[Task 26 Implementation: Problem Solving 26-1](#_Toc183811600)

[Task 27 Implementation: Problem Solving 27-1](#_Toc183811601)

[Task 28 Implementation: Queries that use another query 28-1](#_Toc183811602)

[Task 29 Implementation: Queries that use another query 29-1](#_Toc183811603)

**NB**. Each exercise is a separate section to assist with printing individual exercises.

Answers

[A1 Design: Cardinality 1 1-1](#_Toc184760634)

[A2 Design: Cardinality 2 2-1](#_Toc184760635)

[A3 Design: Cardinality 3 3-1](#_Toc184760636)

[A4 Design: Entity Occurrence Diagrams 1 4-1](#_Toc184760637)

[A5 Design: Entity Occurrence Diagrams 2 5-1](#_Toc184760638)

[A6 Design: Compound Keys 6-1](#_Toc184760639)

[A7 Design: Entity Relationship Diagrams 1 7-1](#_Toc184760640)

[A8 Design: Entity Relationship Diagrams 2 8-1](#_Toc184760641)

[A9 Design: Data Dictionary 1 9-1](#_Toc184760642)

[A10 Design: Data Dictionary 2 10-1](#_Toc184760643)

[A11 Design: Queries that use another query 11-1](#_Toc184760644)

[A12 Design: Queries that use another query 12-1](#_Toc184760645)

[A13 Implementation: SQL Wildcards – Done 13-1](#_Toc184760646)

[A14 Implementation: SQL Wildcards – Done 14-1](#_Toc184760647)

[A15 Implementation: SQL Wildcards – Done 15-4](#_Toc184760648)

[A16 Implementation: Calculations and Computed Fields – Done 16-1](#_Toc184760649)

[A17 Implementation: Calculations and Computed Fields – Done 17-1](#_Toc184760650)

[A18 Implementation: Computed Fields and Aliases – Done 18-1](#_Toc184760651)

[A19 Implementation: Computed Fields and Aliases – Done 19-1](#_Toc184760652)

[A20 Implementation: Computed Fields and Aliases - Done 20-1](#_Toc184760653)

[A21 Implementation: Computed Fields and Aliases - Done 21-1](#_Toc184760654)

[A22 Implementation: Aggregate Functions – Done 22-1](#_Toc184760655)

[A23 Implementation: Aggregate Functions – Done 23-1](#_Toc184760656)

[A24 Implementation: Aggregate Functions – Done 24-1](#_Toc184760657)

[A25 Implementation: Aggregate Functions 25-1](#_Toc184760658)

[A26 Implementation: Problem Solving 26-1](#_Toc184760659)

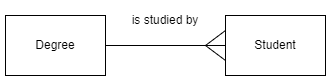
[A27 Implementation: Problem Solving 27-1](#_Toc184760660)

[A28 Implementation: Queries that use another query – Done 28-1](#_Toc184760661)

[A29 Implementation: Queries that use another query – Part 29-1](#_Toc184760662)

# Design: Cardinality 1

1. State the cardinality that exists between the following entities.
   1. PRESIDENT and COUNTRY
   2. PUPIL and SCHOOL SUBJECT
   3. ORCHESTRA and MUSICIAN
   4. BOOK and AUTHOR
   5. COUNTRY and CITY
2. For each of the following, identify two entities and the cardinality of the relationship between the entities.
   1. A shop employs many workers. A worker is employed by, at most, one shop
   2. A manager manages, at most, one department. A department is managed by, at most, one manager.
   3. A holiday resort has many hotels. Each hotel is located in exactly one resort.
   4. A team consists of many players. A player plays for only one team.
   5. A lecturer teaches, at most, one course. A course is taught by exactly one lecturer.
   6. A flight connects two airports. An airport is used by many flights.
   7. An order may be for many products. A product may appear on many orders.
   8. A customer may submit many orders. An order is for exactly one customer.
3. Identify the cardinality between the Degree and Student entities in the entity relationship diagram below.



Describe the ‘is studied by’ relationship.

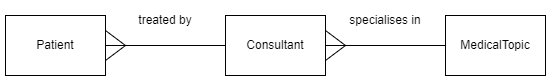
1. A hospital has many wards and each ward can be used to look after one or more patients. The hospital stores details of WARDs and PATIENTs in two separate entities.
   1. Write down at least four attributes that would be stored in each of the WARD and PATIENT entities.
   2. State the cardinality of the relationship between the WARD and PATIENT entities.
   3. Create an entity relationship diagram to model these two entities. Your diagram should indicate:

* the name of each entity
* all of the attributes listed in part (a)
* the name of each relationship
* the cardinality of each relationship

1. A music CD can contain many tracks. A track can appear on many different CDs (for example, on a compilation CD, a greatest hits CD etc). Each CD features one or more artists (a band or a solo artist) and successful artists will feature on many CDs.
   1. Write down at least four attributes that would be stored in each of the CD, TRACK and ARTIST entities.
   2. State the cardinality of the relationship between the CD and TRACK entities.
   3. State the cardinality of the relationship between the CD and ARTIST entities.
   4. Create an entity relationship diagram to model these three entities. Your diagram should indicate:

* the name of each entity
* all of the attributes listed in part (a)
* the name of each relationship
* the cardinality of each relationship

1. Identify the cardinality of the relationship between each pair of entities in the entity relationship diagram below.



Describe each relationship in this entity relationship diagram.

1. Repeat for the entity relationship diagram below.



# Design: Cardinality 2

## Part 1

SurfScotland is a blog used by members to share information about surfing in Scotland. A relational database is used to store details of members and blog posts in two related tables, Member and Post.

### Member Table

|  |  |  |  |
| --- | --- | --- | --- |
| **MemberID** | **Lastname** | **Firstname** | **Email** |
| 0001 | Davies | Jim | jimbo31@scotmail.co.uk |
| 0002 | McKay | Ann | mckaya218@hotmail.com |
| 0003 | Roberts | Carol | croberts123@teachers.com |
| 0004 | Singh | Hardeep | singh832@scotmail.co.uk |

### Post Table

|  |  |  |  |
| --- | --- | --- | --- |
| **PostID** | **Title** | **Date** | **MemberID** |
| 0001 | Welcome to the SurfScotland blog | 2024-08-01 | 0001 |
| 0002 | Belhaven Bay Dunbar | 2024-08-08 | 0001 |
| 0003 | Coldingham Bay Scottish Borders | 2024-08-13 | 0001 |
| 0004 | Hebridean Surf Lewis | 2024-08-15 | 0002 |
| 0005 | Broch Open Surf Competition | 2024-08-15 | 0004 |

1. State the cardinality that exists between the Member and Post entities
2. Describe the type of relationship that exists between the Member and Post entities.

## Part 2

ScotBank uses a relational database to store information about customers and the different types of accounts that they have.

### AccountType Table

|  |  |
| --- | --- |
| **AccountType** | **Account** |
| 01 | Current |
| 02 | Savings |
| 03 | Mortgage |
| 04 | Loan |

### Customer Table

|  |  |  |  |
| --- | --- | --- | --- |
| **CustomerID** | **Lastname** | **Firstname** | **AccountType** |
| 0001 | Davies | Jim | 01 |
| 0001 | Davies | Jim | 02 |
| 0002 | Mckay | Ann | 01 |
| 0002 | Mckay | Ann | 03 |
| 0003 | Roberts | Carol | 02 |
| 0003 | Roberts | Carol | 03 |
| 0003 | Roberts | Carol | 04 |
| 0004 | Singh | Hardeep | 01 |
| 0004 | Singh | Hardeep | 02 |
| 0004 | Singh | Hardeep | 03 |

1. State the cardinality that exists between the Customer and AccountType entities.
2. Describe the type of relationship that exists between these entities.

## Part 3

The RetroClothing website uses a relational database to store details of items of clothing for sale and the brand of each item.

### Item Table

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ItemID** | **Description** | **Size** | **Era** | **BrandID** |
| 0001 | Red swim suit | 10 | 1950s | 003 |
| 0002 | Floral dungarees playsuit | 10 | 1990s | 002 |
| 0003 | Brown swing coat | 16 - 18 | 1960s | 005 |
| 0004 | Circle skirt black white polka dot | 12 - 14 | 1950s | 004 |
| 0005 | Floral print hostess dress | 10 | 1970s | 005 |

### Brand Table

|  |  |  |
| --- | --- | --- |
| **BrandID** | **Brand** | **Nationality** |
| 001 | Valentino | Italian |
| 002 | Mary Quant | British |
| 003 | Rose Marie Reid | US |
| 004 | Kiki Byrne | Norwegian |
| 005 | Susan Small | British |

### Founder Table

|  |  |  |  |
| --- | --- | --- | --- |
| **BrandID** | **Firstname** | **Surname** | **DOB** |
| 001 | Valentino | Garavani | 1932-05-11 |
| 002 | Mary | Quant | 1934-02-11 |
| 003 | Rose | Yancey | 1906-09-12 |
| 004 | Olaug | Grinaker | 1937-04-18 |
| 005 | Leslie | Jones | 1904-07-21 |

1. State the cardinality that exists between the Item and Brand entities.
2. State the cardinality that exists between the Brand and Founded entities.
3. Describe the type of relationship that exists between each pair of entities.

## Part 4

GeoCity is a website that is used by primary children to learn about world geography. A relational database is used to store details of Continents, Countries and their Capital Cities in two related tables.

### Continent Table

|  |  |  |
| --- | --- | --- |
| **Continent** | **Area(km2)** | **Population** |
| Europe | 10,180,000 | 741,447,158 |
| Africa | 30,370,000 | 1,225,080,510 |
| America | 42,549,000 | 1,001,559,000 |
| Antarctica | 14,000,000 | 1,106 |
| Australia | 8,600,000 | 35,000,000 |
| Asia | 44,579,000 | 4,462,676,731 |

### Country Table

|  |  |  |  |
| --- | --- | --- | --- |
| **CountryCode** | **Country** | **Continent** | **CapitalID** |
| BG | Bulgaria | Europe | 001 |
| CAM | Cameroon | Africa | 002 |
| IC | Iceland | Europe | 003 |
| LV | Latvia | Europe | 004 |
| WG | Grenada | Americas | 005 |

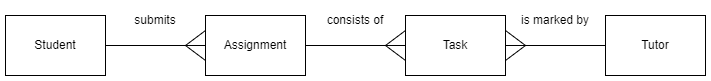
### Capital Table

|  |  |  |
| --- | --- | --- |
| **CapitalID** | **Capital** | **Population** |
| 001 | Sophia | 1300000 |
| 002 | Yaounde | 750000 |
| 003 | Reykjavik | 84000 |
| 004 | Riga | 9000000 |
| 005 | Saint Georges | 30000 |

1. State the cardinality that exists between the Continent and Country entities.
2. State the cardinality that exists between the Country and Capital entities.
3. Describe the type of relationship that exists between each pair of entities.

# Design: Cardinality 3

1. Consider the entity relationship diagram shown below.



* 1. State the cardinality of the relationship between the Student and Assignment entities.
  2. State the cardinality of the relationship between the Assignment and Task entities.
  3. State the cardinality of the relationship between the Task and Tutor entities.
  4. Describe each relationship in the entity relationship diagram.

1. Consider the entities MOVIE, ACTOR and DIRECTOR.
   1. Write down at least 4 attributes that would be stored in each entity.
   2. State the cardinality of the relationship between the MOVIE and ACTOR entities.
   3. State the cardinality of the relationship between the MOVIE and DIRECTOR entities.
   4. Draw an entity relationship diagram to illustrate the relationships between the MOVIE, ACTOR and DIRECTOR entities. The ERD should show the entities, attributes and the relationships.
2. Consider the entities Continent, Country and City.
   1. Write down at least 4 attributes that would be stored in each entity.
   2. State the cardinality of the relationship between the Continent and Country entities.
   3. State the cardinality of the relationship between the Country and City entities.
   4. Draw an entity relationship diagram to illustrate the relationships between the Continent, Country and City entities. The ERD should show only the entities and the relationships.

# Design: Entity Occurrence Diagrams 1

1. The following tables shows which bookings have been made by which customers.

|  |  |
| --- | --- |
| **Booking** | **Customer** |
| BD223145 | 1234 |
| NF874607 | 3456 |
| KL763832 | 3456 |
| TG803017 | 2804 |
| JT397282 | 1234 |

Draw an entity occurrence diagram to represent the relationship between the Booking and Customer entities.

1. Consider the details stored in the Driver and Taxi entities shown below. Draw an entity occurrence diagram to represent the relationship between the Driver and Taxi entities.

### Table Driver

|  |  |  |
| --- | --- | --- |
| **Driver ID** | **Driver Name** | **Taxi** |
| 03 | Jo Fleming | TY528UQ |
| 15 | Ian Smith | AS512ER |
| 18 | Pat Page | VJ537KL |

### Table Taxi

|  |  |  |
| --- | --- | --- |
| **Registration Num** | **Carries** | **Accessible Access** |
| TY528UQ | 5 | Yes |
| AS512ER | 4 | No |
| VJ537KL | 8 | Yes |

1. Consider the details stored in the Course, Tutor and Student tables shown below. Draw an entity occurrence diagram to represent the relationship between:
   1. the Course and Tutor entities.
   2. the Student and Course entities.

### Table Courses

|  |  |  |  |
| --- | --- | --- | --- |
| **Course ID** | **Name** | **Credits** | **Tutor ID** |
| D556 | Computing | 1 | 1989 |
| D766 | Hardware | 1 | 7817 |
| C991 | DB Design | 2 | 7817 |
| M101 | Calculus | 2 | 3456 |

### Table Tutor

|  |  |  |
| --- | --- | --- |
| **Tutor ID** | **Tutor Name** | **Phone Num** |
| 1989 | Mr Boyd | 7618 |
| 7817 | Mrs Harris | 9914 |
| 6516 | Mr George | 1233 |
| 3456 | Ms Cross | 5052 |

### Table Students

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **StudentNo** | **Firstname** | **Surname** | **DOB** | **CourseID** |
| 18728 | Joseph | Millar | 2006-02-06 | D556 |
| 18728 | Joseph | Millar | 2006-02-06 | D766 |
| 18728 | Joseph | Millar | 2006-02-06 | C991 |
| 18729 | Anna | Polizi | 2005-06-14 | D556 |
| 18729 | Anna | Polizi | 2005-06-14 | M101 |

1. Consider the details stored in the Teacher, Class and Pupil tables shown below. Draw an entity occurrence diagram to represent the relationship between:
   1. the Teacher and Class entities.
   2. the Pupil and Class entities.

### Table Teacher

|  |  |  |  |
| --- | --- | --- | --- |
| **TeacherNo** | **Class** | **Name** | **Department** |
| 123 | E23 | Green | English |
| 145 | S79 | Black | Science |
| 168 | M29 | Brown | Music |
| 123 | E31 | Green | English |

### Table Class

|  |  |  |
| --- | --- | --- |
| **Class Code** | **Subject** | **Level** |
| E23 | Literature | Higher |
| M29 | Performance | Higher |
| E31 | Textual Analysis | Higher |
| S79 | Physics | N5 |

### Table Pupil

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pupil ID** | **Firstname** | **Lastname** | **DOB** | **Class** |
| 40438271 | Alan | Adams | 2011-02-10 | E23 |
| 40438271 | Alan | Adams | 2011-02-10 | E31 |
| 39879342 | Barnaby | Brambles | 2012-07-21 | S79 |
| 37690065 | Carl | Crookston | 2012-10-31 | M29 |
| 37690065 | Carl | Crookston | 2012-10-31 | E23 |
| 37690065 | Carl | Crookston | 2012-10-31 | E31 |
| 41414141 | Dhavid | Dhesi | 2012-12-12 | M29 |
| 41414141 | Dhavid | Dhesi | 2012-12-12 | S79 |

1. Consider the details stored in the Employee, Pilot and Admin tables shown below. Draw an entity occurrence diagram to represent the relationship between:
   1. the Employee and Pilot entities.
   2. the Employee and Admin entities.

### Table Pilot

|  |  |
| --- | --- |
| **EmployeeID** | **Grade** |
| 12345 | 2 |
| 15050 | 3 |
| 19753 | 2 |

### Table Admin

|  |  |
| --- | --- |
| **EmployeeID** | **Job Title** |
| 13579 | Personal Assistant |
| 17531 | Booking Agent |

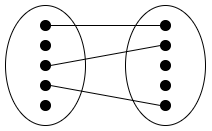
### Table Employee

|  |  |  |
| --- | --- | --- |
| **EmployeeID** | **Firstname** | **Surname** |
| 12345 | Harold | Hamilton |
| 13579 | Lynn | Lindsay |
| 15050 | Paula | Franelli |
| 17531 | Thomas | Moore |
| 19753 | Phil | Avis |

# Design: Entity Occurrence Diagrams 2

## Part 1

1. Which entity occurrence diagram represents a one-to-one relationship?

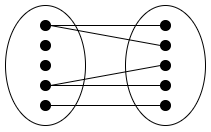
 

**A B**

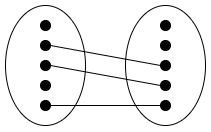
**C D**

1. Which entity occurrence diagram represents a many-to-many relationship?

**A B**

A black and white diagram with circles and lines

Description automatically generated 

**C D**

1. What type of relationship is illustrated in the following entity occurrence diagram?

A black and white diagram of a network

Description automatically generated

## Part 2

1. Draw an entity occurrence diagram to represent the relationship between the Customer and Rental entities. Sample data stored in these entities has been provided below.

### Table Customer

|  |  |  |  |
| --- | --- | --- | --- |
| **CustCode** | **First** | **Name** | **Phone** |
| 1001 | Anne | Jones | 01111111111 |
| 1002 | Sam | McKay | 02222121212 |
| 1003 | Jim | Shaw | 01213132123 |

### Table Rental

|  |  |  |
| --- | --- | --- |
| **ID** | **CustCode** | **Registration** |
| 1 | 1001 | AB12 JKL |
| 2 | 1003 | BA32 MKL |
| 3 | 1001 | CD41 PLM |
| 4 | 1002 | AB22 MNB |
| 5 | 1002 | BA32 MKL |
| 6 | 1001 | AB22 MNB |

1. State the cardinality of the relationship between these two entities.
2. Draw an entity occurrence diagram to represent the relationship between the Rental and Vehicle entities. Sample data stored in the Vehicle entity has been provided below; sample data in the Rental entity was above.

### Vehicle Table

|  |  |  |
| --- | --- | --- |
| **Registration** | **Make** | **Model** |
| AB12 JKL | Ford | Taurus |
| AB22 MNB | Vauxhall | Corsa |
| BA32 MKL | Ford | Focus |
| CD41 PLM | Fiat | 500L |

1. State the cardinality of the relationship between these two entities.

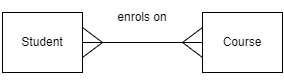
# Design: Compound Keys

## Part 1

Many-to-many (M:N) relationships occur naturally but are implemented by adding an intersect entity, also known as a relationship entity, between the two entities that have the M:N relationship.

If the primary keys of the two entities are used together to create the primary key of the intersect entity, the primary key is known as a compound key.

1. Analysis of the courses that students enrol on shows there is a many-to-many relationship between the entities.

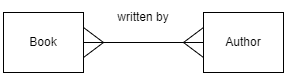


Resolve the M:N relationship by creating a new entity and two relationships: 1:M and M:1.

A diagram of a diagram

Description automatically generated

1. Analysis of books shows that there is a M:N relationship between the books and the authors.

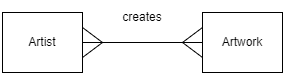


Resolve the M:N relationship by creating a new entity and two relationships: 1:M and M:1.

A diagram of a book written by author

Description automatically generated

1. Analysis of artwork shows that there is a many-to-many relationship between art and the artists that create it.



Resolve the M:N relationship by creating a new entity and two relationships: 1:M and M:1.

A diagram of a artist

Description automatically generated

## Part 2

The Barra Movie Database, known as BMDb, has become too large to manage on paper any more. The BMDb plans to move to a SQL database. An example of the data that is currently held is shown below.

### Data

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **DoB** | **Nationality** | **Film** | **Genre** | **Release Date** |
| Tom Hanks | 1956-07-09 | American | Forrest Gump | Drama | 1994-07-06 |
| Meryl Streep | 1949-06-22 | American | The Devil Wears Prada | Comedy | 2006-06-30 |
| Julie Andrews | 1935-10-01 | British | The Sound of Music | Musical | 1965-03-02 |
| Tom Hanks | 1956-07-09 | American | Saving Private Ryan | War | 1998-07-24 |
| Anne Hathaway | 1982-11-12 | American | The Devil Wears Prada | Comedy | 2006-06-30 |

1. From the data, identify two entities.
2. Create an entity-occurrence diagram from the data.
3. Create an entity-relationship diagram. List the attributes, and ensure that all primary and foreign keys are identified.

# Design: Entity Relationship Diagrams 1

1. Draw an entity relationship diagram to model the following set of conditions relating to Students, Courses and Lecturers.

* Each student is on only one course
* Each course must have one or more students
* Each course is taught by one or more lecturers
* Each lecturer teaches on one or more courses

Your entity relationship diagram should indicate:

* the name of each relationship
* the cardinality of each relationship

1. A relational database is used to store details in a school library. The following design rules are applied:

* Pupils can take out up to six books at a time
* There may be more than one copy of a particular book
* Books are usually written by only one author but sometimes that are written jointly by more than one author

Complete the entity relationship diagram below to model the library system. You should indicate:

* the cardinality of the relationship between each pair of entities
* the name of each relationship in the diagram



1. A chain of shops sells a range of mobile phones and accessories sourced from a variety of mobile phone companies. Customers place orders for phones and accessories with a shop which then supplies the customer from stock or orders the items from one if its suppliers.

After analysing the system, the following relationships have been identified:

* The organisation has many SHOPs in different towns and cities
* Each SHOP sells many PRODUCTs
* Each PRODUCT can be sold in many SHOPs
* PRODUCTs can be either a PHONE or an ACCESSORY
* PRODUCTs can be sourced from many SUPPLIERs
* SUPPLIERs supply many PRODUCTs
* CUSTOMERs place many different ORDERs over time
* Each ORDER can only come from one CUSTOMER
* One ORDER may be for many PRODUCTS
* Each PRODUCT may appear on different ORDERs

Complete the entity relationship diagram to model these entities. You should indicate the name and cardinality of the relationship between each pair of entities.

|  |  |
| --- | --- |
|  | CUSTOMER  ACCESSORY  ORDER  SUPPLIER  PRODUCT  SHOP  MOBILE PHONE |

1. A company is made up of departments. Each employee works in just one of those departments and each department has many employees working in it. A department can be working on many different projects at any time. A project might need many different departments to work on it. Each project requires many resources to complete it, such as computer time or a specialist contractor’s time. Each resource might be required on many different projects.

Use this description to identify the entities and relationships in the system. The first few have been started for you.

* Each EMPLOYEE works in only one DEPARTMENT
* Each DEPARTMENT employs many EMPLOYEEs

Draw an entity relationship diagram for the business. Remember to indicate the name and cardinality of the relationship between each pair of entities.

# Design: Entity Relationship Diagrams 2

1. A college runs many classes. Each class may be taught by several teachers, and a teacher may teach several classes. A particular class always uses the same room. Because classes may meet at different times or on different evenings, it is possible for different classes to use the same room.

Create an entity relationship diagram to represent this system.

1. A relational database system for a yacht club is to store details of skippers, boats and races. In the database design of the entities, the following business rules are applied:

* A boat takes part in many races
* Each boat belongs to one class (a type of boat); several boats may be in the same class
* Each boat has one skipper who only sails that one boat
* Several boats may participate in a race and each race involves many boats

Create an entity relationship diagram to represent this system.

1. An airline provides a chauffeur service to collect customers and drop them off at the airport. A relational database system is being developed to store details of each drop-off. Customers can book only one vehicle at each booking.

The following entities are used to store details of each drop-off:

* Customer (stores details of the customer who made an individual booking)
* Booking (stores details of each booking for the drop-off service)
* Flight (stores details of the flight associated with the customer’s booking)
* Airport (stores details of the airport that the flight takes off from)
* Vehicle (stores details of the vehicle assigned to the booking)

Create an entity relationship diagram to represent this system.

1. A company manufacturers electronic tills for use in shops. The company uses a relational database to store information about its sales. The entities in the database are:

* Shop (stores details of shops that have purchased tills)
* Till (stores details of tills that are produced by the company)
* Salesperson (stores details of the employee responsible for processing the order)
* Order (stores details of the order placed by the shop)
* Item (stores details of individual tills that make up an order)

Create an entity relationship diagram to represent this system.

# Design: Data Dictionary 1

## Part 1

A publishing company uses a relational database to store details about books and customer orders in four separate entities. Details of the entities and attributes used are shown below (primary keys have been underlined and foreign keys are indicated using an asterisk \*).

|  |  |  |  |
| --- | --- | --- | --- |
| **Customer Entity** | **Order Entity** | **Book Entity** | **Author Entity** |
| Customer Number  Customer Name  Customer Address  Customer Town  Customer Postcode | Order Reference  Order Date  ISBN \*  Quantity  Customer Number \* | ISBN  Book Title  Category  Price  Author Code \* | Author Code  Author Name  Author Address  Author Town  Author Postcode |

A sample customer order and sample book details are shown below.

|  |  |  |  |
| --- | --- | --- | --- |
| **Customer Order:**  Note: Category can be:   * Children * Crime * Historical * Large Print * Non-Fiction * Romance | **HYR847GB** | |  |
| Customer Number: | | 782 | |
| Customer Name: | | Inverclyde Books | |
| Customer Address: | | 52 High Street | |
| Customer Town: | | Gourock | |
| Customer Postcode: | | PA19 1UX | |

|  |  |  |
| --- | --- | --- |
| **Order Date** | **ISBN** | **Quantity** |
| 12/03/2014 | 0901714564X | 9 |
| 16/03/2013 | 7289192000S | 15 |
| 27/03/2013 | 0901714564X | 20 |
| 04/04/2013 | 3781972928N | 12 |
| 11/04/2013 | 1217292921B | 9 |

* Quantity must be at least 1 but can no more than 24
* The cheapest book available costs £0.99 and the dearest costs £38.95

|  |  |
| --- | --- |
| **Book** |  |
| **ISBN:** | 0901714564X |
| **Book Title:** | Weather Time |
| **Category:** | Non-Fiction |
| **Price:** | £15.99 |
| **Author Code:** | 87281 |
| **Author Name:** | Julie Adams |
| **Address:** | 15 West Street |
| **Town:** | Greenock |
| **Postcode:** | PA19 7XE |

Create a data dictionary to record details of each of the entities in this relational database system.

## Part 2

Now Rentals DVD club uses a relational database to store details of members, DVDs and rentals. A sample report produced by the system is shown below.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **MEMBER RECORD** | | | | | | | |
|  | |  |  | |  |  | |
| **Member Number:** | | 241 |  | | **Now Rentals** |  | |
| **Surname:** | Smith | |  |  | | |  |
| **Address:** | 23 Jones Road | |  |  |
| **Town** | Walforth | |  |  |
| **Phone:** | 07771234567 | |  | |  |  | |
| **Joined:** | 12/01/2010 | | **Priority Code**: | |  |  | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **DVD ID** | **DVD Title** | **Date Due Back** | **Late?** | **Certificate** | **Certificate Description** |
| 3236 | Wheelies | 11/05/14 | True | U | Universal |
| 6512 | Harrytron | 12/06/14 | False | PG | Parental guidance |
| 4419 | The Brainbox | 30/07/14 | False | PG | Parental guidance |

The database stores the rental details in four entities: MEMBER, RENTAL, DVD and CERTIFICATE.

The attributes in each of those entities are listed below (primary keys are underlined and foreign keys are indicated using an asterisk \*).

|  |  |  |  |
| --- | --- | --- | --- |
| **MEMBER** | **RENTAL** | **DVD** | **CERTIFICATE** |
| Member\_ID  Surname  Address  Town  Phone Number  Date Joined  Priority Code | Rental\_Number  Member\_ID \*  DVD\_ID \*  Date Due Back  Late | DVD\_ID  DVD\_Title  Certificate \* | Certificate  Description |

Certificate can be …

* U (Universal)
* PG (Parental Guidance)
* 12 (suitable for 12 years and over)
* 15 (Suitable only for 15 years and over)
* 18 (Suitable only for adults)
* Members can opt to pay an additional fee to become a Priority member
* All members who take this option are allocated a 4-digit Priority Code between 1111 and 9999

Create a data dictionary to record details of each of the entities in this relational database system.

# Design: Data Dictionary 2

## Part 1

The online registration form used by new customers of Trends is shown below. Identify two types of validation check used by Trends and explain the importance of each.

A screenshot of a login form

Description automatically generated

## Part 2

The Trends website provides a search facility that enables customers to search for a particular item.

A black background with orange and white text

Description automatically generated

Identify the type of validation check used with the search facility and explain its importance.

## Part 3

Admin staff can use software tools to alter the details of the online shopping basket for the Trends website.

A screenshot of a computer

Description automatically generated

Identify the type of validation check illustrated in this case and explain its importance.

## Part 4

MySQA is a service provided by the Scottish Qualification Authority. Candidates who sign up for MySQA will receive their exam results by email and text message. The Sign Up form is shown below.

Identify three types of validation check used in the Sign Up form and explain the importance of each.

A screenshot of a sign up form

Description automatically generated

## Part 5

The data entry form shown below is used by schools to enter estimate grade for candidates being presented for SQA qualifications.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| QualCode | | C716 | Level | N5 | |  |
| Title | Computing Science | | | | |
| SCN | | FirstName | LastName | | Estimate |
| 023456789 | | Jane | Jones | | 4 |
| 023456821 | | Liam  \\  Estimate must be grades 1 to 9 | Lamont | | 6 |
| 023457004 | | Siraaj | Zia | | 0 |
| 023457122 | | Mo | Yuan | |  |

Identify the type of validation check illustrated in this case and explain its importance.

## Part 6

SQA uses a relational database to store details of candidates, qualifications and results in three separate entities:

CANDIDATE (SCN, FirstName, LastName, DOB, Address, School)

QUALIFICATION (QualCode, Title, Level)

RESULT (ResultID, SCN\*, QualCode\*, Date, Estimate, FinalGrade)

Explain why it is not possible to create a record in the Result table before first creating a record in ***both*** the Candidate and Qualification tables.

## Part 7

MovieWatch allows digital TV viewers to rent streamed movies. Details of the rentals and members are stored in two separate linked tables. The field Member ID in the Rental table is foreign key that is used to establish a relationship between the Rental and Member tables.

Use the data in these tables to explain the importance of the foreign key in ensuring referential integrity.

### Rental

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rental Code** | **Member ID** | **Movie Code** | **Rental Date** | **Number Nights** | **Cost** |
| 172634 | 003 | M101 | 2015-07-01 | 3 | £5.99 |
| 172635 | 002 | M104 | 2015-07-01 | 2 | £2.99 |

### Member

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Member ID** | **First Name** | **Last Name** | **Town** | **Phone** |
| 001 | Marge | McGregor | Ayr | 07123456789 |
| 002 | Karl | Killen | Saltcoats | 07798765432 |
| 004 | Jess | Jeffries | Ardrossan | 07912345671 |

## Part 8

An online photo gallery stores details of photos displayed on the site in two separate linked tables: ALBUM and PHOTO.

Use the data in these tables to explain the importance of the foreign key in ensuring referential integrity.

### Album

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Album ID** | **Name** | **Category** | **Description** | **Number Photos** |
| 01 | BMW Cars | Cars | Photos of BMW cars | 25 |
| 02 | Glenrothes | Fife | Photos from around Glenrothes | 4 |
| 03 | Newark Castle | Castles | Photos of Scottish castles | 17 |

### Photo

|  |  |  |  |
| --- | --- | --- | --- |
| **Photo ID** | **Album ID** | **Description** | **Image** |
| 23 | 02 | Thirsty Hippos | hippos\_pmckay.jpg |
| 24 | 20 | Glenrothes Irises | irises\_mharris.jpg |

## Part 9

Trends is an online clothing store. Details of items for sale, purchases and customers are stored in three linked tables: ITEM, PURCHASE and CUSTOMER.

ITEM (Item ID, description, colour, size, price)

PURCHASE (OrderNo, CustomerID\*, ItemID\*, date, quantity)

CUSTOMER (CustomerID, name, address, email)

Before making a purchase, customer must first register their details. Once the details have been registered, they are stored in the Customer table.

A close-up of a button

Description automatically generated

Explain why it is necessary for customers to register their details ***before*** making a purchase.

# Design: Queries that use another query

## Part 1

The Countries database stores details of countries and cities in two separate tables called Country and City. The structure of the tables is shown below.



Design SELECT queries to perform each of the following tasks. Each design should indicate:

* any field(s) or computed values required
* the table(s) or query(queries) needed to provide the details required
* any search criteria to be applied
* what grouping is needed (if appropriate)
* the field(s) used to sort the data and the type(s) of sort required

1. Search the database to display the name, capital city and total population of the country with the largest total population.

### Query 1(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 1(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the name, country and population of any city that has a population which is at least 5,000,000 more than the average population of all the cities.

### Query 2(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 2(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the name of any city that is further north than Reykjavik (the latitude of these cities is greater than the latitude of Reykjavik). The query should show the name of each relevant city, its latitude and country name. The city that is furthest north should be listed first.

### Query 3(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 3(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the name and population of any city in the United Kingdom that has a population which is more than the average population of all the cities in Bolivia. Arrange these details from smallest to largest population.

### Query 4(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 4(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the total number of countries with an area less than 1% of that of the country with the largest area.

### Query 5(1)

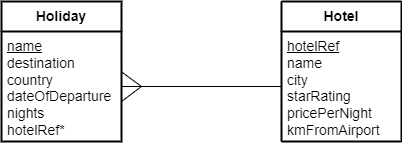
|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 5(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

# Design: Queries that use another query

The SimpleBreaks database stores details of countries and cities in two separate tables called Holiday and Hotel. The structure of the tables is shown below.



Design SELECT queries to perform each of the following tasks. Each design should indicate:

* any field(s) or computed values required
* the table(s) or query(queries) needed to provide the details required
* any search criteria to be applied
* what grouping is needed (if appropriate)
* the field(s) used to sort the data and the type(s) of sort required

1. Search the database to display the name, destination, country and distance from the airport of the hotel that is furthest from the airport.

### Query 1(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 1(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the name and star rating of any hotel with a rating that is poorer that the average star rating of all the holidays that have the word ‘Break’ or ‘Package’ in their title. The hotel with the highest star rating should be listed first; hotels with the same star rating should be listed in alphabetical order of hotel name.

### Query 2(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 2(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the name, city and price per night of any hotel which is dearer that the dearest hotel in Edinburgh. List the hotel details with the dearest hotel first; hotels with the same price should be listed in alphabetical order of city.

### Query 3(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 3(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the number of holidays that have the same star rating as that of the ‘Der Wald’ hotel.

### Query 4(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 4(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the title, city and distance from the airport of any holiday to Lisbon that is closer to the airport than the average distance from the airport of all the hotels in Spain.

### Query 5(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 5(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

1. Search the database to display the title, departure date and duration of any holiday that has the same duration as the longest holiday to a city with the letter ‘o’ as the second character of the city name. Arrange these details so that the most recent holiday is listed first.

### Query 6(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

### Query 6(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) |  |
| Table(s) and query(queries) |  |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

# Implementation: SQL Wildcards

1. Which of the following strings satisfies the condition 'T\_'? (There can be more than one answer)
   1. TW
   2. TWITTER
   3. TAIWAN
   4. To
2. Which of the following strings satisfies the condition 'A\_B%'? (There can be more than one answer)
   1. AKKB
   2. AKBKKK
   3. ABKK
   4. ABBB
3. Which of the following strings satisfies the condition 'Z%K\_R'? (There can be more than one answer)
   1. ZKRR
   2. ZYKR
   3. ZRKKJ
   4. ZABCDEFKGR
4. Write the conditions used to list productNames that:
   1. start with the letter 'S'.
   2. end with the letters 'on'.
   3. contain the pattern 'ill'.
   4. contain the letters 'o' and 'n' (in that order) with exactly one letter between them.
   5. contain the patterns 'aw' and 'ing' separated by exactly two letters.
   6. start with the pattern 'St' which is followed by any character which in turn is followed by the letter 'a' and ends with the letter 's'.

A table called Sales is shown below.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **firstName** | **lastName** | **birthDate** | **joinDate** | **totalSales** |
| Sophie | Lee | 05/01/1960 | 05/04/2015 | 500 |
| Richard | Brown | 07/01/1975 | 05/04/2015 | 200 |
| Jamal | Santo | 08/10/1983 | 09/04/2015 | 350 |
| Casey | Healy | 20/09/1996 | 09/04/2015 | 80 |
| Jill | Wilkes | 20/11/1979 | 15/04/2015 | 210 |

1. Which of the following SQL statement is valid? (There can be more than one answer)
   1. SELECT firstName, lastName

FROM Sales

WHERE firstName LIKE 'A%' lastName LIKE 'W%';

* 1. SELECT firstName, lastName

FROM Sales

WHERE firstName LIKE 'J%' AND lastName LIKE 'W%';

* 1. SELECT firstName, lastName

FROM Sales

firstName LIKE 'J%' AND lastName LIKE 'W%';

* 1. SELECT firstName, lastName

FROM Sales

WHERE firstName LIKE 'J%', lastName LIKE 'W%'

1. How many records will be returned by the following query? (Assume the database is not case sensitive)

SELECT \*

FROM Sales

WHERE lastName LIKE '%l\_e%';

1. How many records will be returned by the following query? (Assume the database is not case sensitive)

SELECT \*

FROM Sales

WHERE firstName LIKE '%i%'

OR lastName LIKE '%e%';

# Implementation: SQL Wildcards

Open the file Clydeview.db file. The Member entity is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| membershipType | PK | Text | 6 | Y |  |
| forename |  | Text | 20 | Y |  |
| surname |  | Text | 20 | Y |  |
| address |  | Text | 30 |  |  |
| town |  | Text | 20 |  |  |
| postcode |  | Text | 8 |  |  |
| dateOfBirth |  | Date |  | Y |  |
| monthOfRenewal |  | Text | 9 |  |  |
| typeOfMembership |  | Text | 7 | Y | Restricted choice: Adult, Child, Senior, Student |

Create SQL queries to display the required details.

1. List the full name, postcode and the renewal month of all members with a surname that begins with the letter 'D'.
2. List the membership number, surname and full postal address of all members who have a surname that contains the pattern 'oo'. List these members in alphabetical order of surname.
3. List the first name and home town of members with a first name that contains the letter 'o' and who live in a town that starts with the letter 'B' and ends with the letter 'n'.
4. List the full name of any members who have a surname that has exactly 4 letters. starting with the letter 'L'.
5. List the membership number, town and postcode of all members with a postcode that contains the letter 'a' and the digit '2' separated by exactly 1 character.
6. List the full name and type of membership of all members who have a surname that contains the letters 'i' and 'e' separated by exactly two characters.
7. List the membership number, date of birth and the type of membership for all adult members who were born in the month of October.

# Implementation: SQL Wildcards

Open the Clydeview.db file. The Plant entity is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| category |  | Text | 10 |  |  |
| plantName |  | Text | 20 | Y |  |
| variety |  | Text | 20 | Y |  |
| code |  | Text | 3 | Y |  |
| referenceID | PK | Text | 3 | Y |  |
| unit |  | Number |  |  |  |
| price |  | Number |  |  |  |
| height |  | Text | 1 |  | Restricted choice: S, M, T |

Create SQL queries to display the required details.

1. List the name and variety of any plant that has the letter 'x' somewhere in the name of its variety. These details should be listed in alphabetical order of plant name.
2. List the category, name, plant code and price of any plant with a plant code that has exactly 2 characters.
3. List the name of any plant, together with its plant code and height, with a code that contains the letter 'P' and a plant name that contains the letters 'a' and 't' separated by exactly one character.
4. List the name, reference ID and price of all plants with an 'a' in the name which also end with the letter 'a'. The dearest plant should be listed first; plants with the same price should be listed in alphabetical order of referenceID.
5. List the plant code, referenceID and category of any plant that has a '3' in the middle of its referenceID (each referenceID has 3 characters) and the letter 'r' anywhere in its plant code. These plants should be listed in alphabetical order of category.
6. List the plant name, unit size and price of any plant with the letters 'a' and 'n' (separated by exactly 2 other letters) in its name. Arrange these plant details so that the largest unit size is listed first; plants with the same unit size should be arranged so that the cheapest plant is listed first.
7. List the referenceID, plant name, variety and height of any plant with the letter 'e' as the second letter of its plant name. Only plants the belong to a variety that starts with the letter 'C' and ends with the letter 'e' should be listed.

# Implementation: Calculations and Computed Fields

Open the file Clydeview.db file. Use the tables provided in this database to generate each of the following computed fields.

The table called Table1 stores the test results of several pupils.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| pupilID | PK | Number |  | Y |  |
| forename |  | Text | 12 | Y |  |
| surname |  | Text | 12 | Y |  |
| test1 |  | Number |  | Y | Range: >= 0 and <=10 |
| test2 |  | Number |  | Y | Range: >= 0 and <=10 |
| test3 |  | Number |  | Y | Range: >= 0 and <=10 |
| test4 |  | Number |  | Y | Range: >= 0 and <=10 |

1. Use a SQL query to list the full name of each pupil together with the pupil’s total mark for all 4 tests.
2. Use a SQL query to list the first name of each pupil with their 4 test marks and the average mark for the 4 tests. These details should be arranged so that the pupil with the highest average mark is listed first.

The table called Table2 stores the hours worked and hourly rate of some staff members.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| staffID | PK | Number |  | Y |  |
| forename |  | Text | 12 | Y |  |
| surname |  | Text | 12 | Y |  |
| hourlyRate |  | Number |  | Y | Range: >= 0 |
| hoursWorked |  | Number |  | Y | Range: >= 0 |

1. Use a SQL query to list the full name of each member of staff together with their total wage.
2. Use a SQL query to list the full name of all members of staff whose wage is over £200.

The table called Table3 stores the details of students and the marks they achieved in monthly tests (each test was out of 16 marks).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| studentID | PK | Number |  | Y |  |
| forename |  | Text | 50 | Y |  |
| surname |  | Text | 50 | Y |  |
| test1 |  | Number |  | Y | Range: >= 0 and <=20 |
| test2 |  | Number |  | Y | Range: >= 0 and <=20 |
| test3 |  | Number |  | Y | Range: >= 0 and <=20 |
| test4 |  | Number |  | Y | Range: >= 0 and <=20 |
| Test5 |  | Number |  | Y | Range: >= 0 and <=20 |

1. Use a SQL query to list each test mark as a percentage.

**NB**: To generate the percentages, divide each mark by 16 and multiply by 100.

1. Use a SQL query to list the full name of each student together with the student’s total mark as a percentage. These details should be listed from smallest percentage to largest; students with the same percentage should be listed in alphabetical order of surname.

The table called Table4 stores details of items for sale in the school tuckshop.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| product | PK | Number |  | Y |  |
| productName |  | Text | 12 | Y |  |
| buyingPrice |  | Number | 50 | Y | Range: >= 0 |
| sellingPrice |  | Number |  | Y | Range: >= 0 |

1. Use a SQL query to list the name of each item, its buying price, selling price and the profit or loss for that item.

**NB**: Profit = Sale price – Cost Price

1. Use a SQL query to list the name of each loss-making item with the amount of its loss. The items should be arranged so that the item with the smallest loss is listed first.

The table called Table5 stores details of products and their prices.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| productName |  | Text | 20 | Y |  |
| productID | PK | Text | 50 | Y |  |
| priceUK |  | Number |  | Y | Range: >= 0.00 |

1. Use a SQL query to list the products name, UK price and the equivalent prices in Euros.

Notes: (1) £1 buys €1.13

(2) use the round() function to round the results to 2 decimal places

1. Use a SQL query to list the ID of any products that cost most than $40 dollars. The query should show the UK prices as well as the equivalent prices in US Dollars. The products should be listed with the cheapest displayed first; products that cost the same should be listed with the highest productID shown first.

Note: £1 buys $1.39

The table called Table6 stores details of fish sales for a fish wholesaler.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| fishType | PK | Text | 12 | Y |  |
| pricePerKilo |  | Number |  | Y | Range: >= 0.00 |
| numberOfKilos |  | Number |  | Y | Range: >= 0.0 |

1. Use a SQL query to list the full details of each type of fish together with the total cost of each sale.
2. Use a SQL query to list the name of type of fish that has sales between £20 and £50. The results should be listed with the dearest sales displayed first.

# Implementation: Calculations and Computed Fields

Open the file Clydeview.db file. Use SQL queries to amend the required data in the tables of this database.

The table called Table1 stores the test results of several pupils.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| pupilID | PK | Number |  | Y |  |
| forename |  | Text | 12 | Y |  |
| surname |  | Text | 12 | Y |  |
| test1 |  | Number |  | Y | Range: >= 0 and <=10 |
| test2 |  | Number |  | Y | Range: >= 0 and <=10 |
| test3 |  | Number |  | Y | Range: >= 0 and <=10 |
| test4 |  | Number |  | Y | Range: >= 0 and <=10 |

All of the test 3 results have been totalled incorrectly.

1. Use a SQL query to add 1 to each of the test 3 results.

The table called Table2 stores the hours worked and hourly rate of some staff members.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| staffID | PK | Number |  | Y |  |
| forename |  | Text | 12 | Y |  |
| surname |  | Text | 12 | Y |  |
| hourlyRate |  | Number |  | Y | Range: >= 0 |
| hoursWorked |  | Number |  | Y | Range: >= 0 |

Members of staff who earn less than £7 per hour are due to receive a 10% pay rise.

1. Use a SQL query to amend the required details of the relevant records.

The table called Table3 stores the details of students and the marks they achieved in monthly tests.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| studentID | PK | Number |  | Y |  |
| forename |  | Text | 50 | Y |  |
| surname |  | Text | 50 | Y |  |
| test1 |  | Number |  | Y | Range: >= 0 and <=20 |
| test2 |  | Number |  | Y | Range: >= 0 and <=20 |
| test3 |  | Number |  | Y | Range: >= 0 and <=20 |
| test4 |  | Number |  | Y | Range: >= 0 and <=20 |
| Test5 |  | Number |  | Y | Range: >= 0 and <=20 |

Both students whose surname starts with the letter 'J' have found errors in their test results for test 2 and test 4.

1. Use a single SQL query to increase each of their test 2 results by 2 marks and each of their test 4 results by 1 mark.

The table called Table4 stores details of items for sale in the school tuckshop. All prices shown are in pence.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| productID | PK | Number |  | Y |  |
| productName |  | Text | 12 | Y |  |
| buyingPrice |  | Number | 50 | Y | Range: >= 0 |
| sellingPrice |  | Number |  | Y | Range: >= 0 |

The buying price of the product with a product name ending in 'e', has been reduced by 2 pence. At the same time, the selling price is being updated to make a profit of 40% of the revised buying price.

1. Use two SQL queries to alter the buying price and selling price of the relevant product.

The table called Table5 stores details of products and their UK prices in pounds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| productName |  | Text | 20 | Y |  |
| productID | PK | Text | 50 | Y |  |
| priceUK |  | Number |  | Y | Range: >= 0.00 |

Some of the product names in this table contain the letter 'a' followed by another letter which in turn, is followed by a space. The price of these products is to be increased by 12%.

1. Use a SQL query to amend the relevant records.

The table called Table6 stores details of fish sales for a fish wholesaler. Prices are in pounds.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| fishType | PK | Text | 12 | Y |  |
| pricePerKilo |  | Number |  | Y | Range: >= 0.00 |
| numberOfKilos |  | Number |  | Y | Range: >= 0.0 |

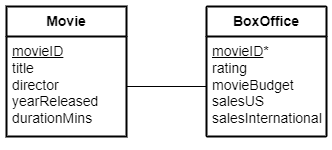
Every week, the wholesaler reduces the price of some of the fish.

This week, the second letter of each type of fish reduced is the letter 'o' and each type of reduced fish has at least 4 letters.

1. Use a single SQL query to reduce the price of these fish by 10% but at the same time, to double the number of kilos of the fish sold.

# Implementation: Computed Fields and Aliases

Open the file Clydeview.db file. The tables called Movie and BoxOffice are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for each computed field.

1. List the title and the profit made of any movie that made a profit with its international sales. Display the movie with the largest profit first.

Note: profit = international sales – movie budget

1. List the name of the director of each movie with value of the combined US and international sales. Display the details in alphabetical order of director; movies by the same director should be listed in order of total income, from least to most.
2. List the name of each movie with the year it was released and its rating. The international sales for each movie should be shown in millions. Only movies with international sales over 150 million that have a rating of at least 8 should be included in the list.
3. List the title of each movie together with its duration. The duration should also be shown in separate hours and minutes. For example:

|  |  |  |
| --- | --- | --- |
| **title** | **Hours** | **Minutes** |
| Cars | 1 | 57 |

The movies should be listed in decreasing order of hours; movies that last for the same number of hours should be listed in decreasing order of minutes.

1. List the title of each movie, the name of each director and its rating as a percentage (the maximum rating is 10). The movie with the highest rating should be listed first; movies with the same rating should be listed in alphabetical order of title.

# Implementation: Computed Fields and Aliases

Open the file Clydeview.db file. The tables called Country and City are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for each computed field. Example output is shown for each answer.

1. List the name of each city in Ethiopia together with the population of the city which should be displayed in thousands.

|  |  |
| --- | --- |
| **City** | **?** |
| Addis Ababa | 2316 |
| Dire Dawa | 194 |
| Gonder | 166 |
| Nazret | 147 |
| … | … |

1. List the name of each country together with its country code and population density (this tells us how many people live in each square kilometre of the country). The only countries listed should be those with a country code that starts with the letter ‘M’. These should be listed from highest density to lowest.

|  |  |  |
| --- | --- | --- |
| **Country** | **code** | **?** |
| Monaco | MC | 31719 |
| Macau | MACX | 31052 |
| Malta | M | 1173 |
| Maldives | MV | 902 |
| … | … | … |

Note: population density is calculated by dividing the total population of the country by its area

1. List the name of the capital cities that have a population of more than 2,000,000 together with the area of their countries in square miles. The only cities shown should be those in countries that have an area over 500,000 square miles. These details should be listed with the smallest area in square miles first.

|  |  |
| --- | --- |
| **capital** | **?** |
| Lima | 501235.8 |
| Tehran | 642720.0 |
| Jakarta | 748581.6 |
| Mexico City | 769294.5 |
| … | … |

Note:

the area of each country is currently in square kilometres. To convert to square miles, multiply by 0.39

1. List each capital city that is east of London with the name of its country and the time difference in hours between it and London. This first countries listed should be those that have the greatest time difference from the UK; countries with the same time difference should be listed alphabetically.

|  |  |  |
| --- | --- | --- |
| **capital** | **Country** | **?** |
| Suva | Fiji | 12 |
| Funafuti | Tuvalu | 12 |
| Wellington | New Zealand | 12 |
| Tarawa | Kiribati | 12 |
| … | … | … |

Notes:

(1) the UK has a longitude of 0 and all countries east of the UK have a longitude > 0

(2) each degree of longitude is equivalent to 4 minutes of time

(3) generate the time difference in hours by dividing the longitude by 15

(4) display time difference to the nearest whole hour

1. List the name of each European capital city north of London with its latitudinal distance north of London. Cities further away should be shown first.

|  |  |
| --- | --- |
| **capital** | **?** |
| Helsinki | 979 |
| Oslo | 953 |
| Stockholm | 887 |
| Tallinn | 881 |
| … | … |

Notes:

(1) European cities have a longitude between -6·5 and 30 degrees east

(2) cities north of London have a latitude between 51·5 and 61 degrees north

(3) each degree of difference in latitude if equivalent to 113 kilometres

# Implementation: Computed Fields and Aliases

Open the file Clydeview.db file. The tables called Holiday and Hotel are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for each computed field.

1. Due to the opening of a new bypass, the travel distance between each hotel in the city of Madrid and the airport has been reduced by 2.8 km.

A query is used to list the title of all holidays affected together with the name of the hotels used, the original distance from the airport and the updated distance due to the new bypass.

1. Rome has introduced a new city tax that costs £7.25 per night.

A query is used to list the name of each hotel in Rome, the cost per night together with the tax inclusive price per night and the equivalent tax inclusive cost in Euros. The hotels should be listed from dearest tax inclusive cost to least (£1 is equivalent to €1.13).

1. Due to their popularity, the duration of all holidays with ‘budget’ in their title has been increased by 1 night and the price per night of the hotels used for these holidays has been increased by 10%

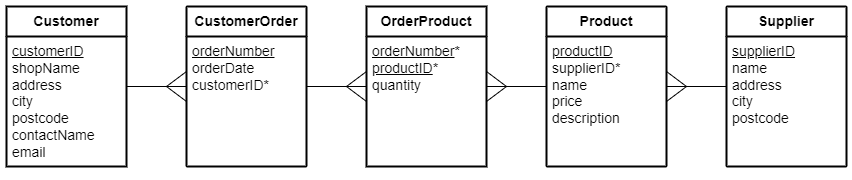
A query is used to list the title of each affected holiday and the name of the hotels used, the duration of each holiday and the original price per night of the hotels used. The query should also show the updated duration and price per night.

1. A query is used to list the title of all holidays to Spain, the name of the hotels used, the duration of each holiday and the price per night. The query should show the total cost of each holiday. The holidays should be listed from dearest to cheapest; holidays that cost the same should be listed in alphabetical order of holiday title.
2. A query is used to list the details of all holidays to hotels with at least 4 stars that have a total cost less than £200. The holiday details should include the title of each holiday, the name of each hotel, the country and city of each destination, the star rating and the total cost of the holiday.

# Implementation: Computed Fields and Aliases

Open the file Clydeview.db file.

The tables called Customer, CustomerOrder, OrderProduct, Product, and Supplier are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for each computed field.

1. List the name of each product included in order 20006 with its price and the quantity ordered. The product number and total cost of each item should be shown, with the most expensive total cost shown first.
2. List all of the products supplied by the supplier called ‘Fun and Games’. The listing should show the name of each product together with its price and discounted price (Fun and Games plans to offer a seasonal discount of 10% on all of its products).
3. Find all items supplied by the 'Dolls House' in order 20007. Display the productID, name, quantity ordered, and total cost per item. Display with the cost increasing. For items with the same cost, display the names alphabetically.
4. 'The Toy Shop' wants to increase the number of each type of bean bag in its order by 25. Create a query to list the product description, price, quantity ordered, total cost, and the updated quantity and updated total cost of all bean bags in the order.

# Implementation: Aggregate Functions

Open the file Clydeview.db file. The table called Employee is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| employeeID | PK | Number |  | Y |  |
| jobTitle |  | Text | 8 | Y |  |
| name |  | Text | 20 |  |  |
| building |  | Text | 2 |  |  |
| yearsEmployed |  | Number |  | Y | Range: >= 0 |

Use SQL queries to display each set of required details, with appropriate aliases.

1. Display the number of employees.
2. List the longest and shortest service of all employees of the company.
3. List the average years of service of all the employees.
4. List the job titles and the average years of service for each of those job titles.
5. List the name of each building and the number of employees who work in it.
6. List the name of each building and the total years service of all employees who work in that building.
7. Employees in each building are due a bonus of 100 times the shortest years of service of any employee who works in that building. List the name of each building and the bonus due.
8. All of the employees with the same job title are due to receive extra holiday days equivalent to half of the longest years of service for that job title. List each job title together with the extra holiday days due for those employees.
9. List the name of each building with the job titles of the employees who work in the building and the number of employees with these job titles.

# Implementation: Aggregate Functions

Open the file Clydeview.db file. The table called Plant is shown below.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Name** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| category |  | Text | 10 | Y |  |
| name |  | Text | 20 | Y |  |
| variety |  | Text | 20 | Y |  |
| code |  | Text | 3 | Y |  |
| referenceID | Y | Text | 3 | Y |  |
| unit |  | Number |  | Y | Range: >=0 |
| price |  | Number |  | Y | Range: >=0.00 |
| height |  | Text | 1 | Y | Restricted choice: S, M, T |

Use SQL queries to display each set of required details, with appropriate aliases.

1. Display the name of each category and the average price of plants in that category. Round the average price to 2 decimal places.
2. Display the plant codes together with the number of plants that share the same code. The code with the most plants should be listed first; codes that have the same number of plants must be listed in alphabetical order of code.
3. Display the list of plant heights and the range of the prices for each plant height (the range of prices is the difference between the dearest and the cheapest plant).
4. Display the number of plants that have the letter 'p' in their name together with the total cost of those plants.
5. Display the largest and average unit size, of all plants that have a referenceID beginning with the letter 'B'. Round the average size to 1 decimal place.
6. Display the list of unit sizes together with the number of plants and the cheapest plant sold in those units. The most populous unit size should be listed first; where the number of plants is the same, the plants should be listed from smallest to largest unit size.
7. Display the list of categories and heights together with the average price of the plants in each sub-category of height.

# Implementation: Aggregate Functions

Open the file Clydeview.db file. The tables called Country and City are shown below.



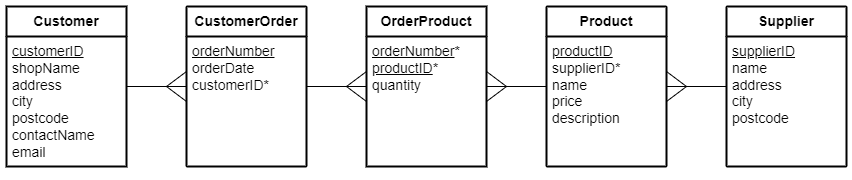
Use SQL queries to display each set of required details. Aliases should be used.

1. List the largest, smallest and average area of all the countries (the average should be rounded to 1 decimal place).
2. List the number of countries that have an area over one million kilometres squared.
3. List the combined area of all countries with a name that starts with ‘D’.
4. List the combined population of all the cities located in Germany.
5. List each country code together with the total population of the cities in that country. The only country codes listed should be those that have exactly two letters ending with the letter 'A'.
6. List the total population of the capital cities of any country that has the pattern 'AM' in its country name.
7. List the name of each country that has at least two words in its name together with the combined population of the cities in each country. The only countries that should be listed are those with a total country population which is over 11 million.
8. List the name of each country and the number of cities in that country (the countries that are listed should be those that contain the letter 'i' somewhere in the name of the country which also ends with the letter 'a'). This list of countries should be displayed in descending order of number of cities; where this is equal, the results should be displayed in alphabetical order of country name.

# Implementation: Aggregate Functions

Open the file Clydeview.db file.

The tables called Customer, CustomerOrder, OrderProduct, Product, and Supplier are shown below.



Use SQL queries to display each set of required details.

1. List the name of each supplier together with the average price of the products that they supply (round the average to 2 decimal places).
2. List the name of each shop together with the largest and the average number of products ordered by the shop (round the average to 1 decimal place).
3. List the name of each shop and the number of orders that the shop has placed. The shop with the largest number of orders should be listed first; shops with the same number of orders should be listed alphabetically.
4. List each order number together with the number of different products ordered in each order.
5. List the name of each supplier together with the total value of the products that they supply. These details should be listed from largest to smallest value.
6. List the name of each product that has been ordered with the combined total quantity of the product that has been ordered. The products should be listed from largest combined total to smallest; product with the same combined total should be listed alphabetically by product name.
7. List the name of each shop and the total number of bears that have been ordered by the shop. The shop that has ordered the smallest number of bears should be listed first.
8. List the name of each shop, the order number of any orders placed by the shop, the date and the total cost of those orders. Arrange these details in alphabetical order of shop name; where a shop has placed two or more orders, the most recent order should be listed first.

# Implementation: Problem Solving

## Background: Go Go Gadgets

GoGoGadgets.com is a website that specialises in quirky and unusual gadgets that are available for purchase through its on-line catalogue. Before customers can make a purchase from the website, they must first register with GoGoGadgets. Customer details are stored by GoGoGadgets in a **Customer** table. On registering, customers are notified by e-mail of their unique customer ID which must be used in all future purchases.

GoGoGadgets’ product range is stored in a table called **Item** and these items can be browsed through an on-line catalogue. Each item is categorised as one of the following: Boys Toys, Girls Gizmos, Office Distractions, Personal Grooming and Computer Accessories. All items cost less than £50.

When placing an order, customers must first log-in and create a new order – this order is added to a **CustOrder** table. Customers can then add items from the catalogue to an order. More than one item can be added to a given order. Each individual item ordered is stored in a separate table called **OrdeItem**.

A relational database has been set up to store details of customers, items, orders and order lines. The resulting data model is shown below where primary keys are shown in **bold and underlined**, while asterisks (\*) indicate foreign keys.

Customer (**customerID**, forename, surname, address1, address2, address3, postcode, customerEmail)

Item (**itemID**, description, category, price)

CustOrder (**orderNo**, customerID\*, orderDate, orderDispatched)

OrderItem (**orderNo**\***, itemID**\*, quantity)

## Part 1: Entity Relationship Diagram

Create the entity-relationship diagram for the GoGoGadgets system.

## Part 2: Data Dictionary

A data dictionary has been produced prior to implementation in a relational database. Part of this is shown below.

### Entity Customer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| customerID | PK | Number |  | Y | Range >= 1 and <= 9999 |
| forename |  | Text | 10 | Y |  |
| Surname |  | Text | 15 | Y |  |
| address1 |  | Text | 20 | Y |  |
| address2 |  | Text | 20 | N |  |
| address3 |  | Text | 20 | Y |  |
| postcode |  | Text | 8 | Y |  |
| customerEmail |  | Text | 25 | **A** |  |

### Entity Item

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| itemID | PK | Number |  | Y |  |
| description |  | Text | 50 | Y |  |
| category |  | Text | 20 | Y | **B** |
| price |  | Number |  | Y | Range < 50 |

### Entity CustOrder

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| orderNo | **C** | Number |  | Y |  |
| customerID | FK | Number |  | Y | **D** |
| orderDate |  | Date |  | Y |  |
| orderDispatched |  | Date |  | N |  |

### Entity OrderItem

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| orderNo | **E** | Number |  | Y | Existing orderNo from Order entity |
| itemID | PK/FK | **F** |  | Y | Existing itemID from Item entity |
| quantity |  | Number |  | Y |  |

1. For text attributes, the number in brackets indicates the size of the attribute (number of characters).
2. Complete the data dictionary by writing down the missing entries for A – F.

*\*\* Check your solution with your teacher before continuing. Once your work has been checked, your teacher will provide you with a completed data dictionary and version 1 of the GoGoGadgets database.*

## Part 3: Tables and Relationships

Open the database file called GoGoGadgets. You **MUST** complete the following two tasks in the sequence indicated.

1. Edit the structure of each table to ensure it matches the data dictionary provided. You should pay particular attention to:

* primary and foreign keys
* data type
* data size
* required
* validation

1. Set up the relationships between all the tables and enforce referential integrity between the tables.

*\*\* Check your solution with your teacher before continuing. Once you work has been checked, your teacher will provide you with the populated GoGoGadgets database.*

## Part 4: Queries and Calculations

### Searching and Sorting

1. A list of all of the items that belong to the category ‘Boys Toys’. The list should include all item details.
2. A list of all items in each order. The list should show the order number, item description and price and should be in ascending order of order number.
3. A list of all the full names of all customers with a surname containing the letters 'em' along with the dates of their orders. The list should be in alphabetical order of surname; when two or more surnames are the same, they should be listed in alphabetical order of first name.

### Computed Fields

1. A list showing the order number, order date, item descriptions, quantities ordered and prices. A calculated field should be used to work out the total cost of each item (quantity x price in each order). The details should be listed in order of date, from oldest to most recent.
2. The company has decided to apply a 5% discount to any items whenever the minimum order quantity is 4. Create a list showing the relevant order numbers, the description of qualifying item, the quantity of the item ordered, the original price, the value of the discount and the discounted price.

### Grouping Data and Aggregate Functions

1. A list showing details of all orders placed by Mari Singer. The list should show the order number, order date, description, quantity ordered, price and the total price of each item in the order. The list should be displayed with details of the most recent order first.
2. A list showing each category with the number of items in each category. Details of the largest category should be listed first.
3. A list showing each order number, order date and the total cost of the order for all orders placed in January 2008. The details of the oldest order should be listed first.

### Additional Queries

A list showing the full name of all customers who have an email address provided by MobileLife.

A list showing the category, the number of orders placed and the total quantity of items in the 'Office Distractions' category that have been ordered.

A list showing the name of each category and the average price of items that belong to that category.

A list showing each order number with the customer’s full name and the number of items ordered. The only orders shown should be those placed by customers whose surname contains the letters 'i' and 'g' separated by one other letter (the letter 'g' is not the last letter).

A list showing the customerID and postcode and the number of orders placed by the customer in 2008. Arrange the list so that the customer who placed the most orders is listed first; customers who placed the same number of orders should be listed alphabetically by postcode.

The company is offering a 5% discount on all orders placed in December 2007.

Produce a list to show each order number and order date, the order totals before discount, the value of each order’s 5% discount and the overall totals after discount. Orders should be listed with the oldest order first. Where two or more orders are placed on the same day, they should be sorted by OrderNo in ascending order.

## Completed Data Dictionary

*\*\* Only to be issued after completion and submission of Task 2*

### Entity: Customer

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| customerID | PK | Number |  | Y | Range >= 1 and <= 9999 |
| forename |  | Text | 10 | Y |  |
| Surname |  | Text | 15 | Y |  |
| address1 |  | Text | 20 | Y |  |
| address2 |  | Text | 20 | N |  |
| address3 |  | Text | 20 | Y |  |
| postcode |  | Text | 8 | Y |  |
| customerEmail |  | Text | 25 | **Y** |  |

### Entity: Item

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| itemID | PK | Number |  | Y |  |
| description |  | Text | 50 | Y |  |
| category |  | Text | 20 | Y | **Restricted Choice (one of Boys Toys, Girls Gizmos, Personal Grooming, Office Distractions or Computer Accessories)** |
| price |  | Number |  | Y | Range < 50 |

### Entity: CustOrder

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| orderNo | **PK** | Number |  | Y |  |
| customerID | FK | Number |  | Y | **Existing customerID from Customer entity** |
| orderDate |  | Date |  | Y |  |
| orderDispatched |  | Date |  | N |  |

### Entity: OrderLine

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| orderNo | **PK/FK** | Number |  | Y | Existing orderNo from Order entity |
| itemID | PK/FK | **Number** |  | Y | Existing itemID from Item entity |
| quantity |  | Number |  | Y |  |

# Implementation: Problem Solving

## Background: Northbank College Department of Informatics

Northbank College Department of Informatics runs a year-long course called Relational Databases.

The course is made up of seven topics each of which has a separate **Assessment**. Each assessment can be one of several types e.g. Multiple Choice or Short Answer and lasts for a different length of time. All assessment marks are recorded as percentages and each assessment may also have a different pass mark. Assessments are available on several dates and may be taken on any of those dates. The duration of each assessment is recorded in minutes.

Every **Student** attending the College will have a unique Student No and must be a member of a **Tutor Group**. Tutor Groups meet at various times and places throughout the week.

The **Result** of every assessment a student takes is recorded as a percentage along with the date of sitting the assessment. Students can only sit an assessment once.

The Department of Informatics uses a relational database to manage this information.

The data model is shown below. (Primary keys are shown in **bold and underlined**, while asterisks (\*) indicate foreign keys.)

### Entities

Student (**studentNo**, forename, surname, address, town, postcode, tgCode\*)

Assessment (**assessmentCode**, title, assessmentType, duration, writer, passMark)

TutorGroup (**tgCode**, tutor, room, day, time)

Result (**studentNo\***, **assessmentCode**\*, mark, assessmentDate)

## Part 1: Data Modelling

Create the entity-relationship diagram for the Northbank College system.

## Part 2: Data Dictionary

A data dictionary has been produced prior to implementation in a relational database. Part of this is shown below.

### Entity Student

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| studentNo | PK | Number |  | Y | Range >= 100000 and <= 199999 |
| forename |  | Text | 20 | Y |  |
| surname |  | Text | 30 | Y |  |
| address |  | Text | 30 | Y |  |
| town |  | Text | 15 | Y |  |
| postcode |  | Text | 8 | Y |  |
| tgCode | FK | Text | 5 | Y | Existing tgCode from TutorGroup table |

### Entity TutorGroup

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| tgCode | PK | Text | 5 | Y |  |
| tutor |  | Text | 15 | Y |  |
| room |  | Text | 5 | Y |  |
| day |  | Text | 9 | Y | Restricted choice (one of Monday, Tuesday, Wednesday, Thursday or Friday) |
| time |  | Time |  | Y |  |

### Entity Assessment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| assessmentCode | PK | Text | 6 | Y |  |
| title |  | Text | 30 | Y |  |
| assessmentType |  | Text | 30 | Y | Restricted choice (one of Multiple Choice, Short Answer, Extended Response, or Practical) |
| duration |  | Number |  | Y | Range >= 15 and <= 150 |
| writer |  | Text | 15 | Y |  |
| passMark |  | Number |  | Y | Range >= 0 and <= 100 |

### Entity Result

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute** | **Key** | **Type** | **Size** | **Req'd** | **Validation** |
| studentNo |  |  |  | Y |  |
| assessmentCode |  |  |  | Y |  |
| mark |  |  |  | Y |  |
| assessmentDate |  | Date |  | Y | Range >= 2013-08-15 and <= 2014-05-12 |

Complete the data dictionary for the Result table.

*\*\* Check your solution with your teacher before continuing. Once your work has been checked, your teacher will provide you with a completed data dictionary and version 1 of the Northbank College database.*

## Part 3: Tables and Relationships

Open the database file called NorthbankCollege. You **MUST** complete the following two tasks in the sequence indicated.

1. Database tables have been created for the Student, Assessment and TutorGroup tables. Edit the structure of each table to ensure it matches the data dictionary provided. You should pay particular attention to:

* primary and foreign keys
* data type
* data size
* validation

1. Use the completed data dictionary provided by your teacher to create the Result table. You should pay particular attention to:

* primary and foreign keys
* data type
* data size
* required
* validation

1. Set up the relationships between all the tables and enforce referential integrity between the tables.

*\*\* Check your solution with your teacher before continuing. Once you work has been checked, your teacher will provide you with the populated Northbank College database.*

## Part 4: Queries and Calculations

Searching and Sorting

1. A list of all students who meet their tutors on a Monday. The list should show the full name of each student and the meeting day.
2. A list of all students who score over 90 in any assessment. The list should show the student number, mark and title of the assessment; details of the highest assessment mark should be listed first; assessments with the same mark should be in alphabetical order title.
3. A list showing the full names of all students who completed assessments in March 2014 together with the assessment date, the tutor name and student’s assessment mark. The list should be in alphabetical order of tutor name; when two or more tutors are the same, the details should be displayed so that the student with the highest mark is listed first.

Calculated Fields

1. A list showing the student number and assessment titles. A computed field should be used to work out the difference between the student’s mark and the pass mark for the assessment. The list should show only those students who fail the assessment by 5 marks or more.
2. The Head of Department is concerned that the assessments set by P Mensinger are more difficult than other assessments and has decided to reduce each of these pass mark by 5%. Prepare a list showing the assessment title, original pass mark and adjusted pass mark for all assessments set by P Mensinger.

Grouping Data and Aggregate Functions

1. A list showing the name of each tutor and the total number of assessments sat by students in their tutor group.
2. A list showing the total assessment time for each assessment type. The list should be in descending order of total time; when total time is equal, the records should be listed in alphabetical order of assessment type.
3. A list showing each tutor group code, tutor name, as well as the minimum, maximum and average mark for each tutor group (the average should be rounded to 1 decimal place).
4. The Head of Department is concerned that some assessments are more difficult than others. Prepare a list showing each assessment title, its pass mark along with the number of students who have passed each assessment.

Additional Queries

1. A list showing the name of each tutor, the room used to their tutor group and the number of student in the tutor group. The only tutors listed should be those who meet in a room that has the digit '1' in it.
2. A list showing the full name of any student with 'll' in their surname with the student’s total mark for all 7 assessments and the student’s overall percentage.
3. A list showing the assessment code and title of the assessment with a code that has 3 characters starting with the digit '3'. The highest, lowest and average mark for the assessment should also be listed.
4. The home address of Aidan Clark needs to be updated. He now stays at 71 Burnside Road, Gourock, PA19 1UX. Use a single query to edit the relevant record in the database.

## Completed Data Dictionary

*\*\* Only to be issued after completion and submission of Task 2*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Req’d** | **Validation** |
| Student | studentNo | PK | Number |  | Y | Range >= 100000 and <= 199999 |
| forename |  | Text | 20 | Y |  |
| surname |  | Text | 30 | Y |  |
| address |  | Text | 30 | Y |  |
| town |  | Text | 15 | Y |  |
| postcode |  | Text | 8 | Y |  |
| tgCode | FK | Text | 5 | Y | Existing tgCode from TutorGroup table |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Req’d** | **Validation** |
| TutorGroup | tgCode | PK | Text | 5 | Y |  |
| tutor |  | Text | 15 | Y |  |
| room |  | Text | 5 | Y |  |
| day |  | Text | 9 | Y | Restricted choice (one of Monday, Tuesday, Wednesday, Thursday or Friday) |
| time |  | Time |  | Y |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Assessment | assessmentCode | PK | Text | 6 | Y |  |
| title |  | Text | 30 | Y |  |
| assessmentType |  | Text | 30 | Y | Restricted choice (one of Multiple Choice, Short Answer, Extended Response, or Practical) |
| duration |  | Number |  | Y | Range >= 15 and <= 150 |
| writer |  | Text | 15 | Y |  |
| passMark |  | Number |  | Y | Range >= 0 and <= 100 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Result | studentNo | **PK/FK** | **Number** | **N** | Y | **Existing studentNo from Student table** |
| assessmentCode | **PK/FK** | **Text (6)** | **N** | Y | **Existing assessmentCode from Assessment table** |
| mark | **PK** | **Number** | **N** | Y | **>=0 and <=100** |
| assessmentDate |  | Date |  | Y | Range >= 2013-08-15 and <= 2014-05-12 |

# Implementation: Queries that use another query

Open the file Clydeview.db file. The tables called Country and City are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for any computed values displayed.

1. Display the name, capital city and total population of the country with the largest total population.
2. Display details the name, country and population of any city that has a population which is at least 5,000,000 more than the average population of all the cities.
3. Display the name of any city that is further north than Reykjavik (the latitude of these cities is greater than the latitude of Reykjavik). The query should show the name of each relevant city, its latitude and country name. The city that is furthest north should be listed first.
4. Display the name and population of any city in the United Kingdom that has a population which is more than the average population of all the cities in Bolivia. Arrange these details from smallest to largest population.
5. Display the total number of countries with an area less than 1% of that of the country with the largest area.

# Implementation: Queries that use another query

Open the file Clydeview.db file. The tables called Holiday and Hotel are shown below.



Create SQL queries to display the required details. An alias should be used to display a meaningful heading for any computed values displayed.

1. Display the name, destination, country and distance from the airport of the hotel that is furthest from the airport.
2. Display the name and star rating of any hotel with a rating that is poorer than the average star rating of all the holidays that have the word ‘Break’ or ‘Package’ in their title. The hotel with the highest star rating should be listed first; hotels with the same star rating should be listed in alphabetical order of hotel name.
3. Display the name, city and price per night of any hotel which is dearer that the dearest hotel in Edinburgh. List the hotel details with the dearest hotel first; hotels with the same price should be listed in alphabetical order of city.
4. Display the number of holidays that have the same star rating as that of the ‘Der Wald’ hotel.
5. Display the title, city and distance from the airport of any holiday to Lisbon that is closer to the airport than the average distance from the airport of all the hotels in Spain.
6. Display the title, departure date and duration of any holiday that has the same duration as the longest holiday to a city with the letter ‘o’ as the second character of the city name. Arrange these details so that the most recent holiday is listed first.
7. Design: Cardinality 1

**Exercise 1 – Answers**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | *(a)* | One-to-one | | | | | | | |
|  | *(b)* | Many-to-many | | | | | | | |
|  | *(c)* | One-to-many | | | | | | | |
|  | *(d)* | Many-to-many | | | | | | | |
|  | *(e)* | One-to-many | | | | | | | |
| 2. | *(a)* | Entities: Shop and Worker  Cardinality: One-to-many | | | | | | | |
|  | *(b)* | Entities: Manager and Department  Cardinality: One-to-one | | | | | | | |
|  | *(c)* | Entities: Resort and Hotel  Cardinality: One-to-many | | | | | | | |
|  | *(d)* | Entities: Team and Player  Cardinality: One-to-many | | | | | | | |
|  | *(e)* | Entities: Lecturer and Course  Cardinality: One-to-one | | | | | | | |
|  | *(f)* | Entities: Flight and Airport  Cardinality: Many-to-many | | | | | | | |
|  | *(g)* | Entities: Order and Product  Cardinality: Many-to-many | | | | | | | |
|  | *(h)* | Entities: Customer and Order  Cardinality: One-to-many | | | | | | | |
| 3. | *(a)* | One-to-many | | | | | | | |
|  | *(b)* | Each degree is studied by many students and each student studies many degrees. | | | | | | | |
| 4. | *(a)* | For example: | | | | | | | |
|  |  | **WARD Entity** | | | | **PATIENT Entity** | | | |
|  |  | wardID  type  capacity  freeBeds | | | | patientRef  firstName  lastName  DOB  consultantID  wardID \* | | | |
|  | *(b)* | Many-to-one | | | | | | | |
|  |  |  | | | | | | | |
|  | *(c)* |  | | | | | | | |
|  |  |  | | | | | | | |
| 5. | *(a)* | For example: | | | | | | | |
|  |  | **CD Entity** | | **Track Entity** | | | | **Artist Entity** | |
|  |  | cdNumber  title  label  artisitID \* | | trackNumber  cdNumber \*  title  duration | | | | artistID  name  numMembers  manager | |
|  | *(b)* | Many-to-many | | | | | | | |
|  | *(c)* | Many-to-many | | | | | | | |
|  |  |  | | | | | | | |
|  | *(d)* | Track |  | | CD | |  | | Artist |
| appears on | | features | |
|  |  | | | | | | | | |
| 6. | *(a)* | **Patient: Consultant** is a one-to-many relationship  **Consultant: MedicalTopic** is a one-to-many relationship | | | | | | | |
|  | *(b)* | **Patient: Consultant**  Each patient is treated by one consultant and each consultant treats many patients.  **Consultant: MedicalTopic**  Each consultant specialises in one medical topic and each medical topic is the specialism of many consultants. | | | | | | | |
| 7. | *(a)* | **Engineer: Project** is a many-to-many relationship  **Project: Manager** is a one-to-many relationship | | | | | | | |
|  | *(b)* | **Engineer: Project**  Each engineer works on many projects and each project has many engineers working on it.  **Project: Manager**  Each project is managed by one manager and each manager manages many projects. | | | | | | | |

1. Design: Cardinality 2

**Exercise 2 – Answers**

Task 1

|  |  |
| --- | --- |
| *(a)* | One-to-many |
| *(b)* | Each member makes many posts and each post is made by one member. |

Task 2

|  |  |
| --- | --- |
| *(a)* | Many-to-many |
| *(b)* | Each customer has many different types of account and each type of account is held by many customers. |

Task 3

|  |  |
| --- | --- |
| *(a)* | One-to-many |
| *(b)* | One-to-one |
| *(c)* | **Item: Brand**  Each item belongs to one brand and each brand sells many different items  **Brand: Founder**  Each brand has one founder and each founder established one brand. |

Task 4

|  |  |
| --- | --- |
| *(a)* | One-to-many |
| *(b)* | One-to-one |
| *(c)* | **Continent: Country**  Each continent has many countries and each country is located in one continent.  **Country: CapitalCity**  Each country has one capital city and each capital city is the capital of one country. |

1. Design: Cardinality 3

**Exercise 3 – Answers**

|  |  |  |
| --- | --- | --- |
| 1. | *(a)* | One-to-many |
|  | *(b)* | One-to-many |
|  | *(c)* | One-to-many |
|  | *(d)* | **Student: Assignment**  Each student submits many assignments and each assignment is submitted by one student.  **Task: Assignment**  Each assignment consists of many tasks and each task is part of one assignment.  **Task: Tutor**  Each task is marked by one tutor and each tutor marks many tasks. |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. | *(a)* | For example: | | | | | | |
|  |  | **MOVIE** | | **ACTOR** | | | **DIRECTOR** | |
|  |  | movieRef  title  duration  classification | | actorNum  firstName  lastName  DOB | | | directorID  firstName  lastName  nationality | |
|  | *(b)* | Many-to-many | | | | | | |
|  | *(c)* | Many-to-many | | | | | | |
|  | *(d)* |  | | | | | | |
|  |  | ACTOR |  | | MOVIE |  | | DIRECTOR |
| stars in | | is directed by | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3. | *(a)* | For example: | | | | | | |
|  |  | | | | | | | |
|  |  | **Continent** | | **Country** | | | **City** | |
|  |  | continent  area(km2)  population  numberCountries | | country  continent \*  population  area(kn2) | | | city  population  area  country \* | |
|  |  |  | | | | | | |
|  | *(b)* | One-to-many | | | | | | |
|  |  |  | | | | | | |
|  | *(c)* | One-to-many | | | | | | |
|  |  |  | | | | | | |
|  | *(d)* | Continent |  | | Country |  | | City |
| is location for | | has | |

1. Design: Entity Occurrence Diagrams 1

**Exercise 4 – Answers**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 1. | **Booking** |  | **Customer** |  |  |
|  |  |  |  |  |  |
|  | BD223145 ⚫ |  |  |  |  |
|  | NF874607 ⚫ |  | ⚫ 1234 |  |  |
|  | KL763832 ⚫ |  | ⚫ 2804 |  |  |
|  | TG803017 ⚫ |  | ⚫ 3456 |  |  |
|  | JT397282 ⚫ |  |  |  |  |
|  |  | | | | |
|  |  | | | | |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2. | **Driver** |  | **Taxi** |  |  |
|  |  |  |  |  |  |
|  | 03 ⚫ |  | ⚫ AS512ER |  |  |
|  | 15 ⚫ |  | ⚫ TY528UQ |  |  |
|  | 18 ⚫ |  | ⚫ VJ537KL |  |  |
|  |  |  |  |  |  |
|  |  | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 3. | *(a)* | **Course** |  | **Tutor** |  |  |
|  |  | |  |  |  |  |
|  | D556 ⚫ | |  | ⚫ 1989 |  |  |
|  | D766 ⚫ | |  | ⚫ 3456 |  |  |
|  | C991 ⚫ | |  | ⚫ 6515 |  |  |
|  | M101 ⚫ | |  | ⚫ 7817 |  |  |
|  |  | |  |  |  |  |
|  |  | |  |  |  |  |
|  |  | |  |  |  |  |
|  | *(b)* | **Course** |  | **Student** |  |  |
|  |  | |  |  |  |  |
|  | D556 ⚫ | |  | ⚫ 18728 |  |  |
|  | D766 ⚫ | |  |  |  |  |
|  | C991 ⚫ | |  |  |  |  |
|  | M101 ⚫ | |  | ⚫ 18729 |  |  |
|  |  | |  |  |  |  |
|  |  | |  |  |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 4. | *(a)* | **Teacher** |  | **Class** |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  | ⚫ E23 |  |  |
|  |  | 123 ⚫ |  | ⚫ M29 |  |  |
|  |  | 145 ⚫ |  | ⚫ E31 |  |  |
|  |  | 168 ⚫ |  | ⚫ S79 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  | | | | |
|  | *(b)* | **Pupil** |  | **Class** |  |  |
|  |  |  |  |  |  |  |
|  |  | 37690065 ⚫ |  | ⚫ M29 |  |  |
|  |  | 39879342 ⚫ |  | ⚫ E23 |  |  |
|  |  | 40438271 ⚫ |  | ⚫ E31 |  |  |
|  |  | 41414141 ⚫ |  | ⚫ S79 |  |  |
|  |  |  |  |  |  |  |
|  |  | | | | | |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 5. | *(a)* | **Employee** |  | **Pilot** |  |  |
|  |  |  |  |  |  |  |
|  |  | 12345 ⚫ |  |  |  |  |
|  |  | 13579 ⚫ |  | ⚫ 12345 |  |  |
|  |  | 15050 ⚫ |  | ⚫ 15050 |  |  |
|  |  | 17531 ⚫ |  | ⚫ 19753 |  |  |
|  |  | 19753 ⚫ |  |  |  |  |
|  |  |  | | | | |
|  |  |  | | | | |
|  |  |  | | | | |
|  | *(b)* | **Employee** |  | **Admin** |  |  |
|  |  |  |  |  |  |  |
|  |  | 12345 ⚫ |  |  |  |  |
|  |  | 13579 ⚫ |  | ⚫ 13579 |  |  |
|  |  | 15050 ⚫ |  |  |  |  |
|  |  | 17531 ⚫ |  | ⚫ 17531 |  |  |
|  |  | 19753 ⚫ |  |  |  |  |
|  |  |  | | | | |
|  |  |  | | | | |

1. Design: Entity Occurrence Diagrams 2

**Exercise 5 – Answers**

Task 1

Diagram A

Task 2

Diagram D

Task 3

One-to-one

Task 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *(a)* | **Customer** |  | **Rental** |  |  |
|  |  |  |  |  |  |
|  |  |  | ⚫ 1 |  |  |
|  |  |  | ⚫ 2 |  |  |
|  | 1001 ⚫ |  | ⚫ 3 |  |  |
|  | 1002 ⚫ |  | ⚫ 4 |  |  |
|  | 1003 ⚫ |  | ⚫ 5 |  |  |
|  |  |  | ⚫ 6 |  |  |
|  |  |  |  |  |  |
|  |  | | | | |
|  |  | | | | |
| *(b)* | One-to-many | | | | |

Task 5

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *(a)* | **Rental** |  | **Vehicle** |  |  |
|  |  |  |  |  |  |
|  | 1 ⚫ |  |  |  |  |
|  | 2 ⚫ |  | ⚫ AB12 JKL |  |  |
|  | 3 ⚫ |  | ⚫ AB22 MNB |  |  |
|  | 4 ⚫ |  | ⚫ BA32 MKL |  |  |
|  | 5 ⚫ |  | ⚫ CD41 PLM |  |  |
|  | 6 ⚫ |  |  |  |  |
|  |  | | | | |
|  |  | | | | |
| *(b)* | One-to-many | | | | |

1. Design: Compound Keys

**Exercise 6 – Answers**

Task 1

The Player table below is used to store details of footballers who play in a local junior league. Determine whether values stored in the fields of the table would be unique and give a reason in each case.

|  |  |  |  |
| --- | --- | --- | --- |
| **Player** | | | |
| **Field** | **Type** | **Unique?** | **Reason** |
| fullName | Text | no | There could be two players with the same name |
| position | Text | no | Each team will have a player in each position meaning that there will be several instances with the same position |
| shirtNumber | Number | no | Each team will use the same shirt numbers meaning that there will be several instances with the same shirt number |
| injured | Boolean | no | There are only two possible values which will be used more than once |
| team | Text | no | All players in the same team will use the same value in this field |

Primary key = team + shirtNumber

Task 2

The House table below is used to store details of houses for sale in an Estate Agency. Determine whether values stored in the fields of the table would be unique and give a reason in each case.

|  |  |  |  |
| --- | --- | --- | --- |
| **House** | | | |
| **Field** | **Type** | **Unique?** | **Reason** |
| number | Number | no | Two different house could have the same house number on different streets |
| street | Text | no | Two or more houses on the same street could be for sale |
| colour | Text | no | There could be two or more houses that are the same colour |
| postcode | Text | no | Two or more houses could share the same postcode |
| price | Number | no | There could be two houses on sale for the same price |

Primary key = postcode + number

Task 3

Possible primary key = customerID + date + time

Task 4

Primary key = homeTeam + awayTeam + date

Task 5

Primary key = make + model + colour

Task 6

Primary key = make + model + colour

1. Design: Entity Relationship Diagrams 1

**Exercise 7 – Answers**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | | Student |  | Course |  | | Lecturer | |
| studies | is taught by | |
|  | |  | | | | | | |
| 2. | | Pupil |  | BookCopy |  | | Book | |
| take out | is a copy of | |
|  | |  |  |  |  | written by | |  |
|  | |  |  |  |  | | Author | |
|  | |  |  |  |  | |  | |
|  | |  |  |  |  | |  | |
|  | |  |  |  |  | |  | |
| PRODUCT  SHOP  ORDER  SUPPLIER  CUSTOMER  MOBILE PHONE  ACCESSORY  sells  is a  is a  supplies  places  is for | |  |  |  |  | |  | |
| 3. |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
| 4. | - see over | | | | | | | |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. | *(a)* | Use this description to identify the entities and relationships in the system. The first few have been started for you.   * Each EMPLOYEE works in only one DEPARTMENT * Each DEPARTMENT employs many EMPLOYEEs * Each DEPARTMENT works on many PROJECTs * Each PROJECT many need many DEPARTMENTs * Each PROJECT may require many RESOURCEs * Each RESOURCE could be required on many PROJECTs | | | | | | |
|  |  |  | | | | | | |
|  |  |  | | | | | | |
|  | *(b)* | EMPLOYEE |  | DEPARTMENT |  | | PROJECT | |
| works in | works on | |
|  |  |  |  |  |  | requires | |  |
|  |  |  |  |  |  | | RESOURCE | |

1. Design: Entity Relationship Diagrams 2

**Exercise 8**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. | Teacher |  | | Class | |  | Room | |
| teaches | | uses |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
| 2. | Race |  | | Boat | |  | Class | |
| involves | | belongs to |
|  |  |  | | has |  |  | |  |
|  |  |  | | Skipper | |  |  | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
| 3. |  |  | | Flight | |  | Airport | |
| takes off from |
|  |  |  | is for | |  |  |  | |
|  | Customer |  | | Booking | |  | Vehicle | |
| makes | | has |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
|  |  | | | | | | | |
| 4. | Shop |  | | Order | |  | Salesperson | |
| places | | is processed by |
|  |  |  | is made of up | |  |  |  | |
|  |  |  | | Item | |  | Till | |
|  | | is for |

1. Design: Data Dictionary 1

**Exercise 9 – Answers**

Task 1

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Key** | **Type** | **Size** | **Reqd** | **Validation** |
| **Entity**: Customer | | | | | |
| customerNumber | PK | Number |  | Yes |  |
| customerName |  | Text | 30 | Yes |  |
| customerAddress |  | Text | 50 | Yes |  |
| customerTown |  | Text | 20 | Yes |  |
| customerPostcode |  | Text | 8 | Yes |  |
| **Entity**: Order | | | | | |
| orderReference | PK | Text | 8 | Yes | Length = 8 |
| orderDate |  | Date |  | Yes |  |
| isbn | FK | Text | 11 | Yes |  |
| quantity |  | Number |  | Yes | Range: >=1 and <=24 |
| customerNumer | FK | Number |  | Yes | Existing customerNumber from Customer table |
| **Entity**: Book | | | | | |
| isbn | PK | Text | 11 | Yes | Length = 11 |
| bookTitle |  | Text | 50 | Yes |  |
| category |  | Text | 11 | Yes | Restricted Choice: Children, Crime, Historical, Large Print, Non-Fiction, Romance |
| price |  | Number |  | Yes | Range: >= 0.99 and <= 38.95 |
| authorCode | FK | Number |  | Yes | Existing authorCode from Author table |
| **Entity**: Author | | | | | |
| authorCode | PK | Number |  | Yes |  |
| authorName |  | Text | 30 | Yes |  |
| authorAddress |  | Text | 30 | Yes |  |
| authorTown |  | Text | 20 | Yes |  |
| authorPostcode |  | Text | 8 | Yes |  |

Task 2

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attribute Name** | **Key** | **Type** | **Size** | **Reqd** | **Validation** |
| **Entity**: Member | | | | | |
| memberID | PK | Number |  | Yes |  |
| surname |  | Text | 30 | Yes |  |
| address |  | Text | 40 | Yes |  |
| town |  | Text | 20 | Yes |  |
| phoneNumber |  | Text | 11 | Yes | Length = 11 |
| dateJoined |  | Date |  | Yes |  |
| priorityCode |  | Number |  | No | Range: >= 1111 and <= 9999 |
| **Entity**: Rental | | | | | |
| rentalNumber | PK | Number |  | Yes |  |
| memberID | FK | Number |  | Yes | Existing memberID from Member table |
| dvdID | FK | Number |  | Yes | Existing dvdID from DVD table |
| dateDueBack |  | Date |  | Yes |  |
| Late? |  | Boolean |  | Yes |  |
| **Entity**: DVD | | | | | | |
| dvdID | PK | Number |  | Yes |  | |
| dvdTitle |  | Text | 50 | Yes |  | |
| certitificate | FK | Text | 2 | Yes | Existing certificate from Certificate table | |
| **Entity**: Certificate | | | | | | |
| certificate | PK | Text | 2 | Yes | Restricted Choice: U, PG, 12, 15, 18 | |
| description |  | Text | 35 | Yes | Restricted Choice: Universal, Parental Guidance, Suitable for 12 years and over, Suitable only for 15 years and over, Suitable for adults only | |

1. Design: Data Dictionary 2

**Exercise 10 – Answers**

Task 1

A presence check has been used for all mandatory fields marked \* (Email, Password, Title, First Name, Last Name, Date of Birth and Contact Number). The presence check will be used to ensure that a value is entered for each of these fields. If of these fields is left blank, the user will receive an error message.

A restricted choice has been used for the Title and Date of Birth fields. Use of a restricted choice will eliminate date entry errors since inly valid options are presented to the user.

Task 2

A presence check has been used. This is used to ensure that search criteria are entered and has not been left blank.

Task 3

A range check has been applied. This ensures that the quantity entered by the user is between 2 and 20 inclusive. If any other values are entered, the user will receive an error message.

Task 4

A length check has been used to ensure that the SCN entered has exactly 9 digits. If the user enters a candidate number with fewer (or more) than 9 digits, they will receive an error message.

A presence check has been used for all mandatory fields marked \* (SCN, Date of Birth, Surname, Postcode or Country, Email address, Re-type email address, Terms and conditions). The presence check will be used to ensure that a value is entered for each of these fields. If of these fields is left blank, the user will receive an error message.

A restricted choice is used for the Country field. This will display a list of possible countries and the user simply selects the one country that applies. Use of a restricted choice will eliminate date entry errors since inly valid options are presented to the user.

Task 5

A range check has been applied. This ensures that the estimate grade entered by schools is between 1 and 9 inclusive. If any other values are entered, the user will receive an error message.

Task 6

The Result table has 3 foreign keys, SCN and QualCode. To ensure that referential integrity is met, all values entered into a foreign key must already exist in the matching primary keys. In this case, an SCN must already exist in the Candidate table before it can be entered in the Result table. Also, a QualCode must already exist in the Qualification table before it can be entered in the Result table.

Task 7

The memberID attribute of the Rental table is a foreign key used to link the Member and Rental tables. Before a value can be entered in this field, a matching record must already exist in the Member table. By setting referential integrity, the developer ensures that only known and existing members can make rentals.

Task 8

The albumID attribute of the Photo table is a foreign key used to link the Album and Photo tables. Before a value can be entered in this foreign key, a matching record must already exist in the Album table. By setting referential integrity, the developer ensures that photos can only be stored in albums that have already been created.

Task 9

The Purchase table include the attribute customerID which is the foreign key used to link the Purchase and Customer tables. Before details of a purchase can be recorded, a matching record must already exist in the Customer table. The use of referential integrity ensures that only registered customers can make purchase.

1. Design: Queries that use another query

1. Query1(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Largest Country Population = MAX(totalPopulation) |
| Table(s) and query(queries) | Country |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query1(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | countryName, capital, Total Population = totalPopulation |
| Table(s) and query(queries) | Country, Query1(1) |
| Search criteria | totalPopulation = Largest Country Population |
| Grouping |  |
| Sort order |  |

1. Query2(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Average City Population = Round(AVG(population),2) |
| Table(s) and query(queries) | City |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query2(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | cityName, countryName, population |
| Table(s) and query(queries) | City, Country, Query2(1) |
| Search criteria | population ≥ 5000000 + Average City Population |
| Grouping |  |
| Sort order |  |

1. Query3(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Reykjavik Latitude = latitude |
| Table(s) and query(queries) | City |
| Search criteria | cityName = Reykjavik |
| Grouping |  |
| Sort order |  |

Query3(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | cityName, latitude, countryName |
| Table(s) and query(queries) | City, Country, Query3(1) |
| Search criteria | latitude > Reykjavik Latitude |
| Grouping |  |
| Sort order | latitude DESC |

1. Query4(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Average Population of Bolivian Cities = Round (AVG(population),2) |
| Table(s) and query(queries) | City, Country |
| Search criteria | countryName = Bolivia |
| Grouping |  |
| Sort order |  |

Query4(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | cityName, population |
| Table(s) and query(queries) | City, Query4(1) |
| Search criteria | countryName = United Kingdom AND population > Average Population of Bolivian Cities |
| Grouping |  |
| Sort order | population ASC |

1. Query5(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Largest Country Area = MAX(area) |
| Table(s) and query(queries) | Country |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query5(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Number of Countries = COUNT(\*) |
| Table(s) and query(queries) | Country, Query5(1) |
| Search criteria | area < 0.01 \* Largest Country Area |
| Grouping |  |
| Sort order |  |

1. Design: Queries that use another query
2. Query1(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Furthest from Airport = MAX(kilometresFromAirport) |
| Table(s) and query(queries) | Hotel |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query1(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | hotelName, destination, country, Distance from the Airport = kilometresFromAirport |
| Table(s) and query(queries) | Hotel, Holiday, Query1(1) |
| Search criteria |  |
| Grouping | kilomeresFromAirport = Furthest from Airport |
| Sort order |  |

1. Query2(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Average Rating = AVG(starRating) |
| Table(s) and query(queries) | Hotel |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query2(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | hotelName, starRating |
| Table(s) and query(queries) | Hotel, Holiday, Query2(1) |
| Search criteria | starRating < Average Rating AND (title LIKE \*Break\* OR title LIKE \*Package\*) |
| Grouping |  |
| Sort order | starRating DESC, hotelName ASC |

1. Query3(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Dearest Edinburgh Hotel = MAX(pricePerNight) |
| Table(s) and query(queries) | Hotel |
| Search criteria |  |
| Grouping |  |
| Sort order |  |

Query3(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | hotelName, city, pricePerNight |
| Table(s) and query(queries) | Hotel, Query3(1) |
| Search criteria | pricePerNight > Dearest Edinburgh Hotel |
| Grouping |  |
| Sort order | pricePerNight DESC, city ASC |

1. Query4(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Der Wald Hotel Rating = starRating |
| Table(s) and query(queries) | Hotel |
| Search criteria | hotelName = Der Wald |
| Grouping |  |
| Sort order |  |

Query4(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Number of Holidays = COUNT(\*) |
| Table(s) and query(queries) | Holiday, Hotel, Query4(1) |
| Search criteria | starRating = Der Wald Hotel Rating |
| Grouping |  |
| Sort order |  |

1. Query5(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Average Distance from Airport = AVG(kilometresFromAirport) |
| Table(s) and query(queries) | Hotel, Holiday |
| Search criteria | country = Spain |
| Grouping |  |
| Sort order |  |

Query5(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | title, city, Distance from Airport = kilometresFromAirport |
| Table(s) and query(queries) | Holiday, Hotel, Query5(1) |
| Search criteria | city = Lisbon AND kilometresFromAirport < Average Distance from Airport |
| Grouping |  |
| Sort order |  |

1. Query6(1)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | Longest Holiday = MAX(nights) |
| Table(s) and query(queries) | Holiday |
| Search criteria | city LIKE ?o\* |
| Grouping |  |
| Sort order |  |

Query6(2)

|  |  |
| --- | --- |
| Field(s) and calculation(s) | title, departureDate, nights AS Duration |
| Table(s) and query(queries) | Holiday, Query6(1) |
| Search criteria | nights = Longest Holiday |
| Grouping |  |
| Sort order | departureDate DESC |

1. Implementation: SQL Wildcards – Done
2. Which of the following strings satisfies the condition 'T\_'? (There can be more than one answer)
   1. TW 
   2. ~~TWITTER~~
   3. ~~TAIWAN~~
   4. To 
3. Which of the following strings satisfies the condition 'A\_B%'? (There can be more than one answer)
   1. ~~AKKB~~
   2. AKBKKK 
   3. ~~ABKK~~
   4. ABBB 
4. Which of the following strings satisfies the condition 'Z%K\_R'? (There can be more than one answer)
   1. ZKRR 
   2. ~~ZYKR~~
   3. ~~ZRKKJ~~
   4. ZABCDEFKGR 
5. Write the conditions used to list productNames that:
   1. start with the letter 'S'.

productName LIKE 'S%'

* 1. end with the letters 'on'.

productName LIKE '%on'

* 1. contain the pattern 'ill'.

productName LIKE '%ill%'

* 1. contain the letters 'o' and 'n' (in that order) with exactly one letter between them.

productName LIKE '%o\_n%'

* 1. contain the patterns 'aw' and 'ing' separated by exactly two letters.

productName LIKE '%aw\_\_ing%'

* 1. start with the pattern 'St' which is followed by any character which in turn is followed by the letter 'a' and ends with the letter 's'.

productName LIKE 'St\_a%s'

1. Which of the following SQL statement is valid? (There can be more than one answer)
   1. ~~SELECT firstName, lastName~~

~~FROM Sales~~

~~WHERE firstName LIKE 'A%' lastName LIKE 'W%';~~

* 1. SELECT firstName, lastName

FROM Sales

WHERE firstName LIKE 'J%' AND lastName LIKE 'W%'; 

* 1. ~~SELECT firstName, lastName~~

~~FROM Sales~~

~~firstName LIKE 'J%' AND lastName LIKE 'W%';~~

* 1. ~~SELECT firstName, lastName~~

~~FROM Sales~~

~~WHERE firstName LIKE 'J%', lastName LIKE 'W%'~~

1. How many records will be returned by the following query? (Assume the database is not case sensitive)

SELECT \*

FROM Sales

WHERE lastName LIKE '%l\_e%';

2 records are returned. They are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **firstName** | **lastName** | **birthdate** | **joinDate** | **totalSales** |
| Sophie | Lee | 05/01/1960 | 05/04/2015 | 500 |
| Jill | Wilkes | 20/11/1979 | 15/04/2015 | 210 |

1. How many records will be returned by the following query? (Assume the database is not case sensitive)

SELECT \*

FROM Sales

WHERE firstName LIKE '%i%'

OR lastName LIKE '%e%';

4 records are returned. They are:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **firstName** | **lastName** | **birthdate** | **joinDate** | **totalSales** |
| Sophie | Lee | 05/01/1960 | 05/04/2015 | 500 |
| Richard | Brown | 07/01/1975 | 05/04/2015 | 200 |
| Casey | Healy | 20/09/1996 | 09/04/2015 | 80 |
| Jill | Wilkes | 20/11/1979 | 15/04/2015 | 210 |

**Marks: 15**

1. Implementation: SQL Wildcards – Done

SELECT forename, surname, postcode, monthOfRenewal

FROM Member

WHERE surname LIKE "D%";

|  |  |  |  |
| --- | --- | --- | --- |
| **forename** | **surname** | **postcode** | **monthOfRenewal** |
| Sopoline | Dominguez | P0R 8UL | November |
| Inga | Davenport | YL4Z 7JQ | August |
| Lev | Dillon | TU7 7GD | August |
| Sylvia | Duffy | EB3 3ZM | May |
| Cadman | Donovan | IP7A 4UF | April |

SELECT membershipNumber, surname, address, town, postcode

FROM Member

WHERE surname LIKE "%oo%"

ORDER BY surname ASC;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| BFU236 | Booker | 15 Feugiat Street | Port Worland | Y9 5JM |
| BWE112 | Brooks | 76 Odio Road | Muncie | E58 8HC |
| YBA200 | Goodman | 27 Aliquet Street | Baldwin | YU5T 0VE |
| AXE117 | Hooper | 73 Ipsum Street | Lebaness | HD8Z 5SB |
| RUD180 | Underwood | 48 Aliquet Street | Baldwin | HW07 6LB |

SELECT forename, town

FROM Member

WHERE forename LIKE "%o%"

AND town LIKE "B%n";

|  |  |
| --- | --- |
| **forename** | **town** |
| Josephine | Brockton |
| Whoopi | Baldwin |
| Forrest | Baldwin |
| Charlotte | Baldwin |
| Roberta | Baldwin |
| Moana | Benton |

SELECT forename, surname

FROM Member

WHERE surname LIKE "L\_\_\_";

|  |  |
| --- | --- |
| **forename** | **surname** |
| Sage | Lang |
| Vernon | Leon |
| Briony | Lane |

SELECT membershipNumber, town, postcode

FROM Member

WHERE postcode LIKE "%a\_2%";

|  |  |  |
| --- | --- | --- |
| **membershipNumber** | **town** | **postcode** |
| CDG191 | Baldwin | AI26 2YL |
| FMH120 | Worland | AY24 2KU |
| OBX110 | Lebaness | A82 8BN |
| QYG131 | Brockton | A22 3TI |
| ZDA101 | Allentown | AI2A 7ZF |

SELECT firstname, surname, typeOfMembership

FROM Member

WHERE surname LIKE "%i\_\_e%";

|  |  |  |
| --- | --- | --- |
| **forename** | **surname** | **typeOfMembership** |
| Dai | Whitney | Adult |
| Wynne | Bridges | Adult |
| Nita | Fisher | Student |
| Jacqueline | Skinner | Adult |
| Kelly | Nieves | Adult |

SELECT membershipNumber, dateOfBirth, typeOfMembership

FROM Member

WHERE typeOfMembership = "Adult"

AND dateOfBirth LIKE "%-10-%";

|  |  |  |
| --- | --- | --- |
| **membershipNumber** | **dateOfBirth** | **typeOfMembership** |
| BWE112 | 1978-10-25 | Adult |
| ESK183 | 1956-10-11 | Adult |
| GXT253 | 1974-10-01 | Adult |
| NRI190 | 1964-10-12 | Adult |
| RTN159 | 1982-10-06 | Adult |

dateOfBirth LIKE "\_\_\_\_\_10%";

dateOfBirth LIKE "%10\_\_\_";

dateOfBirth LIKE "%10-%";

1. Implementation: SQL Wildcards – Done

SELECT plantName, variety

FROM Plant

WHERE variety LIKE "%x%"

ORDER BY plantName ASC;

|  |  |
| --- | --- |
| **plantName** | **variety** |
| Apple | Cox |
| Eunonymous | Mixed |

SELECT category, plantName, code, price

FROM Plant

WHERE code LIKE "\_\_";

|  |  |  |  |
| --- | --- | --- | --- |
| **category** | **plantName** | **code** | **price** |
| Patio | Eunonymous | VY | 12.5 |
| Patio | Carnation | RP | 8.25 |
| Climber | Honeysuckle | YP | 8.25 |
| Perennial | Astilbe | WP | 3.5 |
| Perennial | Phlox | RW | 3.3 |

SELECT plantName, code, height

FROM Plant

WHERE code LIKE "%P%"

AND plantName LIKE "%a\_t%";

|  |  |  |
| --- | --- | --- |
| **plantName** | **code** | **height** |
| Astilbe | WP | M |

SELECT plantName, referenceID, price

FROM Plant

WHERE plantName LIKE "%a%a"

ORDER BY price DESC,

referenceID ASC;

|  |  |  |
| --- | --- | --- |
| **plantName** | **referenceID** | **price** |
| Pyracantha | H60 | 30.0 |
| Magnolia | S73 | 15.0 |
| Magnolia | S79 | 15.0 |
| Camellia | S60 | 13.0 |
| Hydrangea | S68 | 7.75 |

SELECT code, referenceID, category

FROM Plant

WHERE referenceID LIKE "\_3\_"

AND code LIKE "%r%"

ORDER BY category ASC;

|  |  |  |
| --- | --- | --- |
| **code** | **referenceID** | **category** |
| R | F31 | Fruit |
| PR | U37 | Fuchsia |
| RP | B35 | Patio |

SELECT plantName, unit, price

FROM Plant

WHERE plantName LIKE "%a\_\_n%"

ORDER BY unit DESC,

price ASC;

|  |  |  |
| --- | --- | --- |
| **plantName** | **Unit** | **Price** |
| Pyracantha | 5 | 30.0 |
| Aconitem | 1 | 3.0 |
| Daphne | 1 | 16.5 |

SELECT referenceID, plantName, variety, height

FROM Plant

WHERE plantName LIKE "\_e%"

AND variety LIKE "C%e";

|  |  |  |  |
| --- | --- | --- | --- |
| **referenceID** | **plantName** | **variety** | **height** |
| B21 | Begonia | Cocunut Ice | S |
| B22 | Begonia | Cocunut Ice | S |
| B23 | Begonia | Cocunut Ice | S |
| F32 | Pear | Conference | T |

**Marks: 28**

1. Implementation: Calculations and Computed Fields – Done

SELECT forename, surname, test1 + test2 + test3 + test4

FROM Table1;

|  |  |  |
| --- | --- | --- |
| **forename** | **surname** | **test1 + test2 + test3 + test4** |
| Jane | Smith | 30 |
| Duncan | Scott | 35 |
| James | Webster | 30 |
| Julie | O'Brian | 27 |
| Mary | Davis | 30 |

SELECT Forename, test1, test2, test3, test4,

(test1 + test2 + test3 + test4)/4.0

FROM Table1

ORDER BY (test1 + test2 + test3 + test4)/4.0 DESC;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **forename** | **test1** | **test2** | **test3** | **test4** | **(test1 + test2 + test3 + test4)/4.0** |
| Duncan | 10 | 8 | 9 | 8 | 8.75 |
| Annie | 7 | 8 | 8 | 10 | 8.25 |
| Adam | 10 | 8 | 7 | 6 | 7.75 |
| Jane | 9 | 7 | 8 | 6 | 7.5 |
| James | 8 | 6 | 7 | 9 | 7.5 |

SELECT forename, surname, hourlyRate \* hoursWorked

FROM Table2;

|  |  |  |
| --- | --- | --- |
| **forename** | **surname** | **hourlyRate \* hoursWorked** |
| Kenneth | Brown | 150.0 |
| David | Gilmour | 204.0 |
| Ashley | Grant | 210.0 |
| Vickie | Moore | 210.0 |
| Laura | Green | 90.0 |

SELECT forename, surname

FROM Table2

WHERE hourlyRate \* hoursWorked > 200;

|  |  |
| --- | --- |
| **forename** | **surname** |
| David | Gilmour |
| Ashley | Grant |
| Vickie | Moore |

SELECT test1 / 16.0 \* 100, test2 / 16.0 \* 100, test3 / 16.0 \* 100,

test4 / 16.0 \* 100, test5 / 16.0 \* 100

FROM Table3;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **test1 / 16.0 \* 100** | **test2 / 16.0 \* 100** | **test3 / 16.0 \* 100** | **test4 / 16.0\* 100** | **test5 / 16.0 \* 100** |
| 93.75 | 75.0 | 75.0 | 81.25 | 87.5 |
| 75.0 | 68.75 | 87.5 | 81.25 | 100.0 |
| 87.5 | 87.5 | 81.25 | 93.75 | 93.75 |
| 62.5 | 56.25 | 75.0 | 68.75 | 87.5 |
| 87.5 | 87.5 | 87.5 | 75.0 | 87.5 |

SELECT forename, surname, (test1+test2+test3+test4+test5)/80.0\*100

FROM Table3

ORDER BY (test1+test2+test3+test4+test5)/80.0\*100 ASC,

surname ASC;

|  |  |  |
| --- | --- | --- |
| **Forename** | **Surname** | **(test1+test2+test3+test4+test5)/80.0\*100** |
| Billy | Johnston | 70.0 |
| Bill | Johnstone | 82.5 |
| Harry | Smith | 82.5 |
| Tom | Harris | 85.0 |
| Ann | Clark | 88.75 |

SELECT productName, buyingPrice, sellingPrice,

sellingPrice-buyingPrice

FROM Table4;

|  |  |  |  |
| --- | --- | --- | --- |
| **productName** | **buyingPrice** | **sellingPrice** | **sellingPrice-buyingPrice** |
| Mars Bar | 27 | 42 | 15 |
| Snickers | 31 | 25 | -6 |
| Yorkie | 32 | 45 | 13 |
| Bounty | 26 | 21 | -5 |

SELECT productName, sellingPrice-buyingPrice

FROM Table4

WHERE sellingPrice-buyingPrice < 0

ORDER BY sellingPrice-buyingPrice DESC;

|  |  |
| --- | --- |
| **productName** | **sellingPrice-buyingPrice** |
| Bounty | -5 |
| Snickers | -6 |

SELECT productName, priceUK, ROUND(priceUK \* 1.13, 2)

FROM Table5;

|  |  |  |
| --- | --- | --- |
| **productName** | **priceUK** | **ROUND(priceUK \* 1.13, 2)** |
| Basic Keyboard | 7.0 | 7.91 |
| Network Card | 13.5 | 15.25 |
| Optical Mouse | 7.5 | 8.47 |
| Monitor | 30.0 | 33.9 |
| 3D Graphics Card | 30.0 | 33.9 |

SELECT productID, priceUK, ROUND(priceUK \* 1.39, 2)

FROM Table5

WHERE ROUND(priceUK \* 1.39, 2) > 40

ORDER BY priceUK \* 1.39 ASC,

productID DESC;

|  |  |  |
| --- | --- | --- |
| **productID** | **priceUK** | **ROUND(priceUK \* 1.39, 2)** |
| 097 | 30.0 | 41.7 |
| 034 | 30.0 | 41.7 |
| 102 | 75.0 | 104.25 |

SELECT fishType, pricePerKilo, numberOfKilos,

pricePerKilo \* numberOfKilos

FROM Table6;

|  |  |  |  |
| --- | --- | --- | --- |
| **fishType** | **pricePerKilo** | **numberOfKilos** | **pricePerKilo \* numberOfKilos** |
| Anchovies | 15.0 | 2.5 | 37.5 |
| Cod | 12.0 | 3.0 | 36.0 |
| Coley | 7.0 | 1.0 | 7.0 |
| Grey Mullet | 5.5 | 2.5 | 13.75 |
| Gurnard | 5.75 | 3.0 | 17.25 |

SELECT fishType, pricePerKilo \* numberOfKilos

FROM Table6

WHERE pricePerKilo \* numberOfKilos >= 20

AND pricePerKilo \* numberOfKilos <= 50

ORDER BY pricePerKilo \* numberOfKilos DESC;

|  |  |
| --- | --- |
| **fishType** | **pricePerKilo \* numberOfKilos** |
| Monkfish | 40.0 |
| Anchovies | 37.5 |
| Cod | 36.0 |
| Mackeral | 33.95 |
| Snapper | 32.5 |

**Marks: 24**

1. Implementation: Calculations and Computed Fields – Done

UPDATE Table1

SET test3 = test3 + 1;

SELECT \*

FROM Table1;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **pupilID** | **forename** | **surname** | **test1** | **test2** | **test3** | **test4** |
| 1 | Jane | Smith | 9 | 7 | 9 | 6 |
| 2 | Duncan | Scott | 10 | 8 | 10 | 8 |
| 3 | James | Webster | 8 | 6 | 8 | 9 |
| 4 | Julie | O'Brian | 7 | 6 | 10 | 5 |
| 5 | Mary | Davis | 9 | 9 | 8 | 5 |

UPDATE Table2

SET hourlyRate = ROUND(hourlyRate \* 1.1, 2)

WHERE hourlyRate < 7;

SELECT \*

FROM Table2;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **staffID** | **forename** | **surname** | **hourlyRate** | **hoursWorked** |
| 1 | Kenneth | Brown | 7.5 | 20 |
| 2 | David | Gilmour | 6.6 | 34 |
| 3 | Ashley | Grant | 6.6 | 35 |
| 4 | Vickie | Moore | 6.6 | 35 |
| 5 | Laura | Green | 4.95 | 20 |

UPDATE Table3

SET test2 = test2 + 2,

test4 = test4 + 1

WHERE Surname LIKE "S%";

SELECT \*

FROM Table3

WHERE Surname LIKE "J%";

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **studentID** | **forename** | **surname** | **test1** | **test2** | **test3** | **test4** | **test5** |
| 1 | Bill | Smith | 15 | 14 | 12 | 14 | 14 |
| 4 | Billy | Smith | 10 | 11 | 12 | 12 | 14 |

UPDATE Table4

SET buyingPrice = buyingPrice - 2

WHERE productName LIKE "%e";

UPDATE Table4

SET sellingPrice = buyingPrice \* 1.4

WHERE productName LIKE "%e";

SELECT \*

FROM Table4

WHERE productName LIKE "%e";

|  |  |  |  |
| --- | --- | --- | --- |
| **productID** | **productName** | **buyingPrice** | **sellingPrice** |
| 3 | Yorkie | 30 | 42 |

UPDATE Table5

SET priceUK = ROUND(priceUK \* 1.12, 2)

WHERE productName LIKE "%a\_ %";

SELECT \*

FROM Table5

WHERE productName LIKE "%a\_ %";

|  |  |  |
| --- | --- | --- |
| **productName** | **productID** | **priceUK** |
| Optical Mouse | 017 | 8.4 |
| Flat Screen Monitor | 102 | 84.0 |

UPDATE Table6

SET pricePerKilo = ROUND(pricePerKilo \* 0.9, 2),

numberOfKilos = numberOfKilos \* 2

WHERE fishType LIKE "\_o\_\_%";

SELECT \*

FROM Table6

WHERE fishType LIKE "\_o\_\_%";

|  |  |  |
| --- | --- | --- |
| **fishType** | **pricePerKilo** | **numberOfKilos** |
| Coley | 6.3 | 2.0 |
| Monkfish | 14.4 | 5.0 |

1. Implementation: Computed Fields and Aliases – Done

SELECT title, salesInternational - moviebudget AS Profit

FROM Movie, BoxOffice

WHERE Movie.movieID = BoxOffice.movieID

AND salesInternational > moviebudget

ORDER BY Profit DESC;

|  |  |
| --- | --- |
| **title** | **Profit** |
| Toy Story 3 | 148167031 |
| Up | 38338580 |
| Monsters Inc. | 22900000 |
| WALL-E | 22503696 |
| Brave | 1200000 |

SELECT director, salesUS + salesInternational AS Income

FROM Movie, BoxOffice

WHERE Movie.movieID = BoxOffice.movieID

ORDER BY director ASC, Income ASC;

|  |  |
| --- | --- |
| **director** | **Income** |
| Andrew Stanton | 521311860 |
| Andrew Stanton | 936743261 |
| Brad Bird | 623722818 |
| Brad Bird | 631442092 |
| Brenda Chapman | 538983207 |

SELECT title, yearReleased, rating,

salesInternational / 1000000 AS [Int Sales ($M)]

FROM Movie, BoxOffice

WHERE Movie.movieID = BoxOffice.movieID

AND salesInternational > 150000000

AND rating >= 8;

|  |  |  |  |
| --- | --- | --- | --- |
| **title** | **yearReleased** | **rating** | **Int Sales ($M)** |
| Toy Story | 1995 | 8.3 | 170 |
| Monsters Inc. | 2001 | 8.1 | 272 |
| Finding Nemo | 2003 | 8.2 | 555 |
| The Incredibles | 2004 | 8.0 | 370 |
| Ratatouille | 2007 | 8.0 | 417 |

SELECT title, durationsMins/60 AS Hours, durationsMins%60 AS Minutes

FROM Movie

ORDER BY durationsMins DESC;

|  |  |  |
| --- | --- | --- |
| **title** | **Hours** | **Minutes** |
| Cars 2 | 2 | 0 |
| Cars | 1 | 57 |
| The Incredibles | 1 | 56 |
| Ratatouille | 1 | 55 |
| Monsters University | 1 | 50 |

SELECT title, director, CAST(rating\*10 AS INTEGER) AS [Rating %]

FROM Movie, BoxOffice

WHERE Movie.movieID = BoxOffice.movieID

ORDER BY [Rating %] DESC,

title ASC;

|  |  |  |
| --- | --- | --- |
| **title** | **director** | **Rating %** |
| WALL-E | Andrew Stanton | 85 |
| Toy Story 3 | Lee Unkrich | 84 |
| Toy Story | John Lasseter | 83 |
| Up | Pete Docter | 83 |
| Finding Nemo | Andrew Stanton | 82 |

NB. CAST is not part of H CS specification. If not used, Rating % will be a decimal value.

1. Implementation: Computed Fields and Aliases – Done

SELECT City.name AS City, City.population / 1000 AS [Population (1000s)]

FROM Country, City

WHERE Country.code = City.countryCode

AND Country.name = "Ethiopia";

|  |  |
| --- | --- |
| **City** | **Population (1000s)** |
| Addis Ababa | 2316 |
| Dire Dawa | 194 |
| Gonder | 166 |
| Nazret | 147 |
| Harer | 122 |

SELECT name AS Country, code,

population / area AS [Pop Density]

FROM Country

WHERE name LIKE "M%"

ORDER BY [Pop Density] DESC;

|  |  |  |
| --- | --- | --- |
| **Country** | **code** | **Pop Density** |
| Monaco | MC | 31719 |
| Macau | MACX | 31052 |
| Malta | M | 1173 |
| Maldives | MV | 902 |
| Mayotte | MAYO | 598 |

SELECT capital, area \* 0.39 [Area (Sq miles)]

FROM Country, City

WHERE Country.code = City.countryCode

AND capital = City.name

AND City.population > 2000000

AND area \* 0.39 > 500000

ORDER BY [Area (Sq miles)] ASC;

|  |  |
| --- | --- |
| **capital** | **Area (Sq miles)** |
| Lima | 501235.8 |
| Tehran | 642720.0 |
| Jakarta | 748581.6 |
| Mexico City | 769294.5 |
| Kinshasa | 914709.9 |

SELECT capital, Country.name AS Country,

CAST(ROUND(longitude / 15, 0) AS INTEGER) AS [Time Diff]

FROM Country, City

WHERE Country.code = City.countryCode

AND capital = City.name

AND longitude > 0

ORDER BY longitude DESC,

Country.name ASC;

|  |  |  |
| --- | --- | --- |
| **capital** | **Country** | **Time Diff** |
| Suva | Fiji | 12 |
| Funafuti | Tuvalu | 12 |
| Wellington | New Zealand | 12 |
| Tarawa | Kiribati | 12 |
| Majuro | Marshall Islands | 11 |

SELECT capital,

CAST(ROUND((latitude-51.5) \* 113, 0) AS INTEGER) AS [LAT Distance]

FROM Country, City

WHERE Country.code = City.countryCode

AND capital = City.name

AND longitude >= -6.5

AND longitude <= 30

AND latitude >= 51.5

AND latitude <= 61

ORDER BY latitude DESC;

|  |  |
| --- | --- |
| **capital** | **LAT Distance** |
| Helsinki | 979 |
| Oslo | 953 |
| Stockholm | 887 |
| Tallinn | 881 |
| Riga | 622 |

1. Implementation: Computed Fields and Aliases - Done

SELECT title, hotelName, kmFromAirport,

kmFromAirport - 2.8 AS [New Distance]

FROM Hotel, Holiday

WHERE Hotel.hotelRef = Holiday.hotelRef

AND city = "Madrid";

|  |  |  |  |
| --- | --- | --- | --- |
| **title** | **hotelName** | **kmFromAirport** | **New Distance** |
| Capital Spain | Empera | 14.1 | 11.3 |
| Glorious Spain | Hotel Tranquilo | 15.6 | 12.8 |
| Madrid Attractions | Hotel Tranquilo | 15.6 | 12.8 |
| Madrid City Break | Hotel Tranquilo | 15.6 | 12.8 |
| Madrid Summer Madness | Empera | 14.1 | 11.3 |

SELECT hotelName, pricePerNight,

pricePerNight + 7.25 AS [ppn + Tax (£)],

ROUND((pricePerNight + 7.25) \* 1.13, 2) AS [Euros]

FROM Hotel

WHERE city = "Rome"

ORDER BY pricePerNight DESC;

|  |  |  |  |
| --- | --- | --- | --- |
| **hotelName** | **pricePerNight** | **ppn + Tax (£)** | **Euros** |
| Hotel Vicori | 85.0 | 92.25 | 104.24 |
| Hotel Sereno | 82.0 | 89.25 | 100.85 |
| Hotel Fario | 80.0 | 87.25 | 98.59 |
| Hotel Geno | 78.0 | 85.25 | 96.33 |
| Darios | 65.0 | 72.25 | 81.64 |

SELECT title, hotelName, nights, pricePerNight,

nights + 1 AS [New nights],

pricePerNight \* 1.1 AS [New ppn]

FROM Hotel, Holiday

WHERE Hotel.hotelRef = Holiday.hotelRef

AND title LIKE "%budget%";

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **title** | **hotelName** | **nights** | **pricePerNight** | **New nights** | **New ppn** |
| Budget Dublin | Dublin Apartments | 2 | 29.0 | 3 | 31.9 |
| Catalonia on a budget | Barce Bunkhouse | 3 | 25.0 | 4 | 27.5 |
| London on a budget | Sleepy Inn | 3 | 39.0 | 4 | 42.9 |
| Paris on a budget | Napolean Ville | 3 | 48.0 | 4 | 52.8 |
| Portugal on a budget | Lado Colina | 2 | 55.0 | 3 | 60.5 |

SELECT title, hotelName, nights, pricePerNight,

nights \* pricePerNight AS [Cost (£)]

FROM Hotel, Holiday

WHERE Hotel.hotelRef = Holiday.hotelRef

AND country = "Spain"

ORDER BY [Cost (£)] DESC, title ASC;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **title** | **hotelName** | **nights** | **pricePerNight** | **Cost (£)** |
| Glorious Spain | Hotel Tranquilo | 5 | 95.0 | 475.0 |
| Madrid Attractions | Hotel Tranquilo | 5 | 95.0 | 475.0 |
| Spanish Beach | Villa Grande | 5 | 95.0 | 475.0 |
| Spanish Siesta | Hotel Tranquilo | 5 | 95.0 | 475.0 |
| Spanish Sun | Villa Grande | 5 | 95.0 | 475.0 |

SELECT title, hotelName, country, city, starRating,

nights \* pricePerNight AS [Cost (£)]

FROM Hotel, Holiday

WHERE Hotel.hotelRef = Holiday.hotelRef

AND starRating >= 4

AND nights \* pricePerNight < 200;

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **title** | **hotelName** | **country** | **city** | **starRating** | **Cost (£)** |
| Edinburgh City Break | Castle View | Scotland | Edinburgh | 4 | 130.0 |
| Madrid City Break | Hotel Tranquilo | Spain | Madrid | 5 | 190.0 |

1. Implementation: Computed Fields and Aliases - Done

SELECT Product.productID, name, price, quantity,

price \* quantity AS Cost

FROM OrderProduct, Product

WHERE OrderProduct.productID = Product.productID

AND orderNumber = 20006

ORDER BY Cost DESC;

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **productID** | **name** | **price** | **quantity** | **Cost** |
| BR03 | 18 inch teddy bear | 11.99 | 10 | 119.9 |
| BR01 | 8 inch teddy bear | 5.99 | 20 | 119.8 |
| BR02 | 12 inch teddy bear | 8.99 | 10 | 89.9 |

SELECT Product.name, price,

ROUND(price \* 0.9, 2) AS Discounted

FROM Supplier, Product

WHERE Supplier.supplierID = Product.supplierID

AND Supplier.name = "Fun and Games";

|  |  |  |
| --- | --- | --- |
| **name** | **price** | **Discounted** |
| King doll | 9.49 | 8.54 |
| Queen doll | 9.49 | 8.54 |

SELECT Product.productID, Product.name,

quantity, quantity \* price AS Cost

FROM OrderProduct, Product, Supplier

WHERE OrderProduct.productID = Product.productID

AND Product.supplierID = Supplier.supplierID

AND orderNumber = 20007

AND Supplier.name = "Dolls House"

ORDER BY Cost ASC,

Product.name ASC;

|  |  |  |  |
| --- | --- | --- | --- |
| **productID** | **name** | **quantity** | **Cost** |
| RGAN01 | Raggedy Ann | 50 | 249.5 |
| BNBG02 | Bird bean bag toy | 100 | 349.0 |
| BNBG01 | Fish bean bag toy | 100 | 349.0 |
| BNBG03 | Rabbit bean bag toy | 100 | 349.0 |

SELECT description, price, quantity, price \* quantity AS [Cost],

quantity + 25 [Updated Qty],

price \* (quantity + 25) AS [Updated Cost]

FROM Customer, CustomerOrder, OrderProduct, Product

WHERE Customer.customerID = CustomerOrder.customerID

AND CustomerOrder.orderNumber = OrderProduct.orderNumber

AND OrderProduct.productID = Product.productID

AND Customer.name = "The Toy Store"

AND Product.name LIKE "%bean bag%";

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **description** | **price** | **quantity** | **Cost** | **Updated Qty** | **Updated Cost** |
| Fish bean bag toy | 5.49 | 10 | 54.9 | 35 | 192.15 |
| Bird bean bag toy | 4.49 | 10 | 44.9 | 35 | 157.15 |
| Rabbit bean bag toy | 3.49 | 10 | 34.9 | 35 | 122.15 |

1. Implementation: Aggregate Functions – Done

SELECT COUNT(\*) AS Employees

FROM Employee;

|  |
| --- |
| **Employees** |
| 13 |

SELECT MIN(yearsEmployeed) AS [Max Years],

MAX(yearsEmployeed) AS [Min Years]

FROM Employee;

|  |  |
| --- | --- |
| **Max Years** | **Min Years** |
| 1 | 9 |

SELECT AVG(yearsEmployeed) AS [Avg Years]

FROM Employee;

|  |
| --- |
| **Avg Years** |
| 5.0 |

SELECT jobTitle, AVG(yearsEmployeed) AS [Avg Years]

FROM Employee

GROUP BY jobTitle;

|  |  |
| --- | --- |
| **jobTitle** | **Avg Years** |
| Admin | 6.0 |
| Engineer | 3.4 |
| Manager | 6.0 |

SELECT building, COUNT(\*) AS [Employees]

FROM Employee

GROUP BY building;

|  |  |
| --- | --- |
| **building** | **Employees** |
| 1e | 7 |
| 2w | 6 |

SELECT building, SUM(yearsEmployeed) AS Years

FROM Employee

GROUP BY building;

|  |  |
| --- | --- |
| **building** | **Years** |
| 1e | 29 |
| 2w | 36 |

SELECT building, MIN(yearsEmployeed) \* 100 AS Bonus

FROM Employee

GROUP BY building;

|  |  |
| --- | --- |
| **building** | **Bonus** |
| 1e | 100 |
| 2w | 200 |

SELECT jobTitle, MAX(yearsEmployeed) \* 0.5 AS [Extra leave]

FROM Employee

GROUP BY jobTitle;

|  |  |
| --- | --- |
| **jobTitle** | **Extra leave** |
| Admin | 4.0 |
| Engineer | 3.0 |
| Manager | 4.5 |

**NB:** MAX(yearsEmployeed) / 2 will give an integer result.

SELECT building, jobTitle, COUNT(\*) AS Employees

FROM Employee

GROUP BY building, jobTitle;

|  |  |  |
| --- | --- | --- |
| **building** | **jobTitle** | **Employees** |
| 1e | Engineer | 5 |
| 1e | Manager | 2 |
| 2w | Admin | 5 |
| 2w | Manager | 1 |

1. Implementation: Aggregate Functions – Done

SELECT category, ROUND(AVG(price), 2) AS [Avg Price]

FROM Plant

GROUP BY category;

|  |  |
| --- | --- |
| **category** | **Avg Price** |
| Climber | 9.04 |
| Fruit | 16.19 |
| Fuchsia | 6.63 |
| Hedge | 62.8 |
| Patio | 9.92 |

SELECT code, COUNT(\*) AS Count

FROM Plant

GROUP BY code

ORDER BY Count DESC,

code ASC;

|  |  |
| --- | --- |
| **code** | **Count** |
| P | 8 |
| B | 7 |
| G | 6 |
| Y | 6 |
| R | 5 |

SELECT height, MAX(price) - MIN(price) AS Range

FROM Plant

GROUP BY height;

|  |  |
| --- | --- |
| **height** | **Range** |
| M | 67.0 |
| S | 36.7 |
| T | 144.0 |

SELECT COUNT(\*) AS Count, SUM(price) AS [Total (£)]

FROM Plant

WHERE name LIKE "%p%";

|  |  |
| --- | --- |
| **Count** | **Total (£)** |
| 11 | 162.35 |

SELECT MAX(unit) AS Max, ROUND(AVG(unit), 1) AS Avg

FROM Plant

WHERE referenceID LIKE "B%";

|  |  |
| --- | --- |
| **Max** | **Avg** |
| 12 | 3.8 |

SELECT unit, COUNT(\*) AS Count, MIN(price) AS Cheapest

FROM Plant

GROUP BY unit

ORDER BY Count DESC,

unit ASC;

|  |  |  |
| --- | --- | --- |
| **unit** | **Count** | **Cheapest** |
| 1 | 34 | 3.0 |
| 6 | 5 | 6.25 |
| 3 | 4 | 6.0 |
| 5 | 3 | 8.25 |
| 10 | 2 | 50.0 |

SELECT category, height, ROUND(AVG(price), 2) AS [Avg (£)]

FROM Plant

GROUP BY category, height;

|  |  |  |
| --- | --- | --- |
| **category** | **height** | **Avg (£)** |
| Climber | M | 8.25 |
| Climber | T | 9.2 |
| Fruit | M | 16.0 |
| Fruit | T | 16.22 |
| Fuchsia | M | 6.6 |

1. Implementation: Aggregate Functions – Done

SELECT MAX(area) AS Maximum, MIN(area) AS Minimum,

ROUND(AVG(area), 1) AS [Average (1 dp)]

FROM Country;

|  |  |  |
| --- | --- | --- |
| **Maximum** | **Minimum** | **Average (1 dp)** |
| 17075200 | 0.44 | 569693.5 |

SELECT COUNT(\*) AS [1,000,000+]

FROM Country

WHERE area > 1000000;

|  |
| --- |
| **1,000,000+** |
| 30 |

SELECT SUM(area)

FROM Country

WHERE name LIKE "D%";

|  |
| --- |
| **Combined Area** |
| 114546 |

SELECT SUM(City.population) AS [German Cities]

FROM Country, City

WHERE Country.code = City.countryCode

AND Country.name= "Germany";

|  |
| --- |
| **German Cities** |
| 25973481 |

SELECT code, SUM(City.population) AS [City Population]

FROM Country, City

WHERE Country.code = City.countryCode

AND code LIKE "\_A"

GROUP BY code;

|  |  |
| --- | --- |
| **code** | **City Population** |
| JA | 101000 |
| MA | 10005511 |
| NA | 125000 |
| PA | 1259200 |
| RA | 16933516 |

SELECT SUM(City.population) AS [Total Population]

FROM Country, City

WHERE Country.code = City.countryCode

AND Country.capital = City.name

AND Country.name LIKE "%AM%";

|  |
| --- |
| **Total Population** |
| 7141899 |

SELECT Country.name, SUM(City.population)

FROM Country, City

WHERE Country.code = City.countryCode

AND Country.name LIKE "% %"

AND Country.population > 11000000

GROUP BY Country.name;

|  |  |
| --- | --- |
| **name** | **SUM(City.population)** |
| Cote dIvoire | 200659 |
| North Korea | 6735000 |
| Saudi Arabia | 1250000 |
| South Africa | 4750288 |
| South Korea | 26978665 |

SELECT Country.name AS Country, COUNT(\*) AS Cities

FROM Country, city

WHERE Country.code = City. countryCode

AND Country.name LIKE "%i%a"

GROUP BY Country.name

ORDER BY Cities DESC,

Country.name;

|  |  |
| --- | --- |
| **Country** | **Cities** |
| China | 308 |
| Russia | 175 |
| India | 98 |
| Nigeria | 56 |
| Colombia | 49 |

1. Implementation: Aggregate Functions

|  |  |
| --- | --- |
| **1.** | SELECT Supplier.name AS Supplier, round(AVG(Price), 2) AS [Average]  FROM Supplier, Product  WHERE Supplier.supplierID = Product.supplierID  GROUP BY Supplier.name; |
|  |  |
|  |  |
| **2.** | SELECT shopName, MAX(quantity) AS Maximum, round(AVG(quantity), 1) AS Average  FROM Customer, CustomerOrder, OrderProduct  WHERE (Customer.customerID = CustomerOrder.CustomerID) AND (CustomerOrder.orderNumber = OrderProduct.orderNumber)  GROUP BY shopName; |
|  |  |
|  |  |
| **3.** | SELECT shopName, COUNT(\*) AS Orders  FROM Customer, CustomerOrder  WHERE (Customer.customerID = CustomerOrder.CustomerID)  GROUP BY shopName  ORDER BY COUNT(\*) DESC, shopName ASC; |
|  |  |
|  |  |
| **4.** | SELECT CustomerOrder.orderNumber, COUNT(\*) AS Items  FROM CustomerOrder, OrderProduct  WHERE (CustomerOrder.orderNumber = OrderProduct.orderNumber)  GROUP BY CustomerOrder.orderNumber; |
|  |  |
|  |  |
| **5.** | SELECT Supplier.name, SUM(Price) AS [Total Value]  FROM Supplier, Product  WHERE Supplier.supplierID = Product.supplierID  GROUP BY Supplier.name  ORDER BY SUM(Price) DESC; |
|  |  |
|  |  |
| **6.** | SELECT name, SUM(quantity) AS Ordered  FROM OrderProduct, Product  WHERE OrderProduct.productID = Product.productID  GROUP BY name  ORDER BY SUM(quantity) DESC , name; |
|  |  |
|  |  |
| **7.** | SELECT shopName, SUM(quantity) AS Bears  FROM Customer, CustomerOrder, OrderProduct, Product  WHERE (Customer.customerID = CustomerOrder.customerID) AND (CustomerOrder.orderNumber = OrderProduct.orderNumber) AND (OrderProduct.productID = Product.productID) AND (name like "\*bear\*")  GROUP BY shopName  ORDER BY SUM(quantity) ASC; |
|  |  |
|  |  |
| **8.** | SELECT shopName, CustomerOrder.orderNumber, orderDate, SUM(quantity \* Price) AS [Value]  FROM Customer, CustomerOrder, OrderProduct, Product  WHERE (Customer.customerID = CustomerOrder.customerID) AND (CustomerOrder.orderNumber = OrderProduct.orderNumber) AND (OrderProduct.productID = Product.productID) AND (name like "\*bear\*")  GROUP BY Customer.shopName, CustomerOrder.orderNumber, orderDate  ORDER BY shopName ASC, orderDate DESC; |
|  |  |

1. Implementation: Problem Solving

## Part 1

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Customer** | places | **CustOrder** | is for | **OrderItem** | lists | **Item** |
|  |  |  |

## Part 2

|  |  |
| --- | --- |
| A = | Y |
| B = | Restricted Choice (one of Boys Toys, Girls Gizmos, Personal Grooming, Office Distractions or Computer Accessories) |
| C = | PK |
| D = | Existing customerID from Customer entity |
| E = | PK / FK |
| F = | Number |

## Task 4

|  |  |  |  |
| --- | --- | --- | --- |
| **1.** | SELECT \*  FROM Item  WHERE category="Boys Toys"; | | |
|  |  | | |
|  |  | | |
| **2.** | SELECT OrderItem.orderNo, Item.description, Item.price  FROM OrderItem, Item  WHERE OrderItem.itemID=Item.itemID  ORDER BY OrderItem.orderNo; | | |
|  |  | *The answer table for this query has 153 records. This screen shot shows the first 20 of them.* | |
|  |  | | |
| **3.** | SELECT forename, surname, orderDate  FROM Customer, CustOrder  WHERE Customer.customerID=CustOrder.customerID  AND surname LIKE "\*em\*"  ORDER BY surname, forename; | | |
|  |  | | |
| **4.** | SELECT CustOrder.orderNo, orderDate, description, quantity, price, quantity\*price AS Cost  FROM CustOrder, Item, OrderItem  WHERE CustOrder.orderNo=OrderItem.orderNo  AND OrderItem.itemID=Item.itemID  ORDER BY orderDate; | | |
|  |  | | *The answer table for this query has 153 records. This screen shot shows the first 20 of them.* |
| **5.** | SELECT OrderItem.orderNo, description, quantity, price, price\*0.05 AS Discount, price\*0.95 AS DiscountedPrice  FROM OrderItem, Item  WHERE OrderItem.itemID=Item.itemID  AND quantity>=4; | | |
|  |  | | |
|  |  | | |
| **6.** | SELECT CustOrder.orderNo, orderDate, description, quantity, price, quantity\*price AS [Total Item Price]  FROM CustOrder, OrderItem, Item, Customer  WHERE CustOrder.orderNo = OrderItem.orderNo  AND OrderItem.itemID=item.itemID  AND CustOrder.customerID=Customer.customerID  AND forename="Mari"  AND surname ="Singer"  ORDER BY orderDate; | | |
|  |  | | |
|  |  | | |
| **7.** | SELECT category, COUNT(itemID) AS [Number of Items]  FROM Item  GROUP BY category  ORDER BY COUNT(itemID) DESC; | | |
|  |  | | |
|  |  | | |
| **8.** | SELECT OrderItem.orderNo, orderDate, SUM(quantity\*price) AS [Total Order Cost]  FROM CustOrder, OrderItem, Item  WHERE CustOrder.orderNo=OrderItem.orderNo  AND OrderItem.itemID=Item.itemID  AND OrderDate LIKE "\*/01/2008"  GROUP BY OrderItem.orderNo, orderdate  ORDER BY orderDate; | | |
|  |  | | |
|  |  | | |
| **9.** | SELECT forename, surname, customerEmail  FROM Customer  WHERE customerEmail LIKE "\*MobileLife\*"; | | |
|  |  | | |
| **10.** | SELECT category, count(\*) AS [Number of Orders], SUM(quantity) AS [Number of Items Ordered]  FROM OrderItem, Item  WHERE OrderItem.itemID=Item.itemID  AND category="Office Distractions"  GROUP BY category; | | |
|  |  | | |
|  |  | | |
| **11.** | SELECT category, round(AVG(price)) AS [Average Price]  FROM Item  GROUP BY category; | | |
|  |  | | |
|  |  | | |
| **12.** | SELECT forename, surname, COUNT(\*) AS [Number of ItemsOrdered]  FROM Customer, CustOrder, OrderItem  WHERE Customer.customerID=CustOrder.customerID  AND CustOrder.orderNo=OrderItem.orderNo  AND surname LIKE "\*i?g?\*"  GROUP BY surname, forename; | | |
|  |  | | |
|  |  | | |
| **13.** | SELECT Customer.customerID, postcode, COUNT(\*) AS [Number of Orders Placed]  FROM Customer, CustOrder  WHERE Customer.customerID=CustOrder.customerID  AND orderDate LIKE "\*/\*/2008"  GROUP BY Customer.customerID, postcode  ORDER BY COUNT(\*) DESC , postcode; | | |
|  |  | | |
|  |  | | |
| **14.** | SELECT CustOrder.orderNo, orderDate, SUM(quantity\*price) AS [Before Discount], [Before Discount]\*0.05 AS [5% Discount], [Before Discount]\*0.95 AS [Discounted Total]  FROM CustOrder, OrderItem, Item  WHERE CustOrder.orderNo=OrderItem.orderNo  AND OrderItem.itemID=Item.itemID  AND orderDate LIKE "\*/12/2007"  GROUP BY CustOrder.orderNo, orderDate  ORDER BY orderDate, CustOrder.orderNo; | | |
|  |  | | |

## Completed Data Dictionary

*\*\* Only to be issued after completion and submission of Task 2*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Customer | customerID | PK | Number |  | Y | >=0001 and <=9999 |
|  | forename |  | Text | 10 | Y |  |
|  | Surname |  | Text | 15 | Y |  |
|  | address1 |  | Text | 20 | Y |  |
|  | address2 |  | Text | 20 | N |  |
|  | address3 |  | Text | 20 | Y |  |
|  | postcode |  | Text | 8 | Y |  |
|  | customerEmail |  | Text | 25 | **Y** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Item | itemID | PK | Number |  | Y |  |
|  | description |  | Text | 50 | Y |  |
|  | category |  | Text | 20 | Y | **Restricted Choice (one of Boys Toys, Girls Gizmos, Personal Grooming, Office Distractions or Computer Accessories)** |
|  | price |  | Number |  | Y | < 50 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| CustOrder | orderNo | **PK** | Number |  | Y |  |
|  | customerID | FK | Number |  | Y | **Existing customerID from Customer entity** |
|  | orderDate |  | Date |  | Y |  |
|  | orderDispatched |  | Date |  | N |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| OrderLine | orderNo | **PK/FK** | Number |  | Y | Existing orderNo from Order entity |
|  | itemID | PK/FK | **Number** |  | Y | Existing itemID from Item entity |
|  | quantity |  | Number |  | Y |  |

1. Implementation: Problem Solving

**Exercise 10 – Answers**

**Task 1**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **TutorGroup** | has | **Student** | receives | **Result** | is for | **Assessment** |
|  |  |  |

**Task 2**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Result | studentNo | **PK/FK** | **Number** | **N** | Y | **Existing studentNo from Student table** |
| assessmentCode | **PK/FK** | **Text (6)** | **N** | Y | **Existing assessmentCode from Assessment table** |
| mark | **PK** | **Number** | **N** | Y | **>=0 and <=100** |
| assessmentDate |  | Date |  | Y | >=15/08/13 and <=12/05/14 |

**Task 4**

|  |  |  |
| --- | --- | --- |
| **1.** | SELECT forename, surname, day  FROM Student, TutorGroup  WHERE Student.tgCode=TutorGroup.tgCode  AND day="Monday"; | |
|  |  | |
|  |  | |
| **2.** | SELECT Student.studentNo, mark, title  FROM Student, Assessment, Result  WHERE Student.studentNo=Result.studentNo  AND Result.assessmentCode=Assessment.assessmentCode  AND mark>90  ORDER BY mark DESC , title; | |
|  |  | |
|  |  | |
| **3.** | SELECT forename, surname, assessmentDate, tutor, mark  FROM Student, TutorGroup, Result  WHERE Student.tgCode=TutorGroup.tgCode  AND Student.studentNo=Result.studentNo  AND Result.assessmentDate LIKE "\*/03/2014"  ORDER BY TutorGroup.tutor, mark DESC; | |
|  |  | *The answer table for this query has 30 records. This screen shot shows the first 24 of them.* |
|  |  | |
| **4.** | SELECT studentNo, title, (mark-passMark) AS Difference  FROM Result, Assessment  WHERE Result.assessmentCode=Assessment.assessmentCode  AND (mark-passMark)<=-5; | |
|  |  | |
|  |  | |
| **5.** | SELECT title, passMark AS [Original Pass Mark], (passMark\*0.95) AS [Adjusted Pass Mark]  FROM Assessment  WHERE writer LIKE '\*Mensinger\*'; | |
|  |  | |
|  |  | |
| **6.** | SELECT TutorGroup.tgCode, tutor, COUNT(\*) AS [Number of Assessments]  FROM TutorGroup, Student, Result  WHERE TutorGroup.tgCode=Student.tgCode  AND Student.studentNo=Result.studentNo  GROUP BY TutorGroup.tgCode, tutor  ORDER BY COUNT(\*) DESC , TutorGroup.tgCode; | |
|  |  | |
|  |  | |
| **7.** | SELECT SUM(duration) AS [Total Assessment Time], assessmentType  FROM Assessment  GROUP BY assessmentType  ORDER BY SUM(duration) DESC , assessmentType; | |
|  |  | |
| **8.** | SELECT TutorGroup.tgCode, tutor, MIN(mark) AS [Minimum Mark], MAX(mark) AS [Maximum Mark], round(AVG(mark),1) AS [Average Mark]  FROM TutorGroup, Student, Result  WHERE TutorGroup.tgCode=Student.tgCode  AND Student.studentNo=Result.studentNo  GROUP BY TutorGroup.tgCode, tutor; | |
|  |  | |
|  |  | |
| **9.** | SELECT title, passMark, COUNT(\*) AS [Number of Students Passed]  FROM Assessment, Result  WHERE Assessment.assessmentCode=Result.assessmentCode  AND mark > passMark  GROUP BY title, passMark; | |
|  |  | |
|  |  | |
| **10.** | SELECT room, tutor, COUNT(\*) AS [Number of Students]  FROM TutorGroup, Student  WHERE TutorGroup.tgCode=Student.tgCode  AND room LIKE "\*1\*"  GROUP BY tutor, room; | |
|  |  | |
|  |  | |
| **11.** | SELECT tutor, room, COUNT(\*) AS [Number of Students]  FROM TutorGroup, Student  WHERE TutorGroup.tgCode=Student.tgCode  AND room LIKE "\*1\*"  GROUP BY tutor, room; | |
|  |  | |
|  |  | |
| **12.** | SELECT forename, surname, SUM(mark) AS [Total Mark], round([Total Mark]/7) AS [Overall Percentage]  FROM Student, Result, Assessment  WHERE Student.studentNo=Result.studentNo  AND Result.assessmentCode=Assessment.assessmentCode  AND surname LIKE "\*ll\*"  GROUP BY surname, forename; | |
|  |  | |
|  |  | |
| **13.** | SELECT Result.assessmentCode, title, MAX(mark) AS [Highest Mark], MIN(mark) AS [Lowest Mark], round(AVG(mark)) AS [Average Mark]  FROM Result, Assessment  WHERE Result.assessmentCode=Assessment.assessmentCode  AND Result.assessmentCode LIKE "3??"  GROUP BY Result.assessmentCode, title; | |
|  |  | |
|  |  | |
| **14.** | UPDATE Student SET address = "71 Burnside Road", town = "Gourock", postcode = "PA19 1UX"  WHERE forename="Aidan"  AND surname ="Clark"; | |
|  | The updated record is shown below. | |

## Completed Data Dictionary

*\*\* Only to be issued after completion and submission of Task 2*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Req’d** | **Validation** |
| Student | studentNo | PK | Number |  | Y | >=100000 and <=199999 |
| forename |  | Text (20) | 20 | Y |  |
| surname |  | Text (30) | 30 | Y |  |
| address |  | Text (30) | 30 | Y |  |
| town |  | Text (15) | 15 | Y |  |
| postcode |  | Text (8) | 8 | Y |  |
| tgCode | FK | Text (5) | 5 | Y | Existing tgCode from TutorGroup table |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Req’d** | **Validation** |
| TutorGroup | tgCode | PK | Text (5) | 5 | Y |  |
| tutor |  | Text (15) | 15 | Y |  |
| room |  | Text (5) | 5 | Y |  |
| day |  | Text (9) | 9 | Y | Restricted choice (one of Monday, Tuesday, Wednesday, Thursday or Friday) |
| time |  | Time |  | Y |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Assessment | assessmentCode | PK | Text (6) | 6 | Y |  |
| title |  | Text (30) | 30 | Y |  |
| assessmentType |  | Text (30) | 30 | Y | Restricted choice (one of Multiple Choice, Short Answer, Extended Response, or Practical) |
| duration |  | Number |  | Y | >=15 and <=150 |
| writer |  | Text (15) | 15 | Y |  |
| passMark |  | Number |  | Y | >=0 and <=100 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Entity** | **Attribute** | **Key** | **Type** | **Size** | **Required** | **Validation** |
| Result | studentNo | **PK/FK** | **Number** | **N** | Y | **Existing studentNo from Student table** |
| assessmentCode | **PK/FK** | **Text (6)** | **N** | Y | **Existing assessmentCode from Assessment table** |
| mark | **PK** | **Number** | **N** | Y | **>=0 and <=100** |
| assessmentDate |  | Date |  | Y | >=15/08/13 and <=12/05/14 |

1. Implementation: Queries that use another query – Done

CREATE TEMP VIEW Population (biggest) AS

SELECT MAX(population)

FROM Country;

SELECT name, capital, population

FROM Country, Population

WHERE population = biggest;

|  |  |  |
| --- | --- | --- |
| **name** | **capital** | **population** |
| China | Beijing | 1210004956 |

CREATE TEMP VIEW Population (average) AS

SELECT AVG(population)

FROM City;

SELECT City.name, Country.name, city.population

FROM City, Country, Population

WHERE Country.code = City.countryCode

AND City.population >= average + 5000000;

|  |  |  |
| --- | --- | --- |
| **name** | **name** | **population** |
| Moscow | Russia | 8717000 |
| Istanbul | Turkey | 7615500 |
| London | United Kingdom | 6967500 |
| Beijing | China | 7000000 |
| Shanghai | China | 7830000 |

CREATE TEMP VIEW Latitude (reykjavik) AS

SELECT latitude

FROM City

WHERE name = "Reykjavik";

SELECT City.name, Country.name, latitude

FROM Country, City, Latitude

WHERE Country.code = City.countryCode

AND latitude > reykjavik

ORDER BY latitude DESC;

|  |  |  |
| --- | --- | --- |
| **name** | **name** | **latitude** |
| Longyearbyen | Svalbard | 78.13 |
| Vadsoe | Norway | 70.5 |
| Hammerfest | Norway | 70.4 |
| Tromsoe | Norway | 69.4 |
| Narvik | Norway | 68.3 |

CREATE TEMP VIEW Bolivia (cityPops) AS

SELECT AVG(city.population)

FROM Country, City

WHERE Country.code = City.countryCode

AND Country.name = "Bolivia";

SELECT City.name, City.population

FROM Country, City, Bolivia

WHERE Country.code = City.countryCode

AND Country.name = "United Kingdom"

AND City.population > cityPops

ORDER BY City.population ASC;

|  |  |
| --- | --- |
| **name** | **population** |
| Liverpool | 474000 |
| Bradford | 481700 |
| Sheffield | 530100 |
| Glasgow | 674800 |
| Leeds | 724400 |

CREATE TEMP VIEW Area (largest) AS

SELECT MAX(area)

FROM Country;

SELECT COUNT(\*) AS [Small Countries]

FROM Country, Area

WHERE area < largest / 100;

|  |
| --- |
| **Small Countries** |
| 148 |

1. Implementation: Queries that use another query – Part

CREATE TEMP VIEW furthest (km) AS

SELECT MAX(kmFromAirport)

FROM Hotel;

SELECT Hotel.name, destination, country, kmFromAirport

FROM Hotel, Holiday, Furthest

WHERE Hotel.hotelRef = Holiday.hotelRef

AND kmFromAirport = km;

|  |  |  |  |
| --- | --- | --- | --- |
| **name** | **destination** | **country** | **kmFromAirport** |
| Hotel Vicori | Lazio | Italy | 34.5 |

CREATE TEMP VIEW Rating (average) AS

SELECT AVG(starRating)

FROM Hotel, Holiday

WHERE Hotel.hotelRef = Holiday.hotelRef

AND (Holiday.name LIKE "%Break%"

OR Holiday.name LIKE "%Package%");

SELECT Hotel.name, starRating

FROM Hotel, Rating

WHERE Hotel.starRating < average

ORDER BY starRating DESC,

name ASC;

|  |  |
| --- | --- |
| **name** | **starRating** |
| Avanti | 2 |
| Catalonia Inn | 2 |
| Eastern Hotel | 2 |
| Lado Colina | 2 |
| Napolean Ville | 2 |

CREATE TEMP VIEW PPN (dearest) AS

SELECT MAX(pricePerNight)

FROM Hotel

WHERE city = "Edinburgh";

SELECT name, city, pricePerNight

FROM Hotel, PPN

WHERE pricePerNight > dearest

ORDER BY pricePerNight DESC,

name ASC;

|  |  |  |
| --- | --- | --- |
| **name** | **city** | **pricePerNight** |
| Tower Turrets | London | 135.0 |
| Champs-Elysees Star Residence | Paris | 120.0 |
| St James House | London | 120.0 |
| Casa Luxor | Barcelona | 100.0 |
| Hotel Tranquilo | Madrid | 95.0 |

CREATE TEMP VIEW Star (rating) AS

SELECT starRating

FROM Hotel

WHERE name = "Der Wald";

SELECT COUNT(\*) AS [Number of Hotels]

FROM Hotel, Star

WHERE starRating = rating;

|  |
| --- |
| **Number of Hotels** |
| 14 |

|  |  |
| --- | --- |
| **5.** | Query5a  SELECT round(AVG(kilometresFromAirport), 1) AS Average  FROM Holiday, Hotel  WHERE (Holiday.hotelRef = Hotel.hotelRef) AND (Country = "Spain"); |
|  |  |
|  | Query5b  SELECT Title, City, kilometresFromAirport  FROM Holiday, Hotel, Query5a  WHERE (Holiday.hotelRef = Hotel.hotelRef) AND (kilometresFromAirport < Average) AND (city = "Lisbon"); |
|  |  |
| **6.** | Query6a  SELECT MAX(Nights) AS Maximum  FROM Holiday, Hotel  WHERE (Holiday.hotelRef = Hotel.hotelRef) AND (City like "?o\*"); |
|  |  |
|  | Query6b  SELECT Title, dateOfDeparture, Nights  FROM Holiday, Hotel, Query6a  WHERE (Holiday.hotelRef = Hotel.hotelRef) AND (Nights = Maximum)  ORDER BY dateOfDeparture DESC; |
|  |  |