4005CEm coursework

Database systems

Coventry University

Computer Science

Contents

[Introduction 2](#_Toc68528597)

[1NF table 3](#_Toc68528598)

[2NF table 4](#_Toc68528599)

[3NF table 6](#_Toc68528600)

[ER diagram 7](#_Toc68528601)

[SQL Creation code 7](#_Toc68528602)

[Queries: 9](#_Toc68528603)

[Conclusion 10](#_Toc68528604)

[**References** 10](#_Toc68528605)

# Introduction

# 

A database loosely speaking is a collection of data, further in you can refer to a database management system which is used to allow users to create and maintain a database, this helps since it means redundancy can be controlled and only authorised personnel have access to the data and allowing the data to be processed efficiently.

Table

Description automatically generated

The table shouldn’t be kept as a traditional database because it should be in a normalised form since when we design a database for an enterprise, the main objective is to create an accurate representation of the data, relationships between the data, and constrains on the data that is pertinent to the enterprise Connolly (2004, p.387). As it stands the database is not accurate in terms of showing the relationships between the data meaning we need to normalise in order to show the different attributes and the degree of those attributes within the table. We can clearly see the table possess repeat values, for many areas such as the rental ID, throughout meaning it does not satisfy the rule of first normal form, an attribute (column) of a table cannot hold multiple values. It should hold only atomic values Singh (2015). After that we ensure the redundant data across multiple rows is moved into different tables and we must make sure that the resulting tables must be related to each other using foreign keys such data would be the Film name as it can be dependent on a primary key as well as other parts of the table. After this, the table will be in 2NF meaning all that is left is to get it into 3NF, A relation is in third normal form (abbreviated 3NF) if it is in 2NF and none of its non-key attributes are transitively dependent upon any candidate key Demba (2013, p.46). Doing this will make sure to minimize any data redundancy making it easier to maintain data and overall provide a better database design which is more suitable for use when creating a database system.

# 1NF table

|  |  |  |  |
| --- | --- | --- | --- |
| Rental ID | Film Name | Quantity | Film Price |
| 2594 | Gamer | 1 | £3.2 |
| 2594 | Dark Waters | 2 | £2.6 |
| 2594 | Inception | 3 | £4.9 |
| 3412 | A Space Odyssey | 1 | £2.1 |
| 8972 | Dark Waters | 1 | £2.6 |
| 8972 | Taxi Driver | 2 | £1.9 |
| 8972 | The Dark Knight | 2 | £3.5 |
| 8972 | Forrest Gump | 1 | £3.9 |
| 5119 | Dark Waters | 1 | £2.6 |
| 5119 | Gamer | 3 | £3.2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rental ID | Customer ID | Customer Name | Customer Email | Rental Date | Total Price |
| 2594 | 456123 | Daniel Smith | [ds@gmail.com](mailto:ds@gmail.com) | 12/09/2019 | £23.1 |
| 3412 | 827162 | Roger Williams | [rw@gmail.com](mailto:rw@gmail.com) | 05/01/2020 | £2.1 |
| 8972 | 198256 | Jon Snow | [js@gmail.com](mailto:js@gmail.com) | 26/08/2020 | £17.3 |
| 5119 | 049821 | Angela Jones | aj@gmail.com | 18/03/2020 | £12.2 |

The table is in 1NF form above since it does not contain repeating attributes in this case Rental ID and the Film Name as well as Film price.

# 2NF table

|  |  |  |
| --- | --- | --- |
| Rental ID | Film Name | Quantity |
| 2594 | Gamer | 1 |
| 2594 | Dark Waters | 2 |
| 2594 | Inception | 3 |
| 3412 | A Space Odyssey | 1 |
| 8972 | Dark Waters | 1 |
| 8972 | Taxi Driver | 2 |
| 8972 | The Dark Knight | 2 |
| 8972 | Forrest Gump | 1 |
| 5119 | Dark Waters | 1 |
| 5119 | Gamer | 3 |

|  |  |
| --- | --- |
| Film Name | Film Price |
| Gamer | £3.2 |
| Dark Waters | £2.6 |
| Inception | £4.9 |
| A Space Odyssey | £2.1 |
| Taxi Driver | £1.9 |
| The Dark Knight | £3.5 |
| Forrest Gump | £3.9 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Rental ID | Customer ID | Customer Name | Customer Email | Rental Date | Total Price |
| 2594 | 456123 | Daniel Smith | [ds@gmail.com](mailto:ds@gmail.com) | 12/09/2019 | £23.1 |
| 3412 | 827162 | Roger Williams | [rw@gmail.com](mailto:rw@gmail.com) | 05/01/2020 | £2.1 |
| 8972 | 198256 | Jon Snow | [js@gmail.com](mailto:js@gmail.com) | 26/08/2020 | £17.3 |
| 5119 | 049821 | Angela Jones | aj@gmail.com | 18/03/2020 | £12.2 |

This is in 2NF as no attributes are dependent on only part of the primary key, here the film name was dependant on film price though it’s dependant on Rental ID but not the entire key.

# 3NF table

|  |  |  |
| --- | --- | --- |
| Customer ID | Customer Name | Customer Email |
| 456123 | Daniel Smith | [ds@gmail.com](mailto:ds@gmail.com) |
| 827162 | Roger Williams | [rw@gmail.com](mailto:rw@gmail.com) |
| 198256 | Jon Snow | [js@gmail.com](mailto:js@gmail.com) |
| 049821 | Angela Jones | aj@gmail.com |

|  |  |  |
| --- | --- | --- |
| Rental ID | Film Name | Quantity |
| 2594 | Gamer | 1 |
| 2594 | Dark Waters | 2 |
| 2594 | Inception | 3 |
| 3412 | A Space Odyssey | 1 |
| 8972 | Dark Waters | 1 |
| 8972 | Taxi Driver | 2 |
| 8972 | The Dark Knight | 2 |
| 8972 | Forrest Gump | 1 |
| 5119 | Dark Waters | 1 |
| 5119 | Gamer | 3 |

|  |  |
| --- | --- |
| Film Name | Film Price |
| Gamer | £3.2 |
| Dark Waters | £2.6 |
| Inception | £4.9 |
| A Space Odyssey | £2.1 |
| Taxi Driver | £1.9 |
| The Dark Knight | £3.5 |
| Forest Gump | £3.9 |

|  |  |  |  |
| --- | --- | --- | --- |
| Rental ID | Customer ID | Rental Date | Total Price |
| 2594 | 456123 | 12/09/2019 | £23.1 |
| 3412 | 827162 | 05/01/2020 | £2.1 |
| 8972 | 198256 | 26/08/2020 | £17.3 |
| 5119 | 049821 | 18/03/2020 | £12.2 |

This is in 3NF as all attributes are dependent on the primary key, Rental ID, and not dependant on any attribute that isn’t part of the primary key since here the customer ID and film name table both depend on rental ID and every other table possess rental ID.

# ER diagram

Diagram

Description automatically generated

In this ER diagram, we can see the entities are customers, rental order, film, film information. As you can see the attributes have been clearly labelled within the diagram and shows what key each one has. In the case of Film there are 2 primary keys since together Rental ID and Film name create a compound primary key.

# SQL Creation code

CREATE TABLE CustomerInformation (

[Customer ID] INT NOT NULL

PRIMARY KEY,

[Customer Name] VARCHAR (6) NOT NULL,

[Customer Email] VARCHAR (12) NOT NULL

);

INSERT INTO CustomerInformation VALUES (456123,'Daniel Smith','ds@gmail.com'),

(827162,'Roger Williams','rw@gmail.com'),

(198256,'Jon Snow','js@gmail.com'),

(049821,'Angela Jones','aj@gmail.com')

This SQL code creates the customer information table which stores information about customers, using VARCHAR allows the letters in the table to be used with appropriate sizes for the data. The primary key is Customer ID as it’s used in other tables making it the best choice to pull up information about customers.

CREATE TABLE CustomerRental (

[Rental ID] INTEGER PRIMARY KEY

NOT NULL,

[Customer ID] INT NOT NULL,

[Rental Date] DATE NOT NULL,

[Total Price £] INT NOT NULL,

FOREIGN KEY (

[Customer ID]

)

REFERENCES CustomerInformation ([Customer ID])

)

INSERT INTO CustomerRental VALUES (2594,456123,'12/09/2019',23.1),

(3412,827162,'05/01/2020',2.1),

(8972,198256,'26/08/2020',17.3),

(5119,049821,'18/03/2020',2.2);

This SQL code creates the Rental information table containing information on rental dates and total price, it uses [Rental ID] as the primary key referencing customers with the foreign key [Customer ID] ensuring the customer information table has a link to this table so it’s accessible.

CREATE TABLE FilmInformation (

[Film Name] VARCHAR (15) NOT NULL

PRIMARY KEY,

[Film Price £] INT NOT NULL

);

INSERT INTO FilmInformation VALUES ('Gamer',3.2),

('Dark Waters',2.6),

('Inception',4.9),

('A Space Odyssey',2.1),

('Taxi Driver',1.9),

('The Dark Knight',3.5),

('Forrest Gump',3.9);

This SQL code creates the table Film Information containing the information regarding film name and film price, this table contains Film name as the primary key since it’s unique.

CREATE TABLE FilmRental (

[Rental ID] INTEGER NOT NULL,

[Film Name] VARCHAR(15) NOT NULL,

Quantity INT NOT NULL,

FOREIGN KEY (

[Film Name]

)

REFERENCES FilmInformation ([Film Name]),

FOREIGN KEY (

[Rental ID]

)

REFERENCES CustomerRental ([Rental ID])

PRIMARY KEY ([Rental ID],[Film Name])

);

INSERT INTO FilmRental VALUES (2594,'Gamer',1),

(2594,'Dark Waters',2),

(2594,'Inception',3),

(3412,'A Space Odyssey',1),

(8972,'Dark Waters',1),

(8972,'Taxi Driver',2),

(8972,'The Dark Knight',2),

(8972,'Forrest Gump',1),

(5119,'Dark Waters',1),

(5119,'Gamer',3);

The above SQL code is used to create a table containing rental information about a film, it utilizes a compound primary key with [Rental ID] and [Film name] since it ensures there is a unique primary key and no repeats, it also utilises foreign keys for film name and price to ensure the film information table is used and can be linked to the other tables as well.

# Queries:

SELECT [Customer ID], [Customer Name]

FROM CustomerRental, CustomerInformation, FilmInformation, FilmRental

WHERE CustomerRental."Customer ID"=CustomerInformation."Customer ID" AND CustomerRental."Rental ID"=FilmRental."Rental ID" AND FilmRental."Film Name"=FilmInformation."Film Name" AND [Film Price]=2.6

This SQL code answers the query since it ensures the tables have the correct information joined together and makes sure it’s looking for specific information, here ensures it only selects Films priced at £2.6 and allows it to only select and output the customers who rented those films.

SELECT DISTINCT [Customer ID], [Customer Name]

FROM CustomerRental, CustomerInformation, FilmInformation, FilmRental

WHERE CustomerRental."Customer ID"=CustomerInformation."Customer ID" AND CustomerRental."Rental ID"=FilmRental."Rental ID" AND FilmRental."Film Name"=FilmInformation."Film Name" AND ([Film Price]<3 AND [Total Price]>15)

This SQL code answers the query since ensures the correct joins have been done and makes sure to filter out specific information, here it’s only allowing for total price to be £15 and individual film prices to be less than £3, so we will only have the information, the customers who only rented for less than £3 individual films, but the total was greater than £15, we want to be outputted.

SELECT [Customer ID], [Customer Name]

FROM CustomerRental, CustomerInformation, FilmRental

WHERE CustomerRental."Customer ID"=CustomerInformation."Customer ID" AND CustomerRental."Rental ID"=FilmRental."Rental ID" AND [Film Name]='Gamer'

This SQL code answers the query since it makes sure it uses joins to gather the right information that is needed from the correct tables as well as that it ensures that it only filters out films with the name Gamer to output the correct results.

SELECT [Film Name]

FROM FilmRental

WHERE Quantity>0

GROUP BY [Film Name]

This SQL code answers the query since it makes sure to select for only a specific quantity to ensure that it only gets the films that have been rented at least once.

SELECT [Film Name], SUM(Quantity) AS Quantity

FROM FilmRental

GROUP BY [Film Name]

ORDER BY Quantity DESC

This SQL code answers the query since it ensures to output film names with quantity and since its orders the list by the quantity it makes it easy to figure out which film has the highest rentals since it will be shown at the top.

# Conclusion

Overall, the table was not initially a database due to its lack of structure but through the process of normalization we are able to get it into a form where we can create SQL code in order to turn it into an efficient database fit for use.

# **References**

Connolly T.C. (2004) Database Systems: A Practical Approach to Design, Implementation and Management: A Practical Approach to Design, Implementation and Management [https://ebookcentral.proquest.com/lib/coventry/reader.action?docID=5136399#](https://ebookcentral.proquest.com/lib/coventry/reader.action?docID=5136399)

Singh C.S. (2015) Normalization in DBMS: 1NF, 2NF, 3NF and BCNF in Database <https://beginnersbook.com/2015/05/normalization-in-dbms/>

Demba D.M. (2013) Algorithm for Relational Database Normalization Up to 3NF <https://www.researchgate.net/publication/269674039_Algorithm_for_Relational_Database_Normalization_Up_to_3NF>