Contents

Part A) Entity - Relationship Modelling

A. Creation of ERD for scenario

See Figure 2 for ERD

In Figure.1 the Entities – Relations and their attributes have been marked with different colours to be distinguished, the key can also be found in Figure 1.

A human resource management (HRM) department wishes to create a database to monitor its employees. The company is divided into a number of departments, and employees are assigned to one department. Two types of workers have been identified: shop floor workers and office workers. Each shop floor worker has a specific skill and performs a specific task. Office workers on the other hand, are identified by their role and the department to which they belong.

The department has a designated Manager who has overall responsibility for the department and the employees in the department. However, to help manage the department, a number of employees are nominated to supervise groups of staff. When a new employee joins the company, information on previous work history and qualifications is required. On a regular basis, each employee is required to undergo a review, which is normally carried out by the Manager, but may be delegated to a nominated representative.

The company has defined a number of position types, such as Manager, Business Analyst, Salesperson, Secretary, and each type has a number of grades associated with it, which for most non-senior positions determines the employee's salary. At senior level, salary is negotiable. Posts are allocated to a department depending on its workload. For example, a department may be allocated two new Business Analyst posts. A post will be filled by one employee, although over time, employees will fill a number of different posts.

Key

Strong Entity

Attribute

Relationship

Weak Entity

Cardinality

Figure 1 - Understanding entities, attributes, relationships and cardinalities

In our ERD, we have the following Entities, attributes, relationships and cardinalities. These are identified and highlighted in the above figure (Figure 1). In the below table (Table 1), we have listed each entity (strong/weak), the relationship and attributes.

Please note, the links will be shown in the ERD/ERM and not in the below table.

Strong Entities	Weak Entities	Relationships	Attributes
Company	Office Worker	Divided	CompanyName
Department	Shop floor worker	Designated	DepartmentID
Employee	Sales Person	Assigned	DepartmentName
Department Manager	Manager	Allocated (twice)	ManagerID
Posts	Secretary	Review	PostID
Position Type	Business Analyst	Supervises	EmployeeID
			Grade (GradeType)
			Salary (SalaryType)
			Work History
			Qualification
			Skill
			Task
			Role
			Department

Table 1 - List of Strong/weak entities, relationships and attributes

Explanation of ERD

Format of cardinality:
Mandatory/optional many/1 (*Entity name*) to Mandatory/optional many/1 (*Entity name*) [*Relationship*]

Company (attributes) - department (attributes)
Our starting point is Company in which we added as an attribute the companyName. Following on from this, the Company is then divided into multiple departments which we assumed the attributes to be DepartmentID and Department_Name.
Cardinality: mandatory 1 (company) to optional many (departments) [Divided]

Manager (attributes) – department/employees
Onto department manager, each department has a manager and they have a specific responsibility for their employees and the assigned department they are in. The attribute given to manager to identify its managerial position is managerID.
Cardinality: Mandatory 1 (Department Manager) to optional many (Departments) [Responsibility]
Mandatory 1 (Department Manager) to optional many (Employees) [Responsibility]

Employees (attributes)- Department
We move employees are assigned to one department. On employees, we have identified the relationship between departments and employees as mandatory one (department) and optional many (employees) due to the unspecified number of employees, we assume there are multiple employees with the user of the word employees. We have concluded that following attributes are for employees are: Work history, qualification and EmployeeID.
Cardinality: Mandatory 1 (department) to optional many (Employees) [Assigned]

Shop floor/office worker (with attributes)
Under employee we have identified 2 types of workers: Shop floor worker and office worker. These are shown as weak entities due to their dependability. Each type of worker has 2 attributes attached to their working type. Shop floor worker: Skill and Task. Office Worker: Role and Department.
Cardinality: Super Type – Employee → Sub Type – Shop floor worker and Office worker

Employees – manager/representative
Each employee is required to undergo a review which can be done by their department manager or a representative. Therefore, 1 employee will be reviewed by either a manager or representative and as it will be between manager or representative, we choose optional cardinality.
Cardinality: Mandatory one (employees) to Optional one (Representative) [Reviews]
Mandatory one (employees) to Optional one (Department Manager) [Reviews]

Supervises
For the relationship supervises this is formed due to the following statement 'To help manage the department, a number of employees are nominated to supervise groups of staff.' This indicates employees and staff are the same entity and therefore link directly back to its own entity under the supervises relationship. The reason given for using optional many for each of the cardinality is due to the part of the sentence 'A number of employees...supervise groups of staff', this implies that there are many staff managing many other employees giving it a many to many relationships.
Cardinality: Optional many (Employees) to Optional many (employees) [Supervises]

Employees – Position Type

Each employee is defined a position type and this is relayed in the ERD. Under position type there are attributes attached such as Grades and Salary. In addition, each grade and salary are dependable, for example, a non-senior (grade) has a salary that is negotiable (Salary). Due to the ambiguity, we do not know what positions have what grade therefore this is specified at the position type entity. Furthermore, weak entities are given as Business Analyst, Secretary, Sales Person and Manager, this is shown in the format as super and sub types. Additionally, position type will also have the employeeID attribute as it has a number of different positions within the company.

Cardinality: Optional many (Employees) to optional many (Position type) [Allocated]

Super Type – Positon type → Sub Type – Sales Person, Secretary, Business Analyst and Manager

Posts – Departments/employees

Lastly, we have posts who are members of staff that are additional help for each department dependable on the workload. Workload is shown as an attribute from allocated in the relationship towards departments. Posts has been given a postID dependable on the post being allocated and department in need.

Cardinality: Optional many (Posts) to Optional many (Employees & Departments) [Allocated]

B. Explanation of Relational Tables from ERD

See Figure 3 for relational table

In the relational tables, we have defined the ERD to understand the links between the entities.
We have attempted to assume regularities with the entities given for example if we begin with company, each company will have a company name that is defined and this is our primary key for this entity. The company links to departments as the only relationship for this entity.
Under Departments, the primary key is DepartmentID and the department name is another attribute given. Department ID and Department name are needed so they can be identified on the system for example department name: HR (Human resources) can have department ID: 1 and this is stored on the system under the employees who work within the HR (1) department for easier identification. Further, we link the companyName to departments as you need a company to be able to have departments. Onto mangerID, through the statement we were given 'The department has a designated manager' therefore if there are multiple departments, we need to have a managerial ID for each manager within the company. The entity Department has some links using primary key as department ID is used within Employees, Allocated and Position Type.
Moving onto Department Manager, as each department has a designated manager the primary key used here is ManagerID to indicate the managers identification. The managerID as the primary key is linked to other tables such as Departments and Employee and each manager will have an employeeID as well as their manager ID.
Onto the employee entity, firstly we assume that each employee's identification will be established by an employeeID number. We see the following attributes are assigned to employees: Work history and qualification. Furthermore, other entities/attributes from other entities are identified such as departmentID dependable on the department the employees chosen department. As the representative is reviewing the employees work, they are still an employee to be able to review their work and will also be given an employeeID. The employee entity is linked with other entities such as department manager, allocated and position type.
Representative is shown as an entity where the employee will undergo a review from either a manager or a representative. Therefore, the representativeID has been given to this entity to recognise each representative.
Both Shop floor worker and Office worker are shown with the relationship of a super and sub type. This is because they are specialised versions of the employee entity with specific attributes assigned to each particular entity.
The allocated entity is a composite key that holds both foreign and primary keys. The primary and foreign keys for this entity are employeeID and departmentID with the attributes being workload and posts.
Lastly, we have position type which is shown as a super type where we have taken the employeeID as the primary key and the following attributes: Salary type and grade type. Each specific position which is shown as sub types: Sales Person, Manager, Secretary and Business analyst will have different grade types and salary types dependable on the seniority of the position which has not been defined in the statement given.

Here are the tables generated of our ERD:
Note: supertypes and subtypes are illustrated separately

Company (CompanyName)
Departments (DepartmentID, DepartmentName, CompanyName*, ManagerID*)
Department Manager (ManagerID, EmployeeID*)
Representative (RepresentativeID)
Employees (EmployeeID, DepartmentID*, Work history, Qualification, RepresentativeID*, ManagerID*)
Position Type(EmployeeID, Salary type, Grade type, DepartmentID*)

Supertype and subtype:

Position Types
[supertype]
Position Type (EmployeeID, SalaryType, GradeType)

[subtypes]
Sales Person (EmployeeID)
Business Analyst (EmployeeID)
Secretary (EmployeeID)
Manager (EmployeeID)

Types of workers
[supertype]
Employees (EmployeeID, work history, qualification)

[subtype]
Shop floor worker (EmployeeID, skill, task)
Office worker (EmployeeID, role, department)

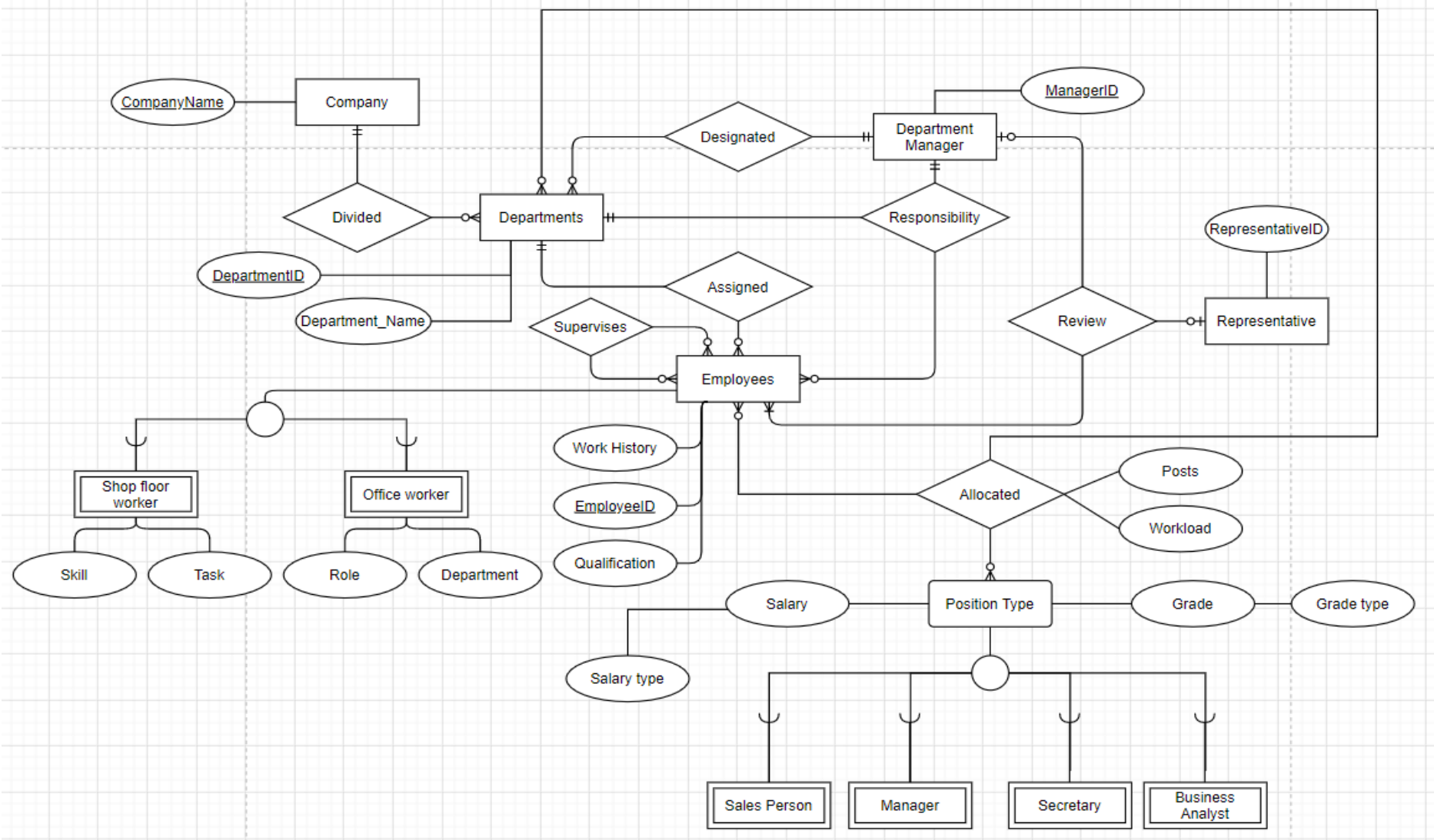

Figure 2 - Entity Relationship Diagram

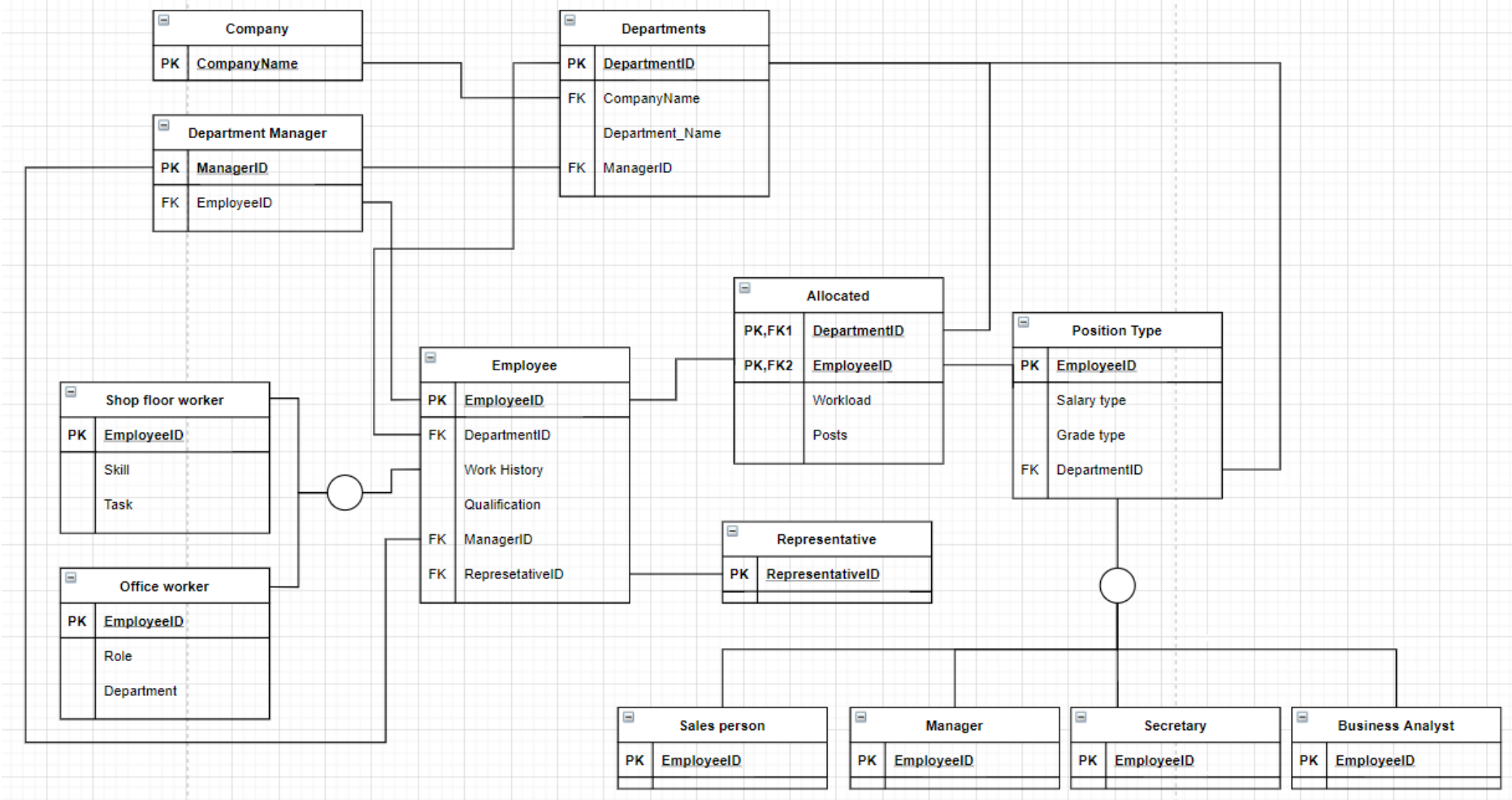

Figure 3 - Relational Table

Part B) SQL Programming

(All the coding has been saved in a .txt file that will be demonstrated in the very end of this part)

A. Creating tables and inserting the data/values

For the SQL Programming part of the Coursework, we have installed the University VPN and we work on PuTTY using Oracle.

We begin by manually creating the tables needed and determining the Primary and Foreign keys, as illustrated in the figures below *(Figure 4 – Figure 7)*. We begin by creating the tables *users* and *category* because they are the tables that we will use to link the other two so as to work for the queries.

Users:

```
SQL> CREATE TABLE users (
  2  userID VARCHAR2(7) PRIMARY KEY,
  3  name VARCHAR2(20) NOT NULL,
  4  emailAddress VARCHAR2(30) NOT NULL
  5  );

Table created.

SQL> INSERT INTO users (userID, name, emailAddress)VALUES ('kendj3', 'Kenderine, J', 'kendj3@hotmail.co.uk');

1 row created.

SQL> INSERT INTO users (userID, name, emailAddress)VALUES ('patel11', 'Patel, F', 'patel11@ntl.co.uk');

1 row created.

SQL> INSERT INTO users (userID, name, emailAddress)VALUES ('flak05', 'Flavel, K', 'flak05@freeserve.co.uk');

1 row created.
```

Figure 4. Creating the table 'users'

Category:

```
SQL> CREATE TABLE category(
  2  categoryCode CHAR(3) PRIMARY KEY,
  3  title VARCHAR2(20) NOT NULL
  4  );

Table created.

SQL> INSERT INTO category VALUES ('C11', 'Classics');

1 row created.

SQL> INSERT INTO category VALUES ('C12', 'Pop-Rock');

1 row created.

SQL> INSERT INTO category VALUES ('C13', 'Movie Soundtrack');

1 row created.
```

Figure 5. Creating the table 'category

Music:

```
SQL> CREATE TABLE music(
  2  musicID CHAR(4) PRIMARY KEY,
  3  title VARCHAR2(30) NOT NULL,
  4  categoryCode CHAR(3) REFERENCES category,
  5  costPerDownload DECIMAL(4,2) NOT NULL
  6  );

Table created.

SQL> INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M001', 'James Bond: Golden Eyes', 'C13', '0.99');

1 row created.

SQL> INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M002', 'Lake House', 'C13', '1.99');

1 row created.

SQL> INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M003', 'Dvorak: Symphony No 9', 'C11', '1.49');

1 row created.
```

Figure 6. Creating the table 'music'

Music Download:

```
SQL> CREATE TABLE musicDownload (
  2  userID VARCHAR2(7) REFERENCES users,
  3  musicID CHAR(4) REFERENCES music,
  4  downloadDate DATE NOT NULL,
  5  PRIMARY KEY (userID,musicID)
  6  );

Table created.

SQL> INSERT INTO musicDownload VALUES ('kendj3','M002', TO_DATE('03-May-18', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('johnsj9','M005', TO_DATE('01-May-19', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('patel11','M002', TO_DATE('06-May-18', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('johnsj9','M001', TO_DATE('06-May-19', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('kendj3','M003', TO_DATE('01-Aug-19', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('keita77','M004', TO_DATE('02-Aug-19', 'dd-mm-yy'));

1 row created.

SQL> INSERT INTO musicDownload VALUES ('simpb91','M007', TO_DATE('05-Sep-18', 'dd-mm-yy'));

1 row created.

SQL>
```

Figure 7. Creating the table 'musicDownload'

After the commands for creating the tables we can check if the rows have been properly inserted as seen in *Figure 8* & in *Figure 9.*

```
USERID  NAME                 EMAILADDRESS
------- -------------------- ------------------------
kendj3  Kenderine, J         kendj3@hotmail.co.uk
patel11 Patel, F             patel11@ntl.co.uk
flak05  Flavel, K            flak05@freeserve.co.uk
johnsj9 Johnson, J           johnsj9@msn.co.uk
keita77 Keita, R             keita77@hotmail.co.uk
simpb91 Simpson, B           Simpb91@tesco.co.uk

6 rows selected.

SQL> SELECT * FROM category;

CAT TITLE
--- --------------------
C11 Classics
C12 Pop-Rock
C13 Movie Soundtrack
```

Figure 8. Checking the tables 'users' & 'category'

```
SQL> SELECT * FROM music;

MUSI TITLE                          CAT COSTPERDOWNLOAD
---- ------------------------------ --- ---------------
M001 James Bond: Golden Eyes        C13              .99
M002 Lake House                     C13             1.99
M003 Dvorak: Symphony No 9          C11             1.49
M004 Handel: Water Music            C11             1.79
M005 Sense and Sensibility          C13              1.5
M006 Beatles: Yesterday             C12              1.1
M007 Elton John: Your Song          C12              .89

7 rows selected.

SQL> SELECT * FROM musicDownload;

USERID  MUSI DOWNLOADD
------- ---- ---------
kendj3  M002 03-MAY-18
johnsj9 M005 01-MAY-19
patel11 M002 06-MAY-18
johnsj9 M001 06-MAY-19
kendj3  M003 01-AUG-19
keita77 M004 02-AUG-19
simpb91 M007 05-SEP-18

7 rows selected.

SQL>
```

Figure 9. Checking the tables 'music' & 'musicDownload'

B. SQL Coding – Answering the Queries

1. The music id, the title and the categoryCode of all the music in the database, ordered by title.

As can be seen in Figure 10 for the 1st question we just need to select the columns of *musicID*, *title*, and *categoryCode* and apply the **ORDER BY** command for the column *title*.

```
SQL> SELECT musicID,title,categoryCode FROM music
  2  ORDER BY title ASC;

MUSI TITLE                          CAT
---- ------------------------------ ---
M006 Beatles: Yesterday             C12
M003 Dvorak: Symphony No 9          C11
M007 Elton John: Your Song          C12
M004 Handel: Water Music            C11
M001 James Bond: Golden Eyes        C13
M002 Lake House                     C13
M005 Sense and Sensibility          C13

7 rows selected.

SQL>
```

Figure 10. Answering the 1st query

2. The number of users who downloaded 'Pop-Rock' category of music.

In Figure 11 we can see that the number of Users who downloaded 'Pop – Rock' category is only one. We used the **INNER JOIN** command so as to browse across the tables using the references, we made in the creating part.

```
SQL> SELECT COUNT(userID) FROM musicDownload
  2  INNER JOIN music ON musicDownload.musicID=music.musicID
  3  INNER JOIN category ON music.categoryCode=category.categoryCode
  4  WHERE category.title='Pop-Rock';

COUNT(USERID)
-------------
            1

SQL>
```

Figure 11. Answering the 2nd query

3. The number of music downloads for each of the categories. The result listing should include the titles of the categories and the number of music downloads for each category title.

In Figure.12 we can see that the 'Pop-Rock' category has been downloaded once, 'Classics' category twice and 'Movie soundtracks' was downloaded 4 times. The **GROUP BY** clause gathers the 2 columns from the 2 separate tables and joins them together by summing the rows as shown in Figure 12.

```
SQL> SELECT category.title,  COUNT(music.categoryCode) FROM musicDownload
INNER JOIN music ON music.musicID=musicDownload.musicID
INNER JOIN category ON music.categoryCode=category.categoryCode
GROUP BY category.title, music.categoryCode;
  2    3    4
TITLE                COUNT(MUSIC.CATEGORYCODE)
-------------------- -------------------------
Pop-Rock                                     1
Classics                                     2
Movie Soundtrack                             4

SQL>
```

Figure 12. Answering the 3rd query

4. The titles of the categories for which music was downloaded more than once.

In Figure 13, we can see that the category 'Classics' is shown twice and 'Movie SoundTrack' is shown 4 times. These are the categories that have been downloaded more than once. In order achieve this result, we have used the **HAVING** clause on the aggregate data to filter by using the augmented assignment operator of greater than 1.

```
SQL> SELECT category.title, COUNT(music.categoryCode) FROM musicDownload
  2  INNER JOIN music ON musicDownload.musicID=music.musicID
  3  INNER JOIN category ON music.categoryCode=category.categoryCode
  4  GROUP BY category.title, music.categoryCode
  5  HAVING COUNT(music.categoryCode) > 1;

TITLE                COUNT(MUSIC.CATEGORYCODE)
-------------------- -------------------------
Classics                                     2
Movie Soundtrack                             4
```

Figure 13. Answering the 4th query

(The whole code from Part B which is saved in a txt file)

```
########################
# Creating users table#
########################

CREATE TABLE users (
userID VARCHAR2(7) PRIMARY KEY,
name VARCHAR2(20) NOT NULL,
emailAddress VARCHAR2(30) NOT NULL
);


#####################################
# Inserting Values in the users Table#
#####################################

INSERT INTO users (userID, name, emailAddress)VALUES ('kendj3', 'Kenderine, J',
'kendj3@hotmail.co.uk');

INSERT INTO users (userID, name, emailAddress)VALUES ('patel11', 'Patel, F',
'patel11@ntl.co.uk');

INSERT INTO users (userID, name, emailAddress)VALUES ('flak05', 'Flavel, K',
'flak05@freeserve.co.uk');


INSERT INTO users (userID, name, emailAddress)VALUES ('johnsj9', 'Johnson, J',
'johnsj9@msn.co.uk');


INSERT INTO users (userID, name, emailAddress)VALUES ('keita77', 'Keita, R',
'keita77@hotmail.co.uk');


INSERT INTO users (userID, name, emailAddress)VALUES ('simpb91', 'Simpson, B',
'Simpb91@tesco.co.uk');



############################
# Creating category table#
############################

CREATE TABLE category(
categoryCode CHAR(3) PRIMARY KEY,
title VARCHAR2(20) NOT NULL
);


########################################
# Inserting Values in the category Table#
########################################

INSERT INTO category VALUES ('C11', 'Classics');
INSERT INTO category VALUES ('C12', 'Pop-Rock');
INSERT INTO category VALUES ('C13', 'Movie Soundtrack');
```

```
########################
# Creating music table #
########################

CREATE TABLE music(
musicID CHAR(4) PRIMARY KEY,
title VARCHAR2(30) NOT NULL,
categoryCode CHAR(3) REFERENCES category,
costPerDownload DECIMAL(4,2) NOT NULL
);


#####################################
# Inserting Values in the music Table #
#####################################


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M001',
'James Bond: Golden Eyes', 'C13', '0.99');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M002',
'Lake House', 'C13', '1.99');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M003',
'Dvorak: Symphony No 9', 'C11', '1.49');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M004',
'Handel: Water Music', 'C11', '1.79');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M005',
'Sense and Sensibility', 'C13', '1.50');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M006',
'Beatles: Yesterday', 'C12', '1.10');


INSERT INTO music (musicID, title, categoryCode, costPerDownload)VALUES ('M007',
'Elton John: Your Song', 'C12', '0.89');



###############################
# Creating musicDownload table#
###############################

CREATE TABLE musicDownload (
userID VARCHAR2(7) REFERENCES users,
musicID CHAR(4) REFERENCES music,
downloadDate DATE NOT NULL,
PRIMARY KEY (userID,musicID)
);
```

```
##############################################
# Inserting Values in the musicDownload Table#
##############################################

INSERT INTO musicDownload VALUES ('kendj3','M002', TO_DATE('03-May-18', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('johnsj9','M005', TO_DATE('01-May-19', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('patel11','M002', TO_DATE('06-May-18', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('johnsj9','M001', TO_DATE('06-May-19', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('kendj3','M003', TO_DATE('01-Aug-19', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('keita77','M004', TO_DATE('02-Aug-19', 'dd-mm-
yy'));

INSERT INTO musicDownload VALUES ('simpb91','M007', TO_DATE('05-Sep-18', 'dd-mm-
yy'));


#########################
#Answers to the queries#
#########################

1)
SELECT musicID,title,categoryCode FROM music
ORDER BY title ASC;


2)
SELECT COUNT(userID) FROM musicDownload
INNER JOIN music ON musicDownload.musicID=music.musicID
INNER JOIN category ON music.categoryCode=category.categoryCode
WHERE category.title='Pop-Rock';


3)
SELECT category.title,  COUNT(music.categoryCode) FROM musicDownload
INNER JOIN music ON music.musicID=musicDownload.musicID
INNER JOIN category ON music.categoryCode=category.categoryCode
GROUP BY category.title, music.categoryCode;


4)
SELECT category.title, COUNT(music.categoryCode) FROM musicDownload
INNER JOIN music ON musicDownload.musicID=music.musicID
INNER JOIN category ON music.categoryCode=category.categoryCode
GROUP BY category.title, music.categoryCode
HAVING COUNT(music.categoryCode) > 1;
```

Part C) Sequential and Parallel Processing

A. i. Creating the corresponding table

In Figure.14 & Figure.15 the whole code for the query is illustrated, which is saved in a txt file. You can see here we are creating the tables, inputting the values and checking the values have been inputted. (Anon., n.d.)

```
SQL> CREATE TABLE sampRec(
  2  orderNo NUMBER NOT NULL,
  3  productNo NUMBER NOT NULL,
  4  price Decimal(5,2),
  5  quantity NUMBER,
  6  sales DECIMAL(6,2),
  7  qtrID NUMBER NOT NULL,
  8  monthID NUMBER NOT NULL,
  9  YearID NUMBER NOT NULL,
 10  PRIMARY KEY(orderNO,productNo)
 11  );

Table created.

SQL> INSERT INTO sampRec VALUES (10107, 2, 95.7, 30, 2871, 1, 2, 2003);

1 row created.

SQL> INSERT INTO sampRec VALUES (10107, 5, 99.91, 39, 3896.49, 1, 2, 2003);

1 row created.
```

Figure 14. Creating the table sampRec

```
SQL> SELECT * FROM sampRec;

   ORDERNO  PRODUCTNO      PRICE   QUANTITY      SALES      QTRID     MONTHID
---------- ---------- ---------- ---------- ---------- ---------- ----------
    YEARID
----------
     10107          2       95.7         30       2871          1          2
      2003

     10107          5      99.91         39    3896.49          1          2
      2003

     10121          5      81.35         34     2765.9          1          2
      2003


   ORDERNO  PRODUCTNO      PRICE   QUANTITY      SALES      QTRID     MONTHID
---------- ---------- ---------- ---------- ---------- ---------- ----------
    YEARID
----------
     10134          2      94.74         41    3884.34          3          7
      2004

     10134          5        100         27    3307.77          3          7
      2004

     10159         14        100         49    5205.27          4         10
      2005


   ORDERNO  PRODUCTNO      PRICE   QUANTITY      SALES      QTRID     MONTHID
---------- ---------- ---------- ---------- ---------- ---------- ----------
    YEARID
----------
     10168          1      96.66         36    3479.76          4         10
      2006

     10180         12        100         42     4695.6          4         11
      2006

     10180          9      86.13         29    2497.77          4         11
      2006

9 rows selected.

SQL>
```

Figure 15. Checking the table sampRec

A. ii. Produce with justification the SQL code to determine, for each product, the total quantity sold in each month for each year.

In comparison to Figure 15 you can clearly see there are 9 rows of recorded data. In Figure 16 where we have coded and summed the quantity for each product per month and year, there is now 8 rows. The rows that have been combined are (*orderNo - 10107, productNo - 5)* and (*orderNo - 10121, productNo - 5)* from Figure 15, it has taken both quantities 39 and 34 and added them together to get a new value of 73 as shown in Figure 16, this is because the attribute of month and year for these products are the same (year 2003 and month 2).

Since the table given is a sample, we wanted to find a way to have a general solution. We firstly select the columns we will need to use from the sampRec table and the SUM of quantities. Then we GROUP BY the columns so as to get the summary of the same values into rows.

In conclusion, any products from this query that exist in our database with the same month and year they will be merged together and the quantity of the records will be combined.

Below the Figure.16 we have inserted the whole code from the text file we worked on.

```
SQL> SELECT sampRec.productNo, sampRec.monthID, sampRec.YearID, SUM(sampRec.quantity)
  2  FROM sampRec
  3  GROUP BY sampRec.productNo,  sampRec.monthID, sampRec.YearID
  4  ORDER BY sampRec.productNo, sampRec.monthID, sampRec.YearID;

 PRODUCTNO    MONTHID     YEARID SUM(SAMPREC.QUANTITY)
---------- ---------- ---------- --------------------
         1         10       2006                   36
         2          2       2003                   30
         2          7       2004                   41
         5          2       2003                   73
         5          7       2004                   27
         9         11       2006                   29
        12         11       2006                   42
        14         10       2005                   49

8 rows selected.
```

Figure 16. Answering the query

(The whole code from Part C)a which is saved in a txt file)

```
############################
#Creating the table sampRec#
############################

CREATE TABLE sampRec(
orderNo NUMBER NOT NULL,
productNo NUMBER NOT NULL,
price Decimal(5,2),
quantity NUMBER,
sales DECIMAL(6,2),
qtrID NUMBER NOT NULL,
monthID NUMBER NOT NULL,
YearID NUMBER NOT NULL,
PRIMARY KEY(orderNO,productNo)
);

#######################################
#Inserting values in the sampRec table#
#######################################

INSERT INTO sampRec VALUES (10107, 2, 95.7, 30, 2871, 1, 2, 2003);
INSERT INTO sampRec VALUES (10107, 5, 99.91, 39, 3896.49, 1, 2, 2003);
INSERT INTO sampRec VALUES (10121, 5, 81.35, 34, 2765.9, 1, 2, 2003);
INSERT INTO sampRec VALUES (10134, 2, 94.74, 41, 3884.34, 3, 7, 2004);
INSERT INTO sampRec VALUES (10134, 5, 100.0, 27, 3307.77, 3, 7, 2004);
INSERT INTO sampRec VALUES (10159, 14, 100.0, 49, 5205.27, 4, 10, 2005);
INSERT INTO sampRec VALUES (10168, 1, 96.66, 36, 3479.76, 4, 10, 2006);
INSERT INTO sampRec VALUES (10180, 12, 100.0, 42, 4695.6, 4, 11, 2006);
INSERT INTO sampRec VALUES (10180, 9, 86.13, 29, 2497.77, 4, 11, 2006);

############################
#Answer to the query a)ii)#
############################

SELECT sampRec.productNo, sampRec.monthID, sampRec.YearID, SUM(sampRec.quantity)
FROM sampRec
GROUP BY sampRec.productNo, sampRec.monthID, sampRec.YearID
ORDER BY sampRec.productNo, sampRec.monthID, sampRec.YearID;
```

B. Assuming that the data is too large to be processed in a centralised manner in a relational database, and that it is stored in an ordinary file, produce a decentralised solution which applies MapReduce to the data processing. Justify your decisions and all the steps of your solution. Use diagrams if required.

MapReduce is a distributed data processing model withing the Hadoop framework and is designed to scale data processing over multiple computing nodes.
By using MapReduce, we are able to handle large values of data because we can allocate our tasks into smaller chunks and process them in parallel. MapReduce is based on a share-nothing architecture which means that each task is independent from one another so the machine can work in parallel with no problems and finish the required task much faster than working based on the relational model. (Anon., n.d.)

Note: In the below MapReduce example, we have taken the first 3 rows from the sample records table.

1. INPUT - SPLITTING

Firstly, we input the data in the form of a file and not as a table. This gives us the ability to work in parallel since we can split the data into small blocks and assign a Mapper to each block so as to do various computations in different nodes. So, we split the data into blocks so as to compute them separately.

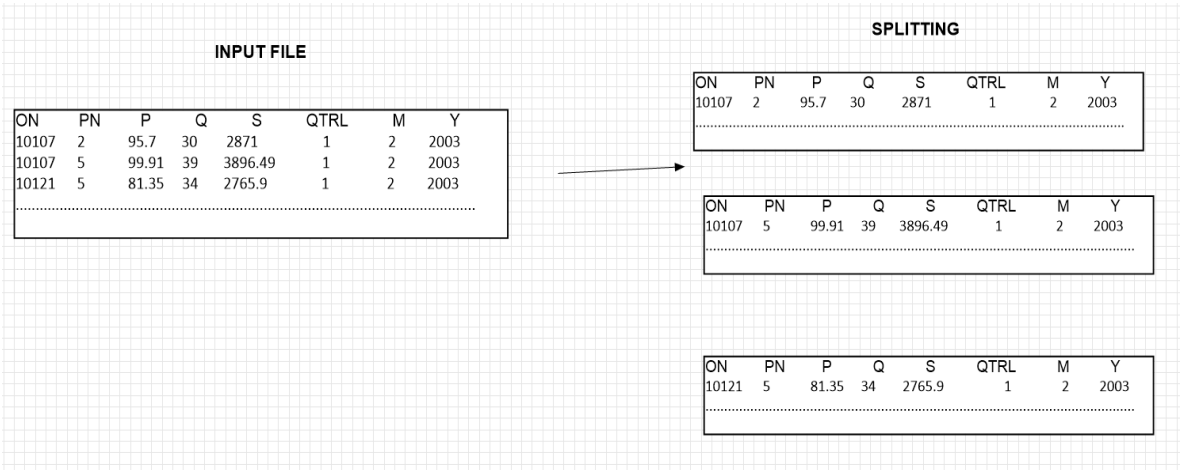

Figure 17. Input - Splitting of the sample data

2. SPLITTING - MAPPING

After the Splitting, we create a list in each chunk where the Mapper keeps only the necessary values that needs like Month, Year and Quantity and stores them as a Key-Value store.

Under the mapping stage of this process we have combined the M and Y together under Q which in the below Figure.18, this is shown for example (M2, Y2003), Q22003.

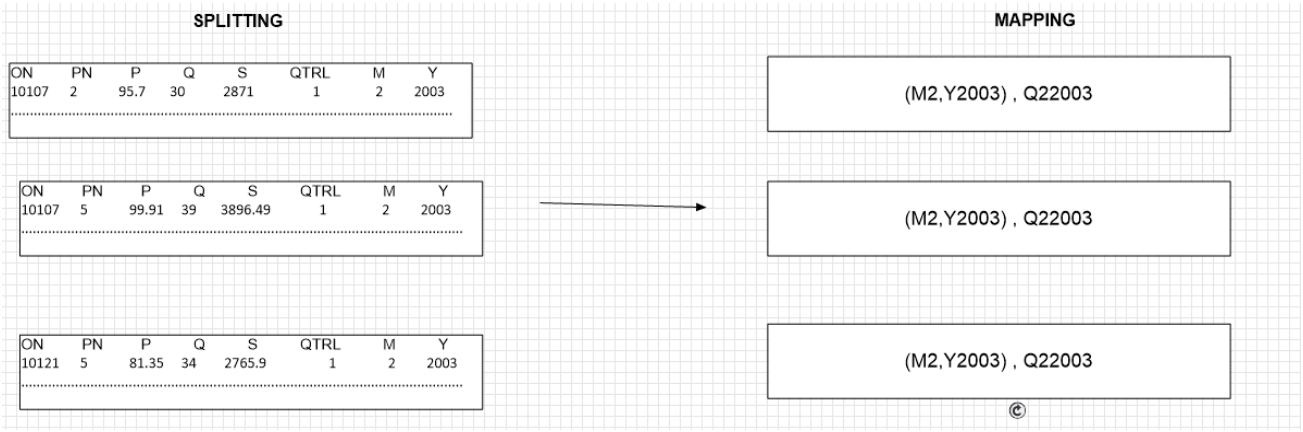

Figure 18. Splitting - Mapping of the sample data

3. MAPPING - SHUFFLING

After the *Mapping* we collect the values with the same Key (Month, Year) with the Shuffling procedure and we store them to different blocks. For example, in the first node we keep all the orders from the Month January, the Year 2003 and the Quantities for each product. Respectively, in the second node we keep the same attributes for February and this is ongoing.

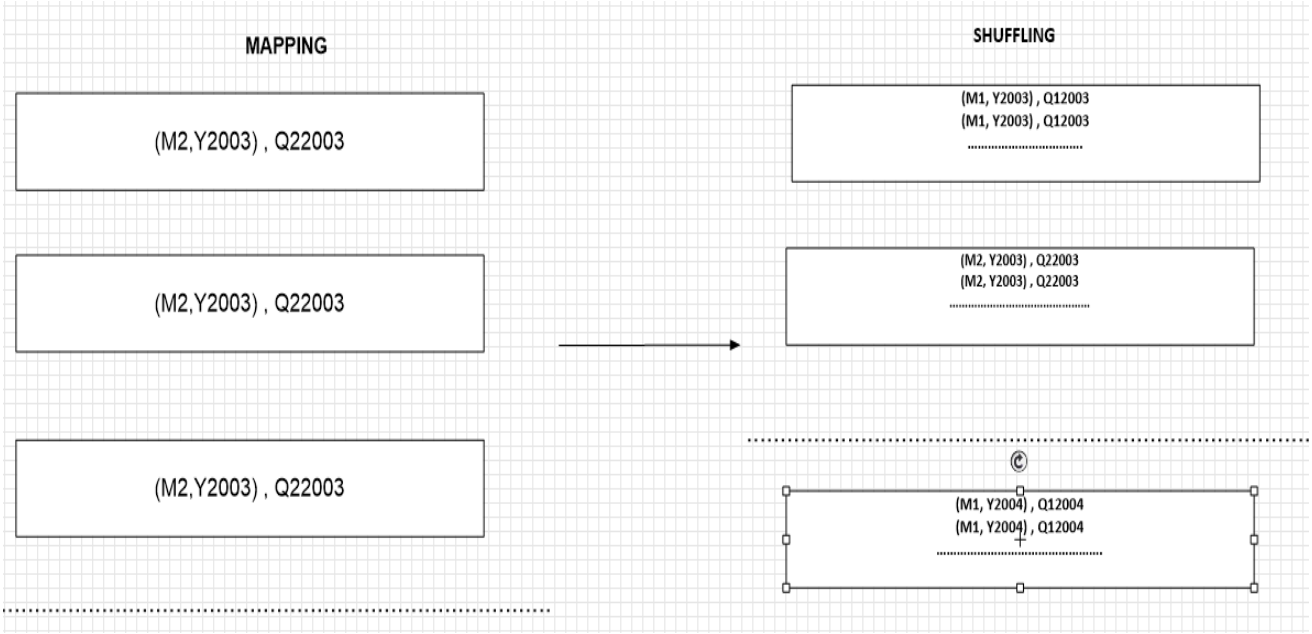

Figure 19. Mapping - Shuffling of the sample data

4. SHUFFLING - REDUCING

After Shuffling, the machine will start Reducing the data by adding all the Quantities and storing them as a sum, based on the Month and Year.

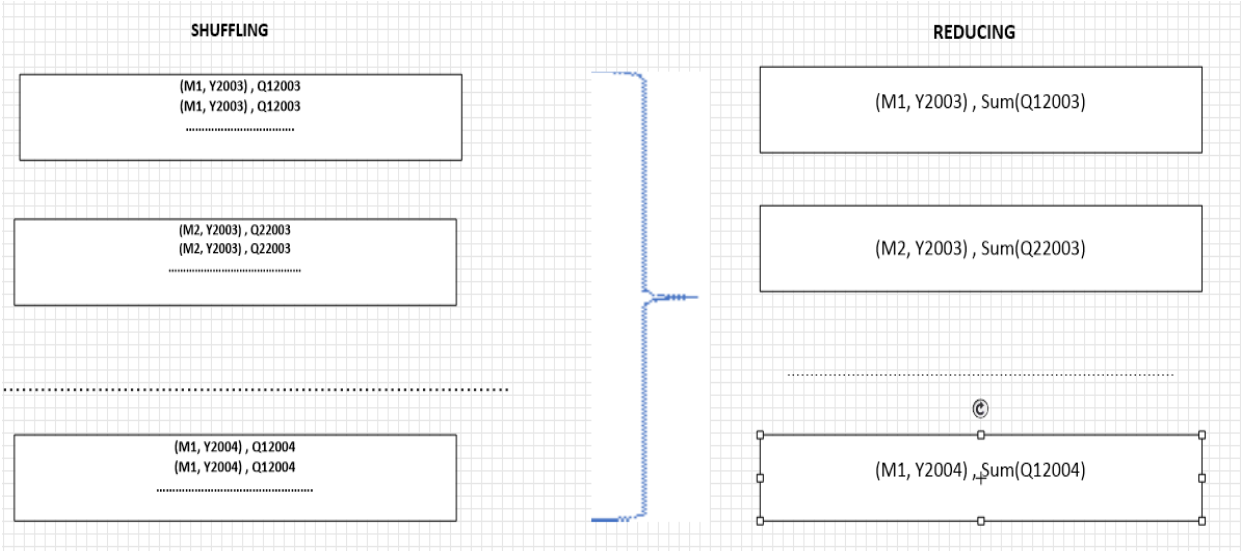

Figure 20. Shuffling - Reducing of the sample data

5. REDUCING – AGGREGATING

Finally, we have a list with all the key attributes and the Sum of Product Quantities that are collected and written to the output file. By having different Mappers running in parallel we can produce the results much faster. Aggregating will just collect the values from the reducing stage which is the last element of MapReduce.

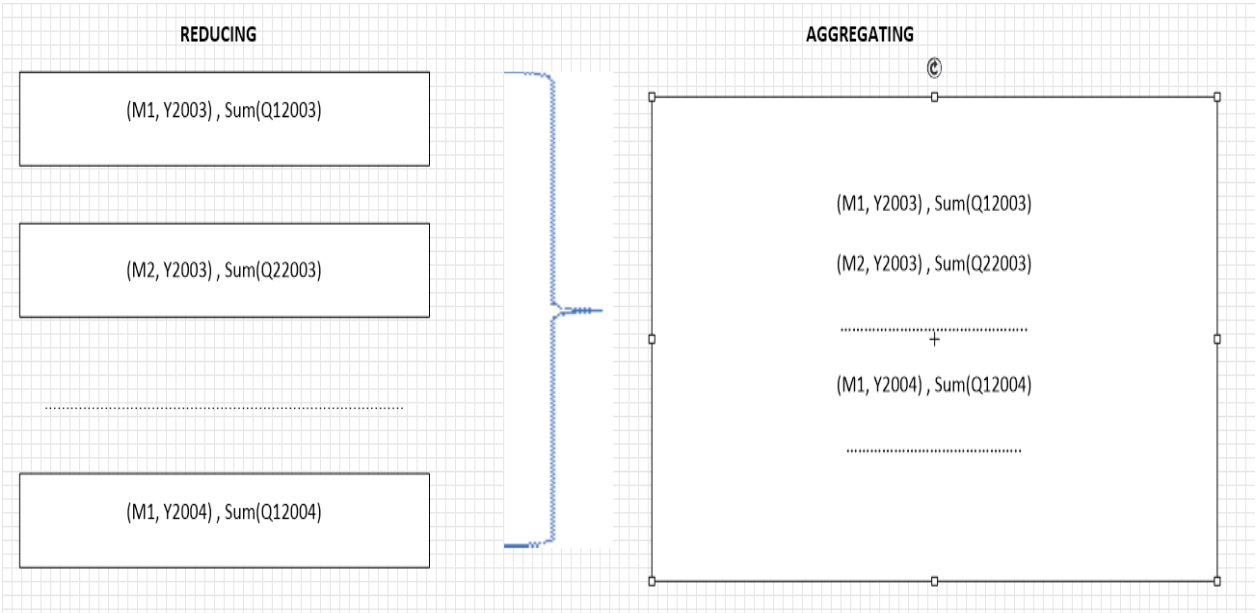

Figure 21. Reducing - Aggregating of the sample data

Part D) Research Report

Craigslist is a popular and renowned for its classified adverts and job postings with over 1.5 million being posted every day. It is also used in over 50 countries and more than 570 cities. Craigslist stores and archives billions of data records in a variety of formats, and with the company evolving and growing the method of storing in MySQL will not be manageable and controllable. (Anon, n.d.)

Relational Databases (RDB) are can become limiting because of the characteristics that RDB uses such as data storage model, schemas and scalability to name some. Firstly, with the data storage model in the RDB it has to have a fixed number of columns and rows, which is unsuitable especially when the data is continually growing at a rate that cannot be handled by RDB. Secondly, MySQL uses "Schema-on-Write" model which is not very flexible. MongoDB's schema uses a "Schema-on-Read" model which allows more flexibility and allows the data to be stored in any form without any restriction which is suitable for Craigslist's operations. Additionally, the analysis manipulation of the data will be determined by the time of the use of the data. Thirdly, the scalability, within MySQL the scaling is done vertically as opposed to MongoDB's horizontally. When performing scaling horizontally it is scaled out across a commodity server and this can be widely supported. (Smallcombe, 2020)

The main reason for the change is because of the flexibility and scalability that MongoDB offers. When Craigslist was created big data was not even considered, over time data has expanded and grown and now big data has been and is continually being developed. Data nowadays comes mostly in an unstructured format, in fact, more than 90% of the data we collect nowadays come in an unstructured form and so Craigslist had to deal with a major problem which was to find a way to manipulate and store information that comes with different forms and in an unstructured way. This is where the development of NoSQL databases and new software came to help with dealing with what we refer to as ''Big Data''.
As mentioned above the scalability is the reason for Craigslist choice for moving to MongoDB as each post and its metadata can be stored as a single document. MongoDB uses a document store, therefore not needing to worry that you may lose values during the movement of the data as the data stored in a document and not in a structured table. The schema fluctuates on the database that is live, this allows MongoDB to accommodate with these changes without costly schema migrations.
Due to MongoDB's design, similar concepts and features to MySQL the transition would be easier than transitioning to a different NoSQL model that does not hold similar features to its original system. (Anon, n.d.;)

Using NoSQL highly outweighs using MySQL, there are multiple advantages that shows NoSQL is by far more superior such as breadth of functionality, handles changeover time, supports multiple data structures, executes the code next to the data and scalable over commodity hardware. Most RDB support the same features but in a slightly different way however, with NoSQL it comes in 4 core types: Key-Value, Column, document (MongoDB) and graph stores. Dependable on the type requirement, there is a database that can suit your specific needs. Due to the large volumes of data, queries and processes now work in parallel to the data. (Fowler, n.d. & Schaefer, 2020)

References

Fowler, A. 10 Advantages of NoSQL over RDBMS - dummies. dummies. Retrieved 28 October 2020, from https://www.dummies.com/programming/big-data/10-advantages-of-nosql-over-rdbms/.

Anon. MongoDB Case Study: Craigslist. MongoDB. Retrieved 27 October 2020, from https://www.mongodb.com/post/15781260117/mongodb-case-study-craigslist.

Smallcombe, M. (2020). SQL vs NoSQL: 5 Critical Differences. Xplenty. Retrieved 27 October 2020, from https://www.xplenty.com/blog/the-sql-vs-nosql-difference/.

Schaefer, L. (2020). NoSQL vs SQL Databases. MongoDB. Retrieved 28 October 2020, from https://www.mongodb.com/nosql-explained/nosql-vs-sql.

Anon. MapReduce 101: What It Is & How to Get Started - Talend. Talend Real-Time Open Source Data Integration Software. Retrieved 27 October 2020, from https://www.talend.com/resources/what-is-mapreduce/.

Anon. W3Schools Online Web Tutorials. W3schools.com. Retrieved 29 November 2020, from https://www.w3schools.com/default.asp.