7086CEM Data Management Systems

Coursework

Database Management

Module Leader: Dr. Rachid Anane

I can confirm that all work submitted is our own: Yes

Part A: Question 1:

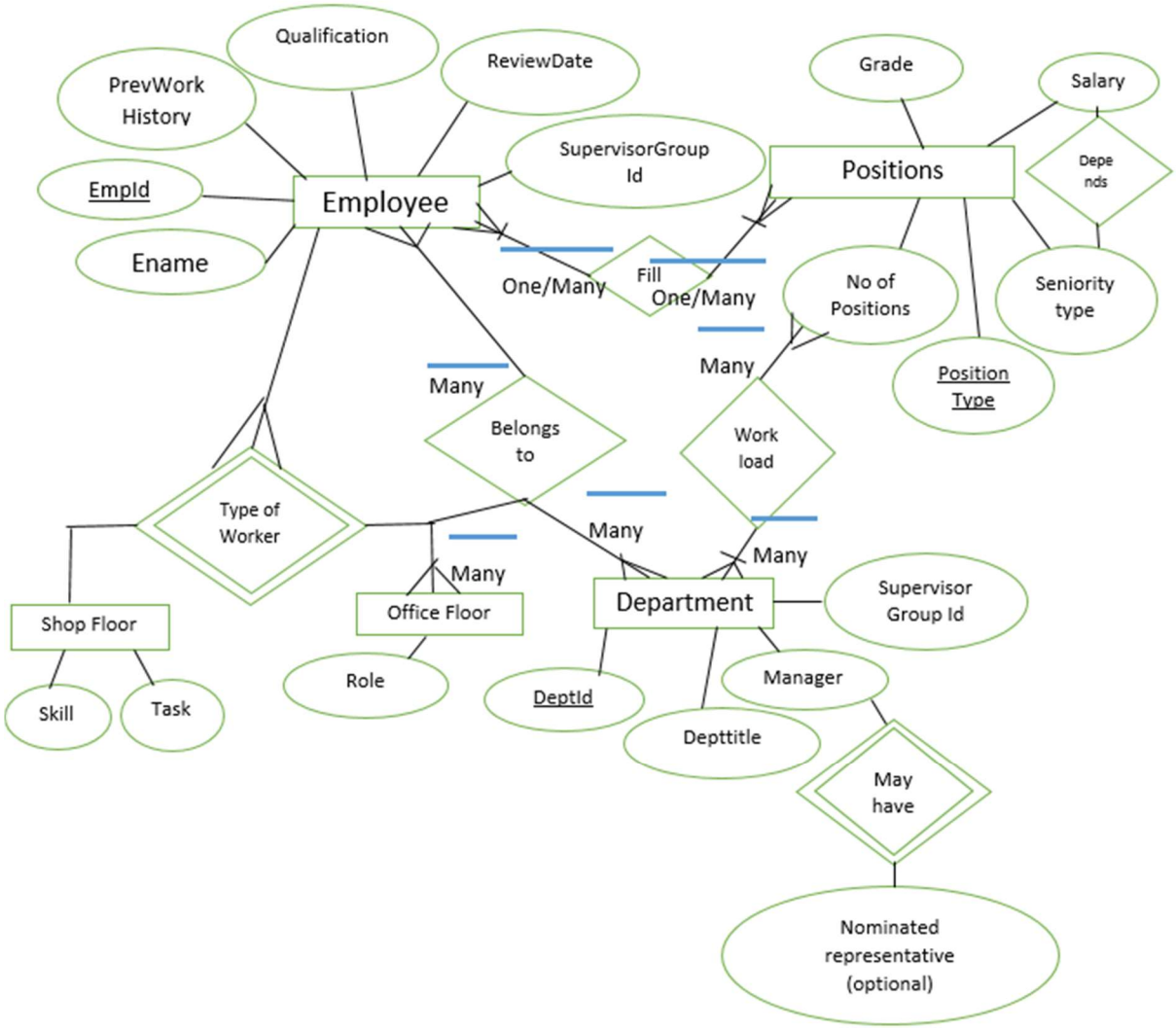
Entities of each table:

- ➔ Entities for Employee table are: EmpId (primary Key) , Ename, PrevWorkHistory, Qualification, ReviewDate, SupervisorGroupId, DeptId (Foreign key-belongs to department table) , PositionType (Foreign key-belongs to positions table), employeetype(Foreign key-belongs to workers table)(optional)
- ➔ Entities for Positions table are: PositionType (primary Key) (such as Manager, Business Analyst, Salesperson, Secretary), Department Id ((primary key) Noofpositions, Grades, Salary (depend on seniioritytype), Seniioritytype.
- ➔ Entities for Department table are: DeptId(primary Key), Dept title , Manager, SupervisorGroupId, Nominated Representative(optional), No of positions (depending upon the work load)
- ➔ Entities for Workers table are: employeetype (primary Key) (Shop floor and office floor workers), Skill, Task, Role, DeptId (to which they belongs to).

Relationships:

- ➔ One employee belongs to one department , many employees belongs to many departments
- ➔ One position is filled by one employee, employee's fills number of different positions.
- ➔ One department have many positions, many departments have many positions.
- ➔ Each position type may have one or many no of positions in each department. (for example, a department may be allocated 10 salesperson positions)
- ➔ Each employee has a designated manager (mandatory).
- ➔ Previous work history and Qualifications of new employee are required in employee table. (mandatory)
- ➔ Each employee is required to undergo a review on a specific date. (Mandatory).
- ➔ Manager may have a nominated representative to carry out review of employees (optional).
- ➔ Seniority type determines the salary, so salary depends on seniority type.
- ➔ Number of positions in each department depends on the workload of that particular department.
- ➔ In workers table, skill and task are only for shop floor workers so it can be null for office floor workers, whereas role and deptid are only for office floor workers so it can be null for shop floor workers.
- ➔ Office workers are identified by their role and the department to which they belong to.
- ➔ Employee type in employee table is optional. If employee belongs to worker type, then the type will be filled in that column, if employee not belongs to worker type then the value will be null in that column.

ER Diagram:



Part A: Question 2:

Employee (EmpId, Ename, PrevWorkHistory, Qualification, ReviewDate, SupervisorGroupId DeptId*, Positiontype*, employeetype*)

Department (DeptId, Deptitle, Manager, SupervisorGroupId, NominatedRepresentative)

Positions (Positiontype, DeptId, Noofpositions, Grade, Salary, Senioritytype)

Workers (employeetype, Skill, Task, Role, DeptId)

Part B: Question 1:

SQL statements for creating all the below four tables (Employee, Department, SalaryGrade, PensionScheme)

- ➔ Given Employee, Department, SalaryGrade and PensionScheme tables. Here Employee table is the dependent table and all other three are independent tables.
- ➔ So first we need to create child tables and then create parent table.
- ➔ Creating Department table and inserting given data into rows of the table is shown below. In this Department table the PRIMARY KEY is “deptId”. It is a child table and there are no foreign keys.

SQL Query:

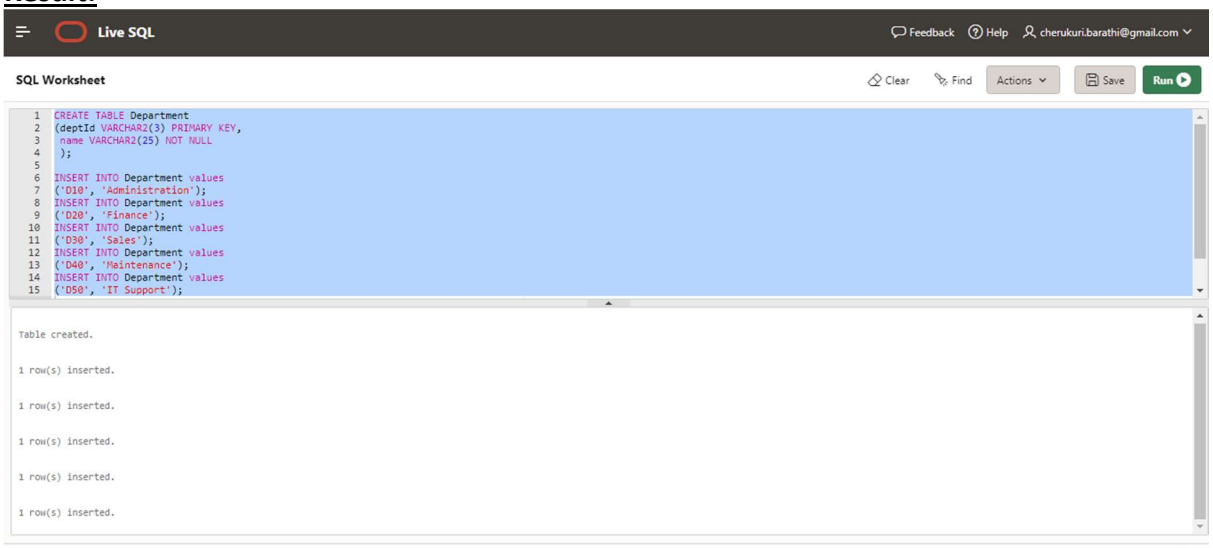
```
CREATE TABLE Department
(deptId VARCHAR2(3) PRIMARY KEY,
 name VARCHAR2(25) NOT NULL
 );

INSERT INTO Department values
('D10', 'Administration');
INSERT INTO Department values
('D20', 'Finance');
INSERT INTO Department values
('D30', 'Sales');
INSERT INTO Department values
('D40', 'Maintenance');
INSERT INTO Department values
('D50', 'IT Support');
```

Solution:

Department table created and 5 rows of data inserted into the Department table.

Result:



SQL Query:

```
select * from Department;
```

Solution:

Printing the Department table to show that the given data is inserted correctly.

Result:

```
16
17  select * from Department;
18
```

DEPTID	NAME
D10	Administration
D20	Finance
D30	Sales
D40	Maintenance
D50	IT Support

Download CSV

5 rows selected.

➔ Creating SalaryGrade table and inserting given data into rows of the table is shown below. In this SalaryGrade table the PRIMARY KEY is “salaryCode”. It is a child table and there are no foreign keys.

SQL Query:

```
CREATE TABLE SalaryGrade
salaryCode VARCHAR2(2) PRIMARY KEY,
startSalary NUMBER(5) NOT NULL,
finishSalary NUMBER(5) NOT NULL
);

INSERT INTO SalaryGrade values
('S1',17000,19000 );
INSERT INTO SalaryGrade values
('S2',19001,24000 );
INSERT INTO SalaryGrade values
('S3',24001,26000 );
INSERT INTO SalaryGrade values
('S4',26001,30000 );
INSERT INTO SalaryGrade values
('S5',30001,39000 );
```

Solution:

SalaryGrade table created and 5 rows of data inserted into the SalaryGrade table.

Result:

SQL Worksheet

```
16  CREATE TABLE SalaryGrade
17  (salaryCode VARCHAR2(2) PRIMARY KEY,
18   startSalary NUMBER(5) NOT NULL,
19   finishSalary NUMBER(5) NOT NULL
20   );
21  INSERT INTO SalaryGrade values
22  ('S1',17000,19000 );
23  INSERT INTO SalaryGrade values
24  ('S2',19001,24000 );
25  INSERT INTO SalaryGrade values
26  ('S3',24001,26000 );
27  INSERT INTO SalaryGrade values
28  ('S4',26001,30000 );
29  INSERT INTO SalaryGrade values
30  ('S5',30001,39000 );
```

```
Table created.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

SQL Query:

```
select * from SalaryGrade;
```

Solution:

Printing the SalaryGrade table to show that the given data is inserted correctly.

Result:

```
32  select * from SalaryGrade;
```

SALARYCODE	STARTSALARY	FINISHSALARY
S1	17000	19000
S2	19001	24000
S3	24001	26000
S4	26001	30000
S5	30001	39000

Download CSV

5 rows selected.

➔ Creating PensionScheme table and inserting given data into rows of the table is shown below. In this PensionScheme table the PRIMARY KEY is “schemeId”. It is a child table and there are no foreign keys.

SQL Query:

```
CREATE TABLE PensionScheme
(schemeId VARCHAR2(4) PRIMARY KEY,
 name char(15) NOT NULL,
 rate NUMBER(2,1) NOT NULL
 );

INSERT INTO PensionScheme values
('S110','AXA',0.5 );
INSERT INTO PensionScheme values
('S121','Premier',0.6);
INSERT INTO PensionScheme values
('S124','Stakeholder',0.4);
INSERT INTO PensionScheme values
('S116','Standard',0.4 );
```

Solution:

PensionScheme table created and 4 rows of data inserted into the PensionScheme table.

Result:

SQL Worksheet

```
33
34  CREATE TABLE PensionScheme
35  (schemeId VARCHAR2(4) PRIMARY KEY,
36   name char(15) NOT NULL,
37   rate NUMBER(2,1) NOT NULL
38   );
39
40  INSERT INTO PensionScheme values
41  ('S110','AXA',0.5 );
42  INSERT INTO PensionScheme values
43  ('S121','Premier',0.6);
44  INSERT INTO PensionScheme values
45  ('S124','Stakeholder',0.4);
46  INSERT INTO PensionScheme values
47  ('S116','Standard',0.4 );
48
```

```
Table created.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

SQL Query:

```
select * from PensionScheme;
```

Solution:

Printing the PensionScheme table to show that the given data is inserted correctly.

Result:

```
49  select * from PensionScheme;
50
51
52
```

SCHEMEID	NAME	RATE
S110	AXA	.5
S121	Premier	.6
S124	Stakeholder	.4
S116	Standard	.4

Download CSV

4 rows selected.

➔ Creating Employee table and inserting given data into rows of the table is shown below. In this Employee table the PRIMARY KEY is “empId”. It is a parent table and there are three foreign keys. “salaryCode” References SalaryGrade table,” deptId” References Department table and “schemeId “ References PensionSchemetable.

SQL Query:

```
CREATE TABLE Employee(
empId VARCHAR(4) NOT NULL PRIMARY KEY,
name VARCHAR2(25) NOT NULL,
adress VARCHAR2(25) NOT NULL,
DOB DATE NOT NULL,
job char(15) NOT NULL,
salaryCode VARCHAR(2) REFERENCES SalaryGrade,
deptId VARCHAR(3) REFERENCES Department,
manager VARCHAR2(4),
schemeId VARCHAR2(4) REFERENCES PensionScheme
);
```

Solution:

Employee table created.

Result:

```
50
51  CREATE TABLE Employee(
52   empId VARCHAR(4) NOT NULL PRIMARY KEY,
53   name VARCHAR2(25) NOT NULL,
54   adress VARCHAR2(25) NOT NULL,
55   DOB DATE NOT NULL,
56   job char(15) NOT NULL,
57   salaryCode VARCHAR(2) REFERENCES SalaryGrade,
58   deptId VARCHAR(3) REFERENCES Department,
59   manager VARCHAR2(4),
60   schemeId VARCHAR2(4) REFERENCES PensionScheme
61   );
62
```

```
Table created.
```

SQL Query:

```
INSERT INTO Employee values
('E101','Keita,j.','1 high street',to_date('06/03/76', 'dd/mm/yy'),
'Clerk','S1','D10','E110','S116');
INSERT INTO Employee values
('E301','Wang,F.','22 railway road',to_date('11/04/80', 'dd/mm/yy'),
'Sales person','S2','D30','E310','S124');
INSERT INTO Employee values
('E310','Flavel,K.','14 crescent road',to_date('25/11/69', 'dd/mm/yy'),
'Manager','S5','D30',' ','S121');
INSERT INTO Employee values
('E501','Payne,J.','7 heap street',to_date('09/02/72', 'dd/mm/yy'),
'Analyst','S5','D50',' ','S121');
INSERT INTO Employee values
('E102','Patel R.','16 glade close',to_date('13/07/74', 'dd/mm/yy'),
'Clerk','S1','D10','E110','S116');
INSERT INTO Employee values
('E110','Smith,B.','199 London road',to_date('22/05/70', 'dd/mm/yy'),
'Manager','S5','D10',' ','S121');
```

Solution:

6 rows of data inserted into the Employee table.

Result:

```
67
68  INSERT INTO Employee values
69  ('E101','Keita,j.','1 high street',to_date('06/03/76', 'dd/mm/yy'),
70  'Clerk','S1','D10','E110','S116');
71  INSERT INTO Employee values
72  ('E301','Wang,F.','22 railway road',to_date('11/04/80', 'dd/mm/yy'),
73  'Sales person','S2','D30','E310','S124');
74  INSERT INTO Employee values
75  ('E310','Flavel,K.','14 crescent road',to_date('25/11/69', 'dd/mm/yy'),
76  'Manager','S5','D30',' ','S121');
77  INSERT INTO Employee values
78  ('E501','Payne,J.','7 heap street',to_date('09/02/72', 'dd/mm/yy'),
79  'Analyst','S5','D50',' ','S121');
80  INSERT INTO Employee values
81  ('E102','Patel R.','16 glade close',to_date('13/07/74', 'dd/mm/yy'),
82  'Clerk','S1','D10','E110','S116');
83  INSERT INTO Employee values
84  ('E110','Smith,B.','199 London road',to_date('22/05/70', 'dd/mm/yy'),
85  'Manager','S5','D10',' ','S121');
86
```

```
Table created.

1 row(s) inserted.

1 row(s) inserted.

1 row(s) inserted.
```

SQL Query:

```
select empId,name,adress,to_char(DOB,'dd/mm/yy'),job,
salaryCode,deptId,manager,schemeId from Employee;
```

Solution:

Printing the Employee table to show that the given data is inserted correctly.

Result:

```
87  select empId,name,adress,to_char(DOB,'dd/mm/yy'),job,
88  salaryCode,deptId,manager,schemeId from Employee;
89
90
```

EMPID	NAME	ADRESS	TO_CHAR(DOB,'DD/MM/YY')	JOB	SALARYCODE	DEPTID	MANAGER	SCHEMEID
E101	Keita,j.	1 high street	06/03/76	Clerk	S1	D10	E110	S116
E301	Wang,F.	22 railway road	11/04/80	Sales person	S2	D30	E310	S124
E310	Flavel,K.	14 crescent road	25/11/69	Manager	S5	D30		S121
E501	Payne,J.	7 heap street	09/02/72	Analyst	S5	D50		S121
E102	Patel R.	16 glade close	13/07/74	Clerk	S1	D10	E110	S116
E110	Smith,B.	199 London road	22/05/70	Manager	S5	D10		S121

Download CSV

6 rows selected.

Part B: Question 2:

a)

SQL Query:

```
select e.name ,S.startSalary,D.deptId FROM Employee E,SalaryGrade
S,Department D
where E.deptId = D.deptId and E.salaryCode = S.salaryCode
ORDER BY E.name ASC ,E.deptId DESC ;
```

Solution:

Resulting table should contain Name (Ascending order), Startsalary and department id of each employee (Descending order)

Result:

```
85  select e.name ,S.startSalary,D.deptId FROM Employee E,SalaryGrade S,Department D
86  where E.deptId = D.deptId and E.salaryCode = S.salaryCode
87  ORDER BY E.name ASC ,E.deptId DESC ;
88
```

NAME	STARTSALARY	DEPTID
Flavel,K.	30001	D30
Keita,j.	17000	D10
Patel R.	17000	D10
Payne,J.	30001	D50
Smith,B.	30001	D10
Wang,F.	19001	D30

Download CSV

6 rows selected.

b)

SQL Query:

```
select s.name as schemename,count(e.empId) from PensionScheme s,employee E
where E.schemeId = S.schemeId group by s.name;
```

Solution:

Resulting table should contain Scheme name and count of employees who joined the scheme.

Result:

```
89  select s.name as schemename,count(e.empId) from PensionScheme s,employee E
90  where E.schemeId = S.schemeId group by s.name;
91
```

SCHEMENAME	COUNT(E.EMPID)
Premier	3
Stakeholder	1
Standard	2

Download CSV

3 rows selected.

c)

SQL Query:

```
select count(e.empId) from employee e ,SalaryGrade s
where e.Job != 'Manager'and e.salaryCode = s.salaryCode
and s.finishSalary > 35000;
```

Solution:

Resulting table should contain count of employees who are not managers but having annual salary of over 35,000£.

Result:

```
90
91  select count(e.empId) from employee e ,SalaryGrade s
92  where e.Job != 'Manager'and e.salaryCode = s.salaryCode
93  and s.finishSalary > 35000;
94
```

COUNT(E.EMPID)
1

Download CSV

d)

SQL Query:

```
select e.empId ,e.name,m.name as manager
from employee e
join employee m
ON (e.manager = m.empId);
```

Solution:

Resulting table should contain employee id and name of each employee with their corresponding manager name.

Result:

```
90  select e.empId ,e.name,m.name as manager
91  from employee e
92  join employee m
93  ON (e.manager = m.empId);
94
```

EMPID	NAME	MANAGER
E301	Wang,F.	Flavel,K.
E101	Keita,j.	Smith,B.
E102	Patel R.	Smith,B.

Download CSV

3 rows selected.

Part C: Question 1:

a) SQL statement to create the given table

➔ Creating an SQL table in Database:

```
CREATE TABLE Sales(
OrderNo NUMBER(5) NOT NULL,
ProductNo NUMBER(3) NOT NULL,
Price NUMBER(5,2) NOT NULL,
Quantity NUMBER(3) NOT NULL,
Sales NUMBER(7,2) NOT NULL,
MonthId NUMBER(2) NOT NULL CHECK (MonthId >= 1 and MonthId <=12),
YearId NUMBER(4) NOT NULL
);
```

➔ Inserting given rows into the Sales table

```
INSERT INTO Sales values
(10107,2,95.7,30,2871,2,2003);
INSERT INTO Sales values
(10107,5,99.91,39,3896.49,2,2003);
INSERT INTO Sales values
(10110,9,86.13,29,2497.77,2,2003);
INSERT INTO Sales values
(10121,5,81.35,34,2765.9,11,2003);
INSERT INTO Sales values
(10134,2,94.74,41,3884.34,7,2004);
INSERT INTO Sales values
(10134,5,100,27,3307.77,7,2004);
INSERT INTO Sales values
(10159,14,100,49,5205.27,10,2005);
INSERT INTO Sales values
(10161,9,86.13,29,2497.77,10,2005);
INSERT INTO Sales values
(10163,14,100,20,2000,10,2005);
INSERT INTO Sales values
(10168,1,96.66,36,3479.76,10,2006);
INSERT INTO Sales values
(10180,12,100,42,4695.6,11,2006);
```

➔ Display the table

```
select * from sales;
```

b) Select statement for fetching the no of products which were sold in each month of each year for each product.

```
SELECT productno,sum(Quantity),monthId,yearId
from sales
group by productNo,monthId,yearId
order by yearId ASC,monthId ASC;
```

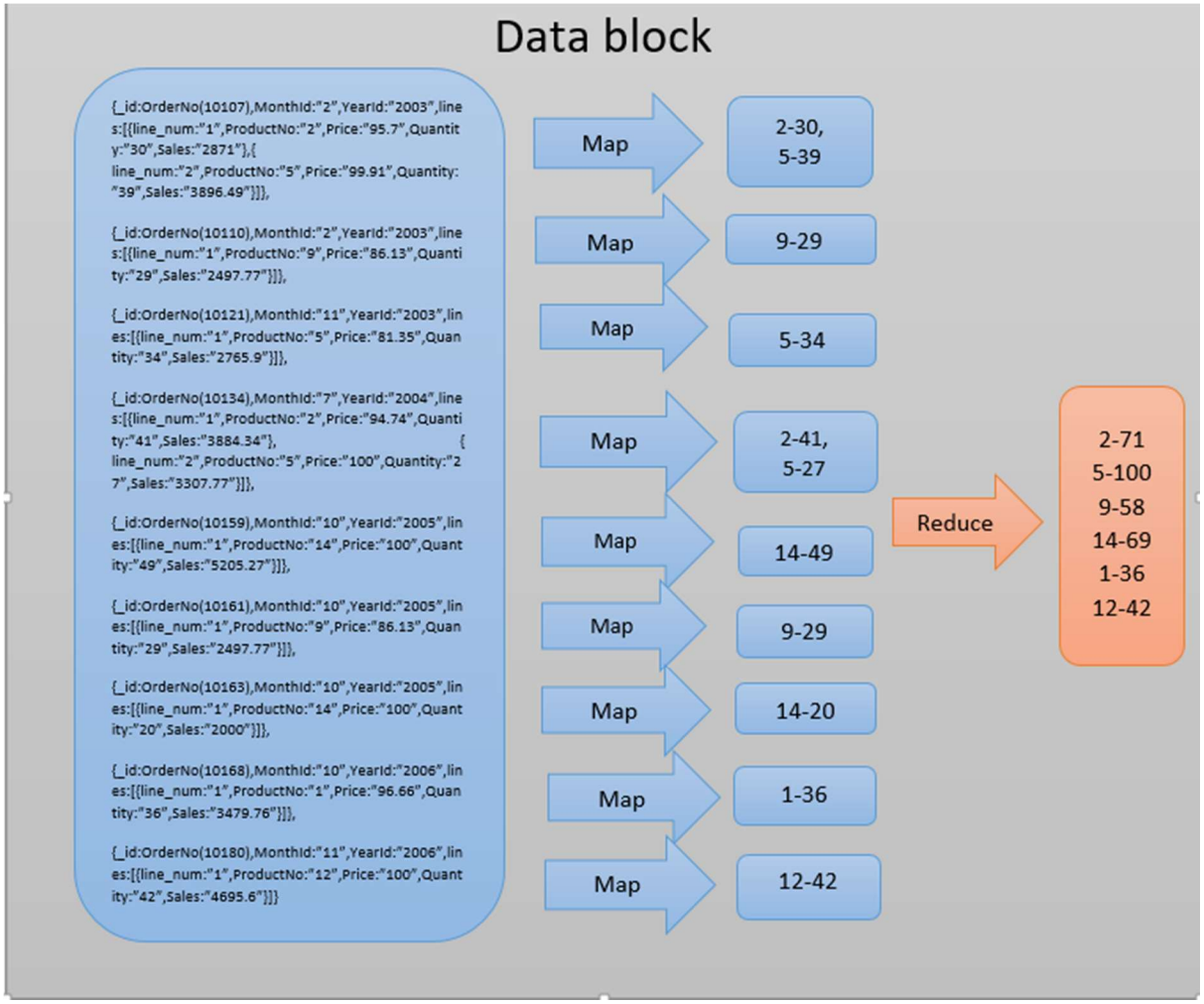
Part C: Question 2:

Created a Data block from the given sales table and applied MapReduce on the data block.

Above diagram shows the Data block representation of given Sales data.

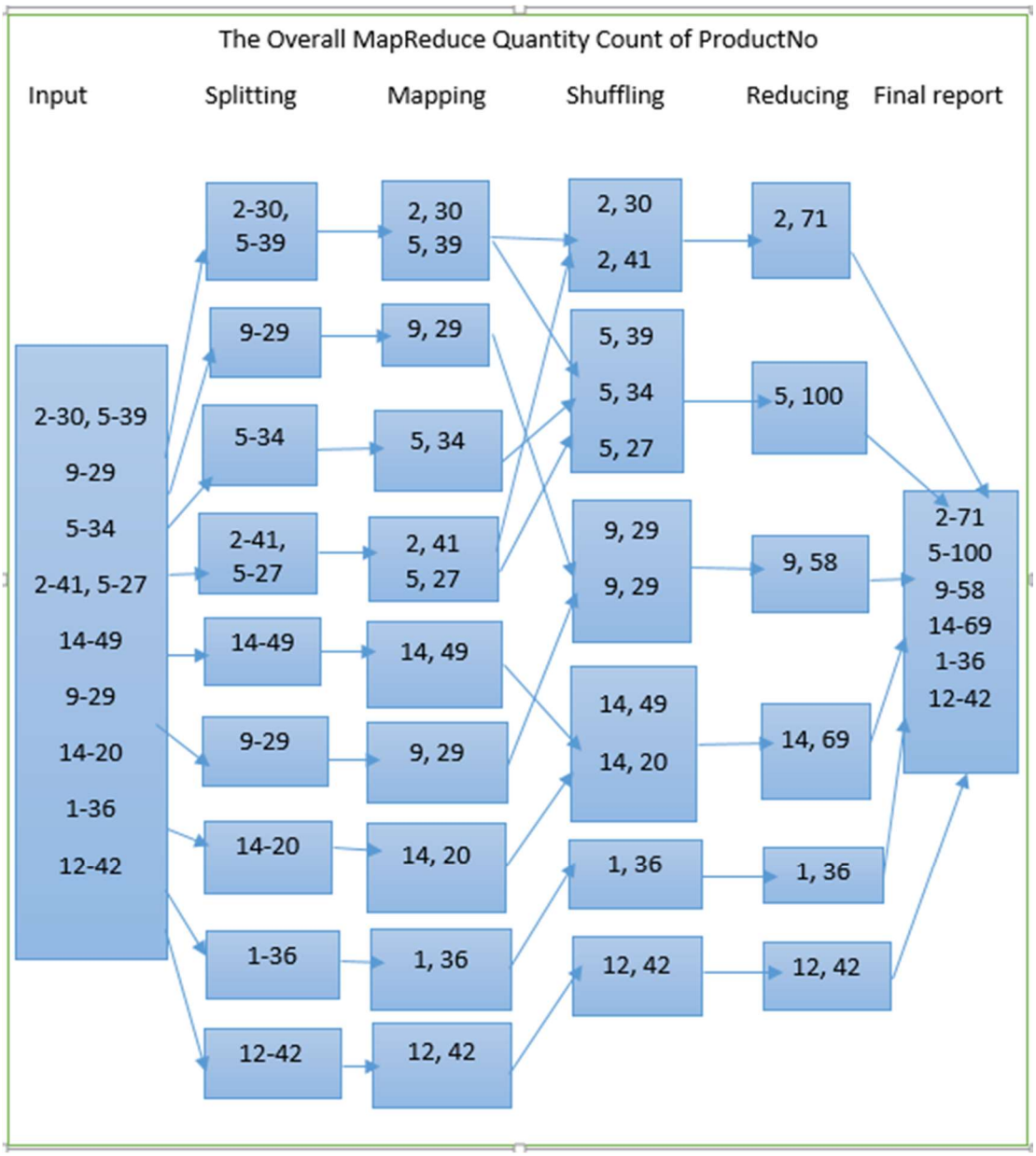
This is an illustration of MapReduce that determines the total Quantity of each product (indicated by ProductNo) that has been sold (this will help determine the (key, value) pairs).

The data is stored as (key, value) pairs, with the order number (OrderNo) as the initial key and the remainder of the invoice data as value.
Example: for the first entry in the list:
Key = OrderNo (10107)
Value =
MonthId:"2",YearId:"2003",lines:[{line_num:"1",ProductNo:"2",Price:"95.7",Quantity:"30",Sales:"2871"},{line_num:"2",ProductNo:"5",Price:"99.91",Quantity:"39",Sales:"3896.49"}]
Below picture shows the Map and Reduce operation on the given data block.



The map function parses each order to find data about Quantity of the product sold on that order. Since the purpose is to find the total quantity of each product that has been sold, the focus is on ProductNo and Quantity. The result of the map function is a new list of (key, value) pairs in which the Product Code (ProductNo) is the key and the Quantity of product (Quantity) is the value: (ProductNo, Quantity).An example is: (2, 30).

The shuffle function is shown in the below diagram.



Example.

(Key, value) = (2, 30) and
(Key, value) = (2, 41) are grouped together.
The reduce function then takes that list of (key, value) pairs and combines them by summing Up the values associated with each key (ProductNo) to produce the summary result.
Example:
The key 2 will be associated with 71 as the sum of the values 30 and 41 above. The final 0utput for this key is (2, 71)

Part D: Research Report

Strengths of Relational databases:

There are many advantages of using relational database over any other type of database, a relational database helps in maintaining the data integrity, data accuracy, reduces data redundancy to minimum or zero, data scalability, data flexibility and facilitates makes it easy to implement security methods. Above all, a Relational Database Management system is a simpler database model, both to design and implement.

Limitations of Relational databases:

There are few limitations of relational database. They are single CPU, maximum memory should be used by 1GB RAM, maximum number of SQL server Express is 16 on machine.The fields that is present on a relational database is with limitations. Limitations is essence means that it cannot accommodate more information. Despite if more information are provided, it may lead to data loss.

Inorder to overcome the limitations of relational databases, we can move to NoSQL databases. NoSQL databases provides various advantages over traditional relational databases. Due to Schema less, Scalability, High availability, Flexibility, cost effective so many relational databases are migrating their data to NoSQL databases. There are four main types of NoSQL databases: Key-Value databases, Column-based databases, Document databases, Graph-oriented Databases. There are many popular NoSQL databases available in the market. They are Riak, Redis, Casandra, Hbase, MongoDB, CouchDB, Neo4j etc.

Mongo DB is a document-oriented database which helps in grouping data more logically. These databases store documents on the part of “key-value”. These may be stored in XML, JSON, or BSON formats. The document is not well defined before the creation of unstructured data in MongoDB. The MongoDB has lots of extensive built-in-features and is highly compatible with other systems, as we use JSON query to access data we can say that MongoDB is extensive and flexible. MongoDB is highly scalable, so it became the most popular in the world.

There are more than 570 cities in 50 countries served by Craigslist, an ad posting community. With 1.5 million new ads posted every day, craigslist must archive billions of records in many different formats, Querying and reporting on these archives must be possible at runtime. Due to management costs and limitations in flexibility, Craigslist stores its information in a MySQL cluster, it hindered the use of relational databases in the future. For its scalability and flexibility, Craigslist migrated over two billion documents to MongoDB in 2011.The archive on Craigslist is stored using MySQL, which is one of the only options before. Data from the live database was copied to the archive system by the original Craigslist archive application. Relational databases are flexible, but they have limited flexibility and long delays because live database changes need to be propagated to archives. Craigslist had difficulty moving billions of rows of data to the archive after making changes to their MySQL cluster. The production database would be clogged with archive-ready data.

There was a decline in performance on the live database. The company needs to eliminate further impediments to its growth and serve its customers so that it can continue to grow and thrive. Several alternative explanations were considered by the team. Craigslist chose MongoDB because of the database's built-in scalability options. A collection or set of documents or other items that make up a single group can be called a document set. If an application's schema changes frequently, MongoDB can accommodate the changes without forcing the application to undergo costly data migrations. In addition, MongoDB's support for auto-sharding and high availability eased operational pain points for Craigslist. MongoDB enabled Craigslist to scale horizontally across commodity hardware without having to write and maintain complex, custom sharding code. Using auto-sharding, Craigslist's initial MongoDB deployment was designed to hold over 5 billion documents and 10TB of data. MongoDB concepts and features are almost equal. In many respects, to relational databases so Craigslist's developers found the transition seamless.

References:

https://www.educba.com/relational-database-advantages/

https://www.quora.com/What-are-the-limitations-of-SQL

https://www.hitechwhizz.com/2021/04/6-advantages-and-disadvantages-limitations-benefits-of-relational-database.html

https://www.linkedin.com/pulse/mongodb-real-world-industry-usecases-devendra-kanade

https://core.ac.uk/download/pdf/231153824.pdf