Obot A. Enomfon - 14243972

and

Roxanne K. Rugema – 9255858

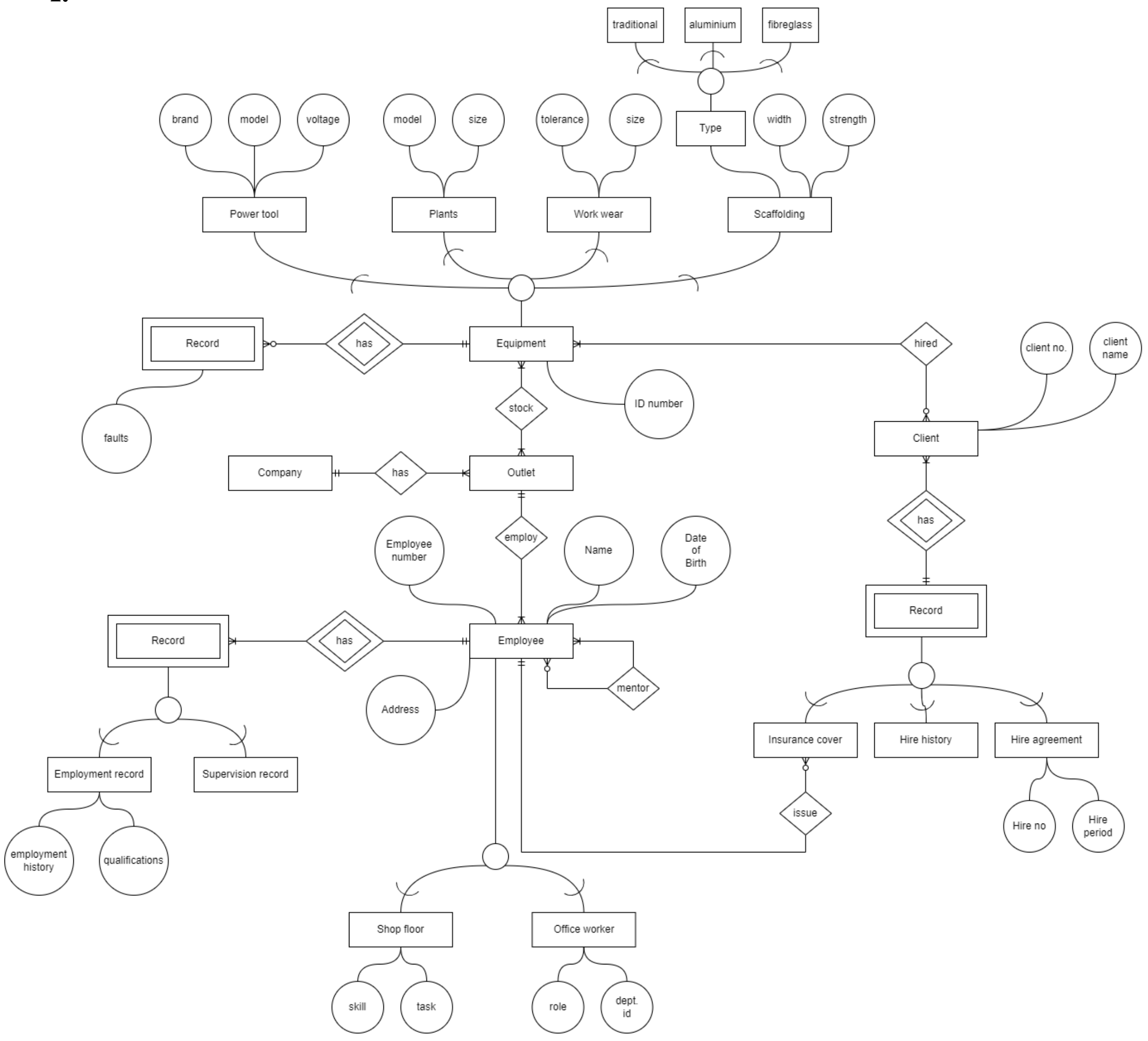
7086CEM - Data Management Systems

7086 Coursework

Faculty of Engineering, Environment and Computing, Coventry University
MSc Data Science/Data Science and Computational Intelligence
Coventry, England

Part A

I.



II.

EQUIPMENT (<u>EquipmentID,</u> EquipmentType)
POWER_TOOLS (<u>EquipmentID</u>, Brand, Model, Voltage)
PLANTS (<u>EquipmentID</u>, Model, Size)
SCAFFOLDING (<u>EquipmentID</u>, Scaffolding_type, Strength, Width)
WORK_WEAR (<u>EquipmentID</u>, Tolerance, Size)

Justification: To ensure that the relational tables generated are in 3NF, individual tables were created for each subtype of equipment and as seen in the table generated above, they inherit the primary key of the supertype – Equipment. To distinguish between types a new attribute -EquipmentType- is added to the supertype. This attribute will be set to "P" ,"S", "PT" or "WW", when an equipment is added to the supertype, and to the subtype.

EMPLOYEE (<u>EmploymentID</u>, Name, Date_of_Birth, Address, Qualification, History, EmployeeType)
SHOP_FLOOR (<u>EmploymentID</u>, Skill, Task)
OFFICE (<u>EmploymentID</u>, Role, Department)

Justification: To ensure that the relational tables generated are in 3NF, individual tables were created for each subtype of employees and as seen in the table generated above, they inherit the primary key of the supertype – Employee. To distinguish between types a new attribute -EmployeeType- is added to the supertype. This attribute will be set to “O” or “S”, when an employee is added to the supertype, and to the subtype.

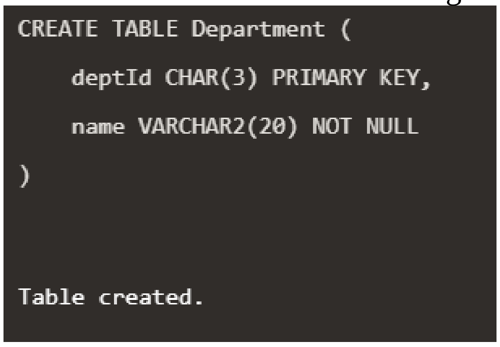
CLIENT (<u>Client_ID</u>, Name, Hire_no*, Hire_history)
HIRE_AGREEMENT (Client_ID, <u>Hire_no</u>, Hire_period)

Justification: To ensure that the relational tables generated are in 3NF, individual tables were created for Client and Hire Agreement. The Hire number which is the primary key of the Hire Agreement table was added to the Client table as a foreign key so that the Hire agreement table which carries the hire period is easily accessible from the Client table.

Part B

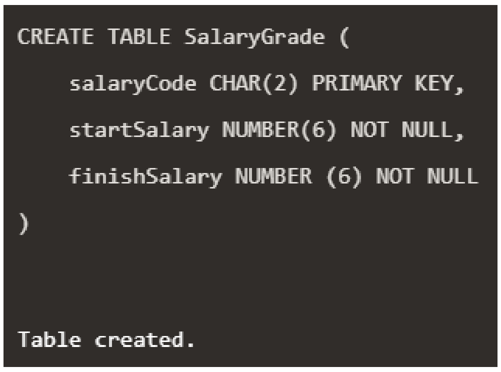
1. First, we created the tables for Department, Salary Grade and Pension Scheme as they had foreign keys in the Employee table, therefore needed to exist before creating that table.

Below is the Statement for creating the Department table where deptId is labelled the PRIMARY KEY and a NOT NULL constraint on the name attribute as all employees must have their name recorded according to the data provided in the table.



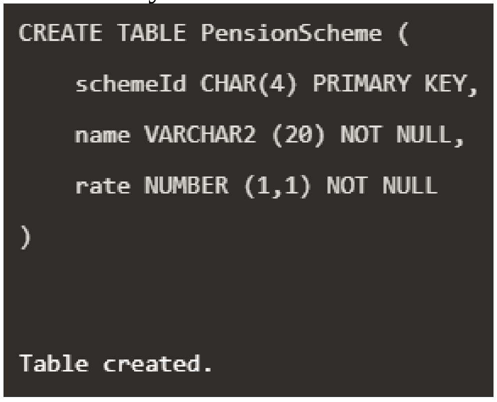
```
CREATE TABLE Department (

    deptId CHAR(3) PRIMARY KEY,

    name VARCHAR2(20) NOT NULL

)



Table created.
```

Next is the SalaryGrade table, in which the salaryCode is the PRIMARY CODE



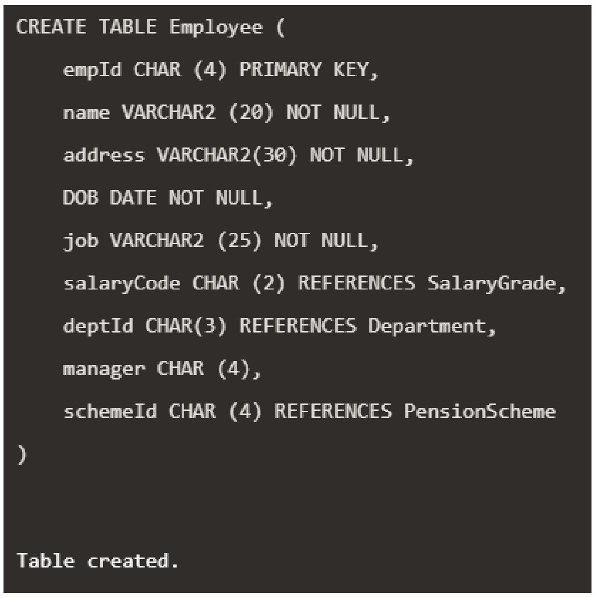
```
CREATE TABLE SalaryGrade (

    salaryCode CHAR(2) PRIMARY KEY,

    startSalary NUMBER(6) NOT NULL,

    finishSalary NUMBER (6) NOT NULL

)



Table created.
```

Followed by the PensionScheme in which the PRIMARY KEY is the schemeId.



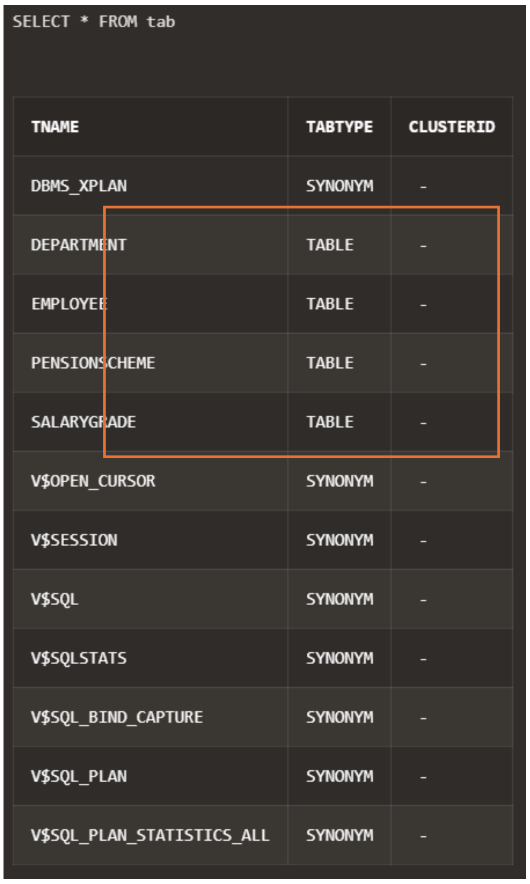
```
CREATE TABLE PensionScheme (

    schemeId CHAR(4) PRIMARY KEY,

    name VARCHAR2 (20) NOT NULL,

    rate NUMBER (1,1) NOT NULL

)



Table created.
```

Lastly for the table creation queries, the Employee table was created where the PRIMARY KEY is the empId and the salaryCode, deptId and schemeId as foreign keys here, all reference back to the tables where they are primary keys.



```
CREATE TABLE Employee (

    empId CHAR (4) PRIMARY KEY,

    name VARCHAR2 (20) NOT NULL,

    address VARCHAR2(30) NOT NULL,

    DOB DATE NOT NULL,

    job VARCHAR2 (25) NOT NULL,

    salaryCode CHAR (2) REFERENCES SalaryGrade,

    deptId CHAR(3) REFERENCES Department,

    manager CHAR (4),

    schemeId CHAR (4) REFERENCES PensionScheme

)



Table created.
```

The screenshot below shows that these 4 new tables now exist in the session:

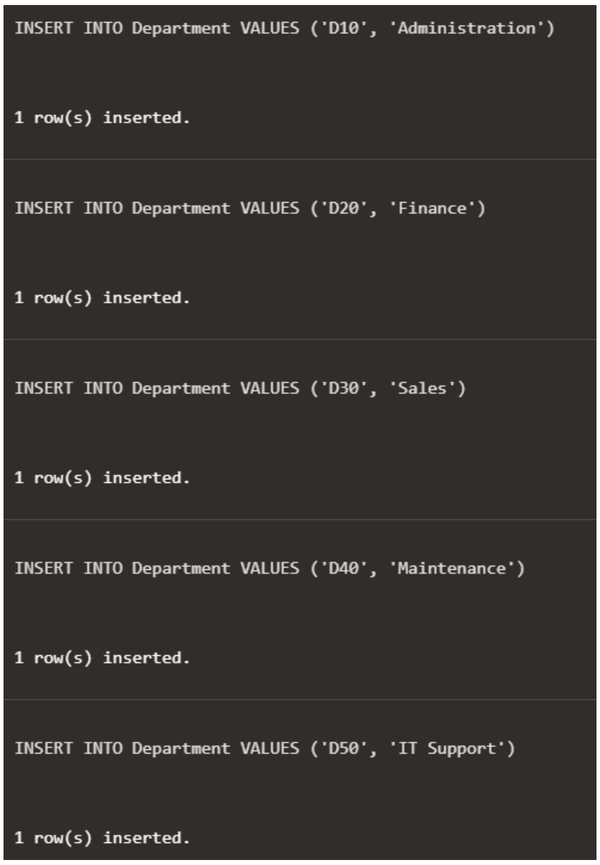


```
SELECT * FROM tab
```

TNAME	TABTYPE	CLUSTERID
DBMS_XPLAN	SYNONYM	-
DEPARTMENT	TABLE	-
EMPLOYEE	TABLE	-
PENSIONSCHEME	TABLE	-
SALARYGRADE	TABLE	-
V$OPEN_CURSOR	SYNONYM	-
V$SESSION	SYNONYM	-
V$SQL	SYNONYM	-
V$SQLSTATS	SYNONYM	-
V$SQL_BIND_CAPTURE	SYNONYM	-
V$SQL_PLAN	SYNONYM	-
V$SQL_PLAN_STATISTICS_ALL	SYNONYM	-

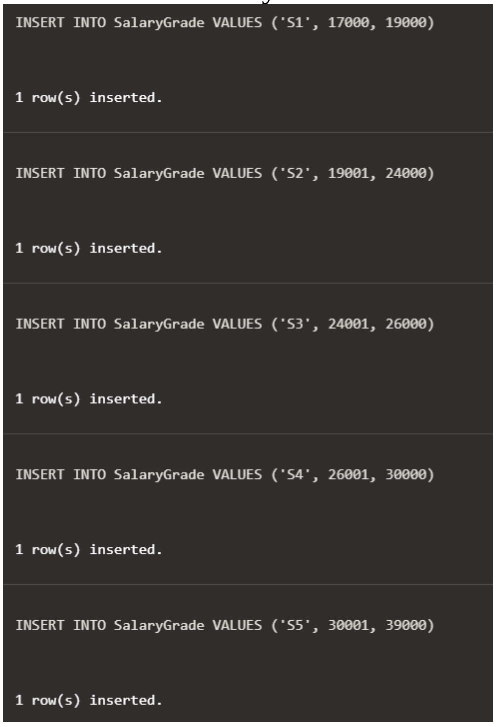
The next step was to add the data values into the tables.

The values for the Departments table we added as follows:



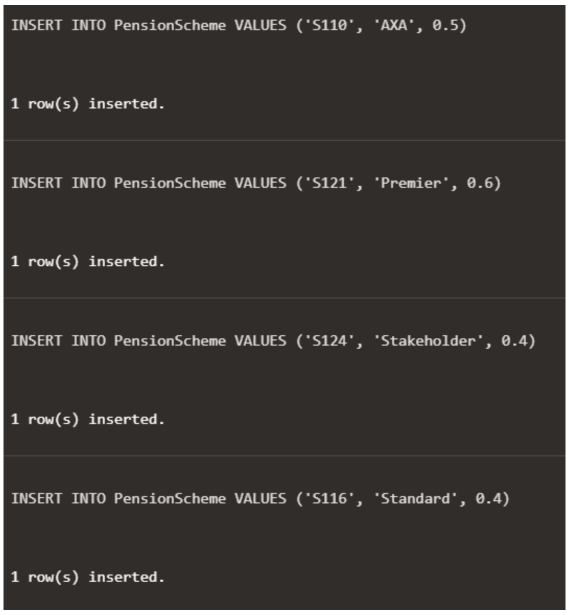
```
INSERT INTO Department VALUES ('D10', 'Administration')

1 row(s) inserted.

INSERT INTO Department VALUES ('D20', 'Finance')

1 row(s) inserted.

INSERT INTO Department VALUES ('D30', 'Sales')

1 row(s) inserted.

INSERT INTO Department VALUES ('D40', 'Maintenance')

1 row(s) inserted.

INSERT INTO Department VALUES ('D50', 'IT Support')

1 row(s) inserted.
```

The values for SalaryGrade were added as follows:



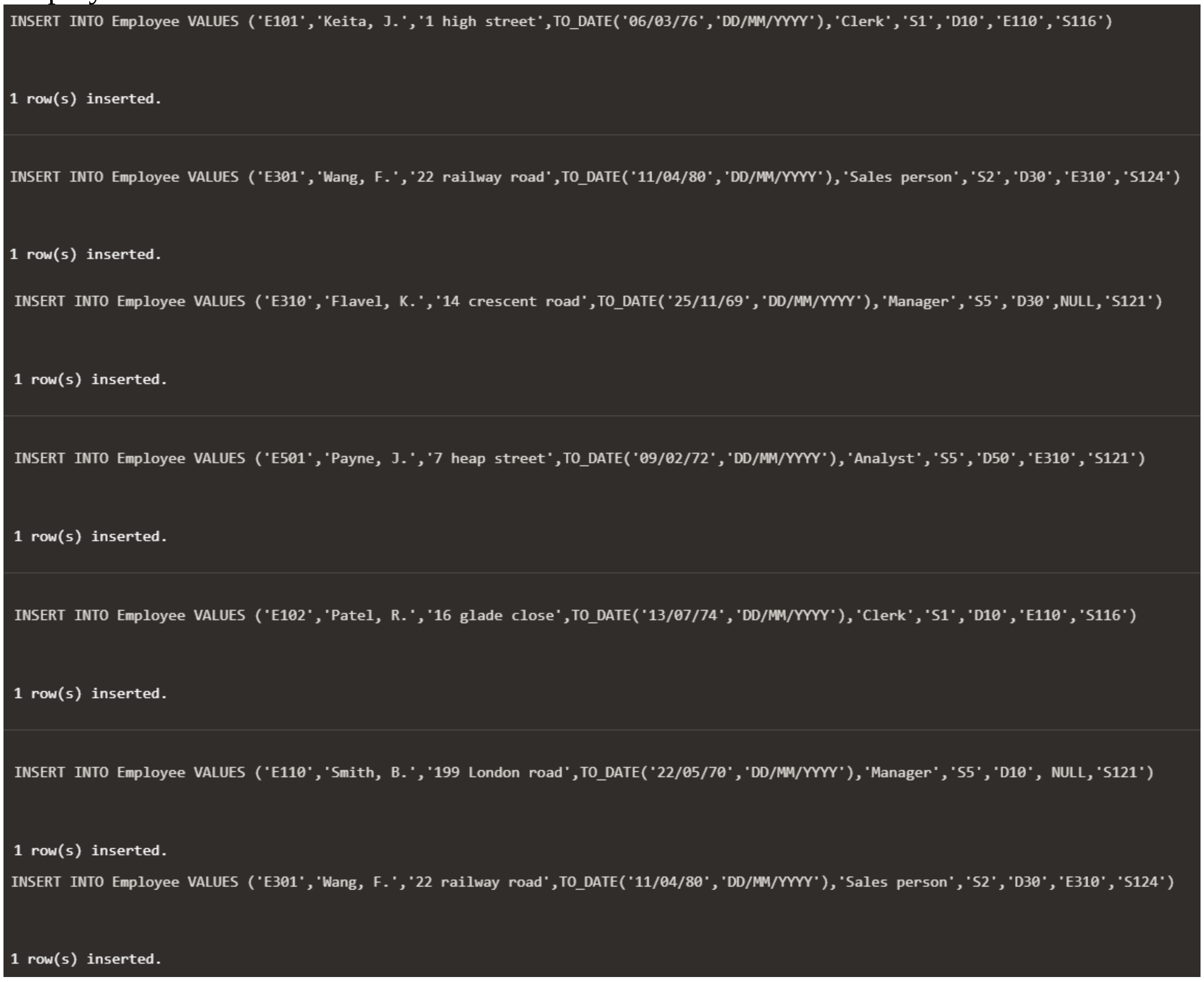
```
INSERT INTO SalaryGrade VALUES ('S1', 17000, 19000)

1 row(s) inserted.

INSERT INTO SalaryGrade VALUES ('S2', 19001, 24000)

1 row(s) inserted.

INSERT INTO SalaryGrade VALUES ('S3', 24001, 26000)

1 row(s) inserted.

INSERT INTO SalaryGrade VALUES ('S4', 26001, 30000)

1 row(s) inserted.

INSERT INTO SalaryGrade VALUES ('S5', 30001, 39000)

1 row(s) inserted.
```

PensionScheme data was added with the following statements:



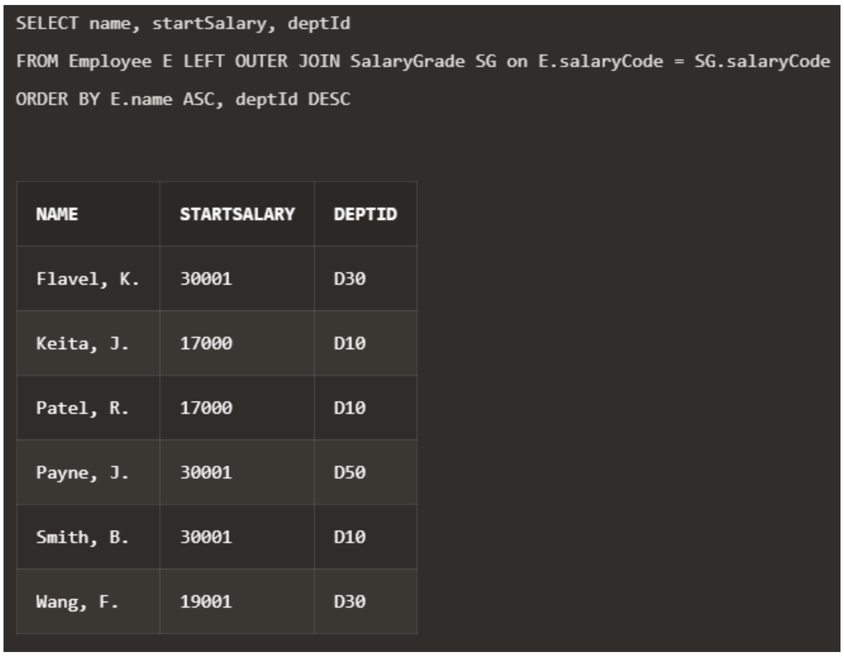
```
INSERT INTO PensionScheme VALUES ('S110', 'AXA', 0.5)

1 row(s) inserted.

INSERT INTO PensionScheme VALUES ('S121', 'Premier', 0.6)

1 row(s) inserted.

INSERT INTO PensionScheme VALUES ('S124', 'Stakeholder', 0.4)

1 row(s) inserted.

INSERT INTO PensionScheme VALUES ('S116', 'Standard', 0.4)

1 row(s) inserted.
```

Employee values were added as follows:



```
INSERT INTO Employee VALUES ('E101','Keita, J.','1 high street',TO_DATE('06/03/76','DD/MM/YYYY'),'Clerk','S1','D10','E110','S116')

1 row(s) inserted.

INSERT INTO Employee VALUES ('E301','Wang, F.','22 railway road',TO_DATE('11/04/80','DD/MM/YYYY'),'Sales person','S2','D30','E310','S124')

1 row(s) inserted.
INSERT INTO Employee VALUES ('E310','Flavel, K.','14 crescent road',TO_DATE('25/11/69','DD/MM/YYYY'),'Manager','S5','D30',NULL,'S121')

1 row(s) inserted.

INSERT INTO Employee VALUES ('E501','Payne, J.','7 heap street',TO_DATE('09/02/72','DD/MM/YYYY'),'Analyst','S5','D50','E310','S121')

1 row(s) inserted.

INSERT INTO Employee VALUES ('E102','Patel, R.','16 glade close',TO_DATE('13/07/74','DD/MM/YYYY'),'Clerk','S1','D10','E110','S116')

1 row(s) inserted.

INSERT INTO Employee VALUES ('E110','Smith, B.','199 London road',TO_DATE('22/05/70','DD/MM/YYYY'),'Manager','S5','D10', NULL,'S121')

1 row(s) inserted.
INSERT INTO Employee VALUES ('E301','Wang, F.','22 railway road',TO_DATE('11/04/80','DD/MM/YYYY'),'Sales person','S2','D30','E310','S124')

1 row(s) inserted.
```

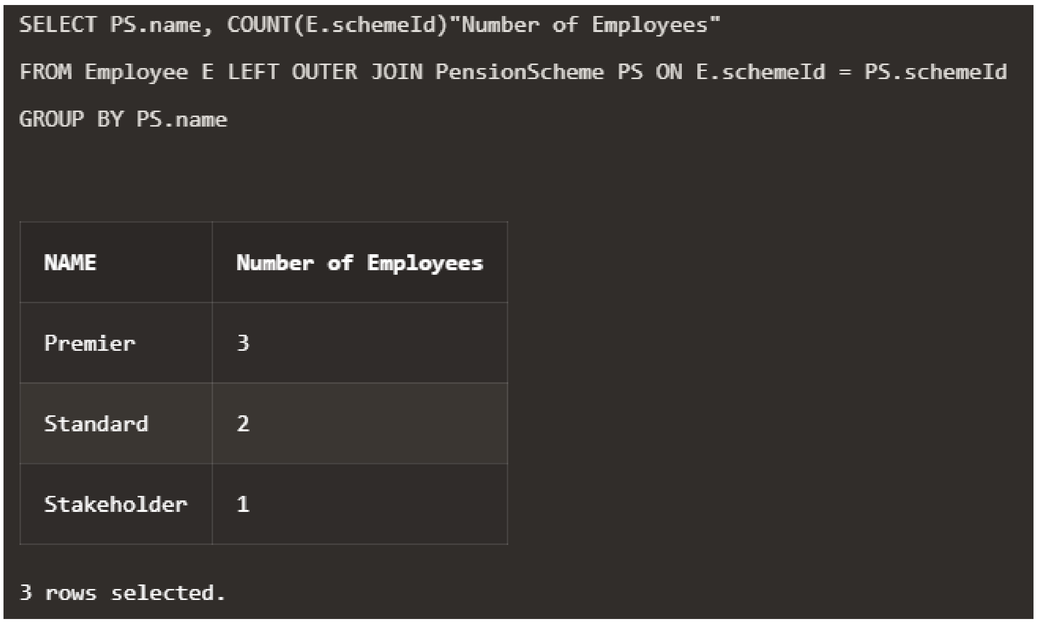
1. The following Statements and outputs were obtained for the queries in this section.
   a) After joining the SalaryGrade table and Employee table to get the the start salary corresponding to each employee by name as well as deptId, the ORDER description was used to sort the values by name in ascending order.



```
SELECT name, startSalary, deptId
FROM Employee E LEFT OUTER JOIN SalaryGrade SG on E.salaryCode = SG.salaryCode
ORDER BY E.name ASC, deptId DESC
```

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

b) The number of employees on each pension scheme is given through the below statement, where the Employee table was joined to the PensionSceme table to link the scheme Id's to the scheme names in the PS table.

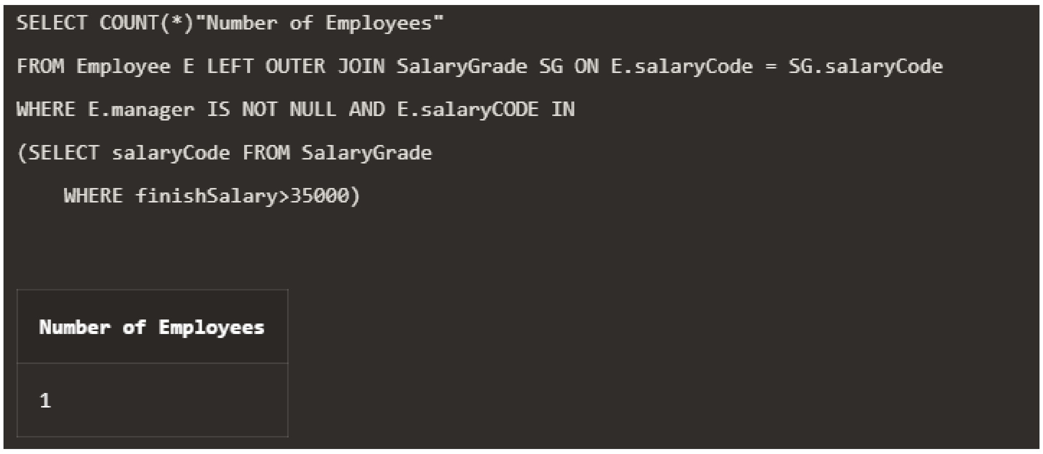


```
SELECT PS.name, COUNT(E.schemeId)"Number of Employees"
FROM Employee E LEFT OUTER JOIN PensionScheme PS ON E.schemeId = PS.schemeId
GROUP BY PS.name
```

NAME	Number of Employees
Premier	3
Standard	2
Stakeholder	1

```
3 rows selected.
```

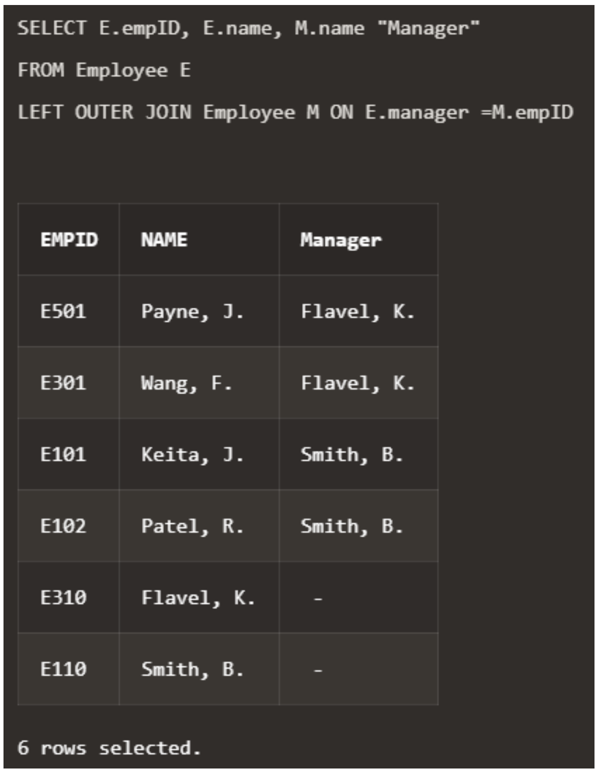
c) The number of non managerial employees earning over 35K is found through the statement below, where the NOT NULL constraint is used to exclude managers in the employee table from the search. Employee is joined to the SalaryGrade table where the finishing salaries are stored. The WHERE statement is used with the 'more than' constraint to restrict the output to values where the finish salary is above 35K.



```
SELECT COUNT(*)"Number of Employees"
FROM Employee E LEFT OUTER JOIN SalaryGrade SG ON E.salaryCode = SG.salaryCode
WHERE E.manager IS NOT NULL AND E.salaryCODE IN
(SELECT salaryCode FROM SalaryGrade
    WHERE finishSalary>35000)
```

Number of Employees
1

d) The list of employees and their managers was created through the following statement, where Employee table was left joined from the manager attribute to the employee ID attribute to match the ID's in the manager column and output their names instead. As seen the last two employees contain null values in this output table as they are the managers.



```
SELECT E.empID, E.name, M.name "Manager"
FROM Employee E
LEFT OUTER JOIN Employee M ON E.manager =M.empID
```

EMPID	NAME	Manager
E501	Payne, J.	Flavel, K.
E301	Wang, F.	Flavel, K.
E101	Keita, J.	Smith, B.
E102	Patel, R.	Smith, B.
E310	Flavel, K.	-
E110	Smith, B.	-

6 rows selected.

Part C

1a.
```
CREATE TABLE FlightRecord (
    year_of_flight INT CHECK (year_of_flight BETWEEN 2000 AND 2020),
    month_of_flight INT CHECK (month_of_flight BETWEEN 1 AND 12),
    day_of_month INT CHECK (day_of_month BETWEEN 1 AND 31),
    day_of_week INT CHECK (day_of_week BETWEEN 1 AND 7),
    departure_time INT CHECK (departure_time BETWEEN 0 AND 2359),
    actual_departure_time INT CHECK (actual_departure_time BETWEEN 0 AND 2359),
    arrival_time INT CHECK (arrival_time BETWEEN 0 AND 2359),
    airline_carrier VARCHAR(10) NOT NULL,
    flight_number VARCHAR(10) NOT NULL,
    departure_delay INT CHECK (departure_delay >= 0),
    arrival_delay INT CHECK (arrival_delay >= 0),
    weather_delay INT CHECK (weather_delay >= 0),
    PRIMARY KEY (airline_carrier, flight_number)
    );
    INSERT INTO FlightRecord VALUES (2015, 4, 20, 5, 1430, 1400, 1820, '131',
    'JL729', 30, 15, 0);
```

Justification: The check constraint was introduced to ensure that the values entered are within the set ranges. In addition, the primary key must be a composite key consisting of the airline_carrier and flight_number to easily identify a particular flight.

b.
```
SELECT airline_carrier, COUNT(*) AS total_delayed_flights
FROM FlightRecord
WHERE departure_delay > 0 OR arrival_delay > 0 OR weather_delay > 0
GROUP BY airline_carrier;
```

Justification: To determine the number of flights which were delayed for each carrier a count function as well as a condition to check the values for all the different delays needed to be introduced.

2. DATA BLOCK

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘131’, **Flight Number**: ‘JL729’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘19’, **Weekday**: ‘4’, **Departure Time**: ‘1120’, **Actual_Departure_Time**: ‘1100’, **Arrival Time**: ‘1520’, **Airline Carrier**: ‘131’, **Flight Number**: ‘JL730’, **Departure Delay**: ‘20’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1010’, **Actual_Departure_Time**: ‘1000’, **Arrival Time**: ‘1230’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘10’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘18’, **Weekday**: ‘3’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL729’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1600’, **Actual_Departure_Time**: ‘1600’, **Arrival Time**: ‘1950’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL730’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘10’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘23’, **Weekday**: ‘1’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1430’, **Arrival Time**: ‘1750’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL732’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘17, **Weekday**: ‘2’, **Departure Time**: ‘0930’, **Actual_Departure_Time**: ‘0900’, **Arrival Time**: ‘1320’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL737’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘05’}

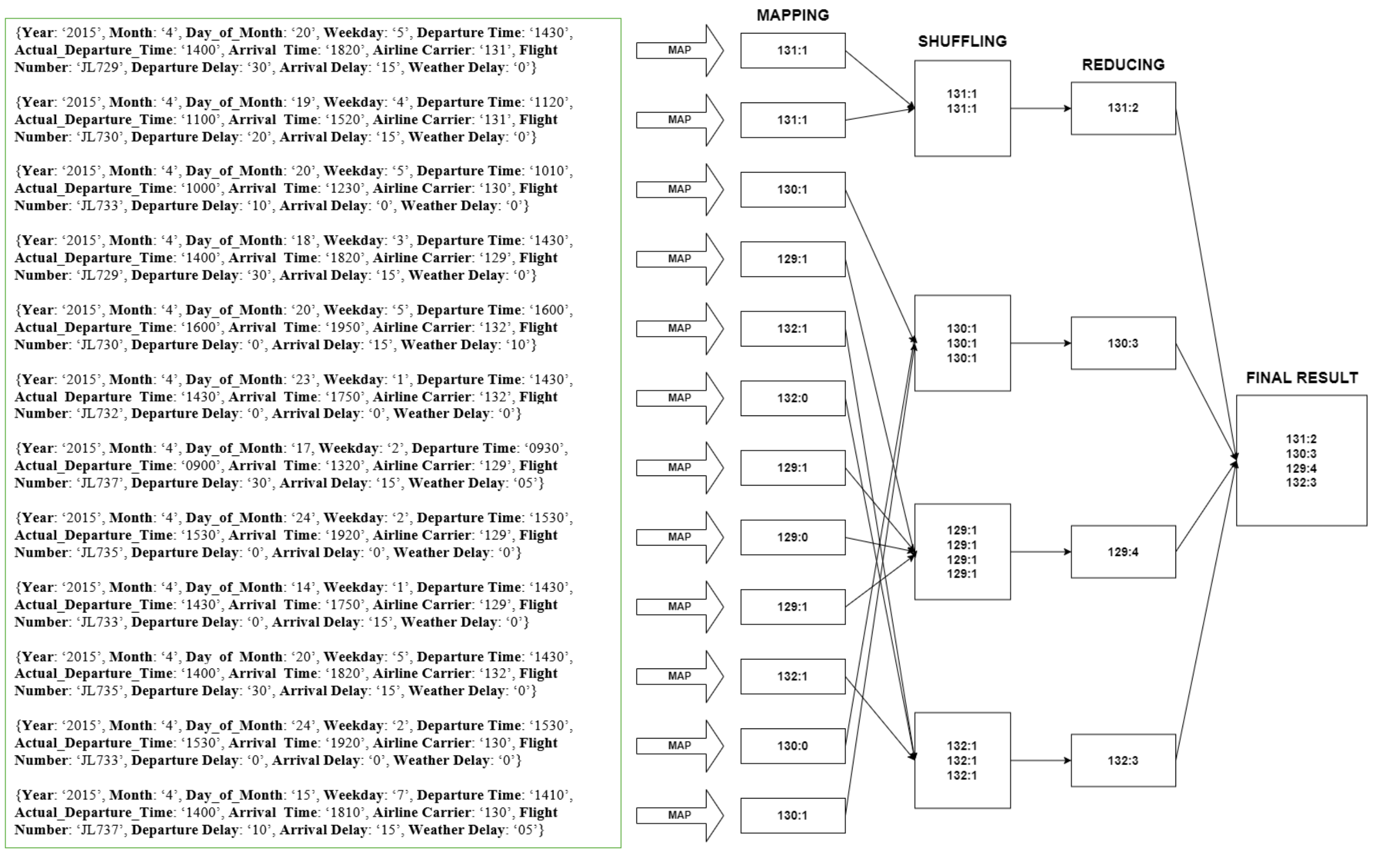
{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘24’, **Weekday**: ‘2’, **Departure Time**: ‘1530’, **Actual_Departure_Time**: ‘1530’, **Arrival Time**: ‘1920’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL735’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘14’, **Weekday**: ‘1’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1430’, **Arrival Time**: ‘1750’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL735’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘24’, **Weekday**: ‘2’, **Departure Time**: ‘1530’, **Actual_Departure_Time**: ‘1530’, **Arrival Time**: ‘1920’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘15’, **Weekday**: ‘7’, **Departure Time**: ‘1410’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1810’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL737’, **Departure Delay**: ‘10’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘05’}



{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘131’, **Flight Number**: ‘JL729’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘19’, **Weekday**: ‘4’, **Departure Time**: ‘1120’, **Actual_Departure_Time**: ‘1100’, **Arrival Time**: ‘1520’, **Airline Carrier**: ‘131’, **Flight Number**: ‘JL730’, **Departure Delay**: ‘20’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1010’, **Actual_Departure_Time**: ‘1000’, **Arrival Time**: ‘1230’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘10’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘18’, **Weekday**: ‘3’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL729’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1600’, **Actual_Departure_Time**: ‘1600’, **Arrival Time**: ‘1950’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL730’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘10’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘23’, **Weekday**: ‘1’, **Departure Time**: ‘1430’, **Actual Departure Time**: ‘1430’, **Arrival Time**: ‘1750’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL732’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘17, **Weekday**: ‘2’, **Departure Time**: ‘0930’, **Actual_Departure_Time**: ‘0900’, **Arrival Time**: ‘1320’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL737’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘05’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘24’, **Weekday**: ‘2’, **Departure Time**: ‘1530’, **Actual_Departure_Time**: ‘1530’, **Arrival Time**: ‘1920’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL735’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘14’, **Weekday**: ‘1’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1430’, **Arrival Time**: ‘1750’, **Airline Carrier**: ‘129’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day of Month**: ‘20’, **Weekday**: ‘5’, **Departure Time**: ‘1430’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1820’, **Airline Carrier**: ‘132’, **Flight Number**: ‘JL735’, **Departure Delay**: ‘30’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘24’, **Weekday**: ‘2’, **Departure Time**: ‘1530’, **Actual_Departure_Time**: ‘1530’, **Arrival Time**: ‘1920’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL733’, **Departure Delay**: ‘0’, **Arrival Delay**: ‘0’, **Weather Delay**: ‘0’}

{**Year**: ‘2015’, **Month**: ‘4’, **Day_of_Month**: ‘15’, **Weekday**: ‘7’, **Departure Time**: ‘1410’, **Actual_Departure_Time**: ‘1400’, **Arrival Time**: ‘1810’, **Airline Carrier**: ‘130’, **Flight Number**: ‘JL737’, **Departure Delay**: ‘10’, **Arrival Delay**: ‘15’, **Weather Delay**: ‘05’}

For the **first entry** in the list:
Key = Flight Number : ‘JL729’
Value = Year: ‘2015’, Month: ‘4’, Day_of_Month: ‘20’, Weekday: ‘5’, Departure Time: ‘1430’, Actual_Departure_Time: ‘1400’, Arrival Time: ‘1820’, Airline Carrier: ‘131’, Departure Delay: ‘30’, Arrival Delay: ‘15’, Weather Delay: ‘0’
Note: For a delay to be counted, the condition Departure delay > 0 or Arrival delay > 0 or Weather delay > 0 must be met. Where all values equal 0 no delay is counted.

The **Map function** breaks down each **Flight Record** to find data about the flight details linked to that flight number. Since the purpose is to find the number of flights which were delayed for each carrier, the focus is on Airline Carrier and Delays. The result of the map function is a new list of (key, value) pairs in which the Airline carrier is the key and the delays the value: (Airline carrier, delays).

The **Shuffle function** groups together all the (key, value) pairs that have the same the key. Hence, as shown in the diagram above, (key, value) = (130, 1) and (key, value) = (130, 1) were grouped together.

The **Reduce function** takes that list of (key, value) pairs and combines them by adding up the values associated with each key (Airline carrier) to produce the summary result.
Example:
The key 130 will relate to 3 as the sum of the value 1 in three places. The final output for this key is (130, 3)

Part D

Relational database management system (RDBMS) MySQL which was previously employed by Craiglist is a free database management system that arranges data into tabular form with rows and columns for storing and presenting information. MySQL is compatible with many operating systems (Admin, 2019), optimal for transactional applications since it ensures relational integrity (Naveen, 2024), and has an in-built robust data security level with password encryption (Admin, 2019).

MongoDB is a non-relational (NoSQL) database system which stores data in collections as Binary JSON (BSON) documents, each of which is fundamentally composed of a key-value pair structure. MongoDB is suitable for capturing data whose structure is unknown because it can store schema-less data with ease (B, 2022).

Despite being straightforward and simple to use, MySQL can only be scaled vertically by exp anding the server's memory, storage space, or processing power. Costs associated with vertical scaling can be considerable, especially for big databases with high query volume. Additionally, MySQL needs a set schema that is specified up front and which the data must adhere to (Gopalakrishnan & Gopalakrishnan, 2023). This feature ensures that migrations are necessary for any changes to be made, which makes it less adaptable for evolving data models. (Naveen, 2024).

MongoDB, on the other hand, is intended to be scaled horizontally by dividing data among several servers. Naveen (2024). Also known as sharding, this approach involves introducing a new server instead of changing the configurations of the existing server which is usually less expensive (Gopalakrishnan & Gopalakrishnan, 2023). MongoDB also has a dynamic and high-performance schema which makes it easy to store and manipulate different types of data without compromising data integrity. A data collection can store different types of documents without any problem. It is easily saved as it does not require a predefined schema when a new type of document arrives (B, 2022).

Any modifications to Craigslist's live database schema would also need to be implemented on their archives due to the rigid constraints of the RDBMS MySQL schema. The legislative demand that Craigslist had to archive documents after 60 days required migrating data from the live database to the archived system and altering billions of rows of data in both the archives and the live database, which was difficult, expensive and time-consuming (MongoDB, n.d.).

In terms of scalability and flexibility, however, MongoDB provided Craiglist with an advantage because of its dynamic schema, which allows for schema changes on the live database without the need for expensive schema migrations.
Along with its inherent scalability, MongoDB boasted of the ability to store each post and its metadata as a single document (MongoDB, n.d.).

The operational challenges faced by Craigslist were mitigated by MongoDB's high availability and support for auto-sharding. In conclusion, Craigslist moved from MySQL to MongoDB mostly due to its need for a more flexible schema, a high-performance and scalable system capable of managing lots of data (MongoDB, n.d.).

REFERENCES

Admin. (2019, July 15). *MySQL advantages and disadvantages*. W3schools. https://www.w3schools.blog/mysql-advantages-disadvantages

(Admin, 2019)

Pedamkar, P. (2023, March 1). *MongoDB vs SQL server*. EDUCBA. https://www.educba.com/mongodb-vs-sql-server/

(Pedamkar, 2023)

Naveen. (2024, March 18). *MapReduce Tutorial - Learn MapReduce Basics in 5 days*. Intellipaat. https://intellipaat.com/blog/mongodb-vs-sql/

(Naveen, 2024)

Gopalakrishnan, J., & Gopalakrishnan, J. (2023, May 17). *KNOWI - MongoDB vs SQL - An in-depth comparison*. Knowi. https://www.knowi.com/blog/mongodb-vs-sql/

(Gopalakrishnan & Gopalakrishnan, 2023)

B, N. (2022, February 10). *MongoDB vs SQL Server: Which is Better? [10 Critical Differences]*. Learn | Hevo. https://hevodata.com/learn/mongodb-vs-sql-server/

(B, 2022)

MongoDB. (n.d.). *MonGoDB Case Study: Craigslist*. https://www.mongodb.com/post/15781260117/mongodb-case-study-craigslist

(MongoDB, n.d.)