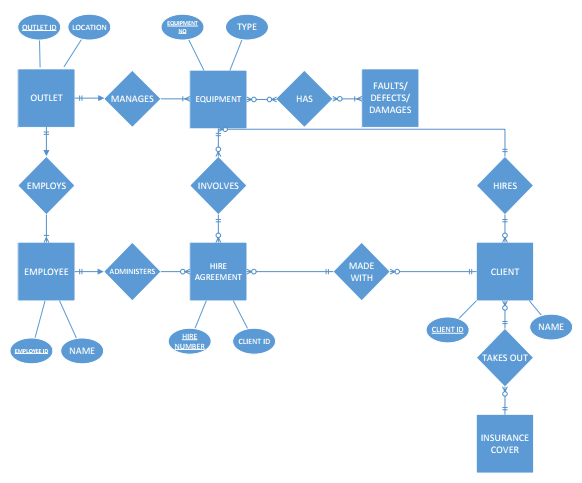
DATABASE MANAGEMENT

# PART A: CONCEPTUAL MODELLING [DATABASE DESIGN USING ER MODELLING]

1. ER Diagram



(Lucidchart, 2024)

1. Relational Table

Equipment (Equipment number, Type, Brand(for power tools), Model, Voltage(for power tools), Size (in tonnes for plants), Scaffold Type, Scaffold Width, Tolerance(for work wear), Size(for work wear))

Outlet (Outlet ID, Location, Manager ID\*, Supervision Record)

Employee (Employee ID, Name, Date of Birth, Address, Type(shop floor or office worker), Skill(for shop floor workers), Role(for office workers), Department)

Hire Agreement (Hire number, Equipment number\*, Client ID\*, Insurance cover, hire period, Return Date, Faults/Defects/Damages)

Client (Client ID, Name, Hire History)

# PART B: DATABASE CREATION AND QUERYING [SQL PROGRAMMING]

QUESTION ONE

**CREATE** **TABLE** Department **(**

deptId NUMBER **PRIMARY** **KEY,**

name VARCHAR2**(**100**)**

**);**

**CREATE** **TABLE** SalaryGrade **(**

salaryCode VARCHAR2**(**10**)** **PRIMARY** **KEY,**

startSalary NUMBER**,**

finishSalary NUMBER

**);**

**CREATE** **TABLE** PensionScheme **(**

schemeId NUMBER **PRIMARY** **KEY,**

name VARCHAR2**(**100**),**

rate NUMBER

**);**

**CREATE** **TABLE** Employee **(**

empId NUMBER **PRIMARY** **KEY,**

name VARCHAR2**(**100**),**

address VARCHAR2**(**200**),**

DOB DATE**,**

job VARCHAR2**(**100**),**

salaryCode VARCHAR2**(**10**),**

deptId NUMBER**,**

manager NUMBER**,**

schemeId NUMBER**,**

**FOREIGN** **KEY** **(**deptId**)** **REFERENCES** Department**(**deptId**),**

**FOREIGN** **KEY** **(**salaryCode**)** **REFERENCES** SalaryGrade**(**salaryCode**),**

**FOREIGN** **KEY** **(**schemeId**)** **REFERENCES** PensionScheme**(**schemeId**)**

**);**

QUESTION TWO

**PART A**

**SELECT** e**.**name**,** s**.**starting\_salary**,** e**.**deptId

**FROM** Employee e

**JOIN** SalaryGrade s **ON** e**.**salaryCode **=** s**.**salaryCode

**ORDER** **BY** e**.**deptId **DESC,** e**.**name **ASC;**

**PART B**

**SELECT** ps**.**schemeName**,** **COUNT(**e**.**schemeId**)** **AS** numEmployees

**FROM** PensionScheme ps

**LEFT** **JOIN** Employee e **ON** ps**.**schemeId **=** e**.**schemeId

**GROUP** **BY** ps**.**schemeName**;**

**PART C**

**SELECT** **COUNT(\*)** **AS** numEmployees

**FROM** Employee

**WHERE** job **!=** 'Manager' **AND**

salaryCode **IN** **(SELECT** salaryCode **FROM** SalaryGrade **WHERE** starting\_salary **>** 35000**);**

**PART D**

**SELECT** e**.**empId**,** e**.**name **AS** employee\_name**,** m**.**name **AS** manager\_name

**FROM** Employee e

**LEFT** **JOIN** Employee m **ON** e**.**manager **=** m**.**empId**;**

# PART C: DISTRIBUTED PROCESSING FRAMEWORKS

## QUESTION ONE

**CREATE** **TABLE** FlightData **(**

Year INT**,**

**Month** INT**,**

DayOfMonth INT**,**

DayOfWeek INT**,**

DepartureTime INT**,**

ActualDepartureTime INT**,**

ArrivalTime INT**,**

CarrierCode INT**,**

FlightNumber VARCHAR**(**10**),**

DepartureDelayMinutes INT**,**

ArrivalDelayMinutes INT**,**

WeatherDelayMinutes INT

**);**

SQL statements to determined delayed flights:

**SELECT** CarrierCode**,** **COUNT(\*)** **AS** DelayedFlights

**FROM** FlightData

**WHERE** DepartureDelayMinutes **>** 0 **OR** ArrivalDelayMinutes **>** 0 **OR** WeatherDelayMinutes **>** 0

**GROUP** **BY** CarrierCode**;**

## QUESTION TWO

1. **Map Phase:**
   * Each record (flight) in the file is read by a mapper.
   * Mapper extracts the CarrierCode and checks if the flight was delayed.
   * For each delayed flight, mapper emits a key-value pair where the key is the CarrierCode and the value is 1.
2. **Shuffle and Sort Phase:**
   * Output of mappers is shuffled and sorted based on the keys (CarrierCode).
3. **Reduce Phase:**
   * Each reducer receives a list of key-value pairs grouped by CarrierCode.
   * Reducer sums up the values (1s) for each CarrierCode, which gives the count of delayed flights for that carrier.
4. **Output Phase:**
   * Final output consists of CarrierCode and the count of delayed flights for each carrier.

**Justifications:**

* MapReduce is chosen for its ability to process large volumes of data in parallel across multiple nodes in a distributed manner.
* Mapper extracts only necessary information (CarrierCode and delay status) to minimize data transfer during the shuffle phase.
* Reducers aggregate the counts of delayed flights for each CarrierCode efficiently.
* The use of MapReduce allows scalability, fault-tolerance, and efficient processing of large datasets that cannot be handled by traditional centralized systems.

# PART D: RESEARCH REPORT

Craigslist's decision to migrate from relational MySQL servers to NoSQL MongoDB servers in 2011 was influenced by several factors stemming from the unique challenges they faced in managing their vast amount of data and the limitations of relational databases in addressing those challenges effectively (McKenzie, 2023).

Relational databases, like MySQL, are well-suited for structured data and transactions with predefined schemas. However, Craigslist encountered difficulties due to the dynamic and unstructured nature of their data. With over 1.5 million new ads posting daily and a billion records generated, their data had grown immensely, making it increasingly challenging to manage within the constraints of a relational database model.

One of the primary challenges Craigslist faced with MySQL was schema flexibility. As the structure of their data changed over the years, accommodating these changes in a relational database often proved to be a costly and time-consuming endeavour. Any alteration to the database schema required downtime, impacting the site's availability and user experience. Additionally, introducing new features became cumbersome due to the rigid schema requirements, further hindering Craigslist's ability to innovate and adapt to evolving user needs (MongoDB, 2020).

Moreover, maintaining compliance with legislative requirements posed another challenge for Craigslist. Legislation mandated a 60-day retention period for records in the live portion of the site, after which data had to be migrated to an archival space. Managing this archival process within MySQL was labour-intensive and time-consuming, as each change to the live database schema necessitated a corresponding update to the entire archive. This process, which took months to complete, not only increased operational overhead but also introduced potential risks and inconsistencies in data management.

In contrast, NoSQL databases like MongoDB offer a more flexible and scalable approach to handling large volumes of unstructured data. MongoDB's document-oriented data model allows for schema-less storage, enabling Craigslist to adapt to changes in data structure without the need for costly schema migrations. This flexibility significantly reduced the complexity and downtime associated with database alterations, empowering Craigslist to iterate and innovate more rapidly.

Furthermore, MongoDB's distributed architecture and horizontal scalability made it well-suited for handling Craigslist's massive dataset. By distributing data across multiple nodes and dynamically scaling resources as needed, MongoDB provided the scalability and performance required to support Craigslist's growing user base and data volume.

Additionally, MongoDB's support for sharding—a technique for horizontal partitioning of data—offered improved fault tolerance and resilience against hardware failures. By distributing data across multiple shards, MongoDB ensured high availability and data redundancy, mitigating the risk of data loss or downtime due to hardware failures.

Overall, the decision to migrate from MySQL to MongoDB was driven by Craigslist's need for a more flexible, scalable, and efficient solution to manage their vast amount of unstructured data. MongoDB's schema flexibility, scalability, and resilience addressed the limitations of MySQL, enabling Craigslist to overcome the challenges associated with data management, compliance, and innovation, ultimately enhancing the reliability and performance of their platform.

# References

Lucidchart. (2024, March 25). *Lucidchart.* From Lucidchart: https://www.lucidchart.com/pages/er-diagrams#:~:text=Make%20an%20ERD-,What%20is%20an%20ER%20diagram%3F,each%20other%20within%20a%20system.

McKenzie, C. (2023, Jan 9). *The CraigList Dilemma: A case study for big data and nosql solutions*. From TheServerSide: https://www.theserverside.com/feature/The-Craigslist-Dilemma-A-case-study-for-big-data-and-NoSQL-solutions#:~:text=Unfortunately%2C%20instead%20of%20making%20the,scale%20up%20easily%20over%20time.

MongoDB. (2020, March 2). *MongoDB Case Study: CraigsList*. From MongoDB: https://www.mongodb.com/post/15781260117/mongodb-case-study-craigslist