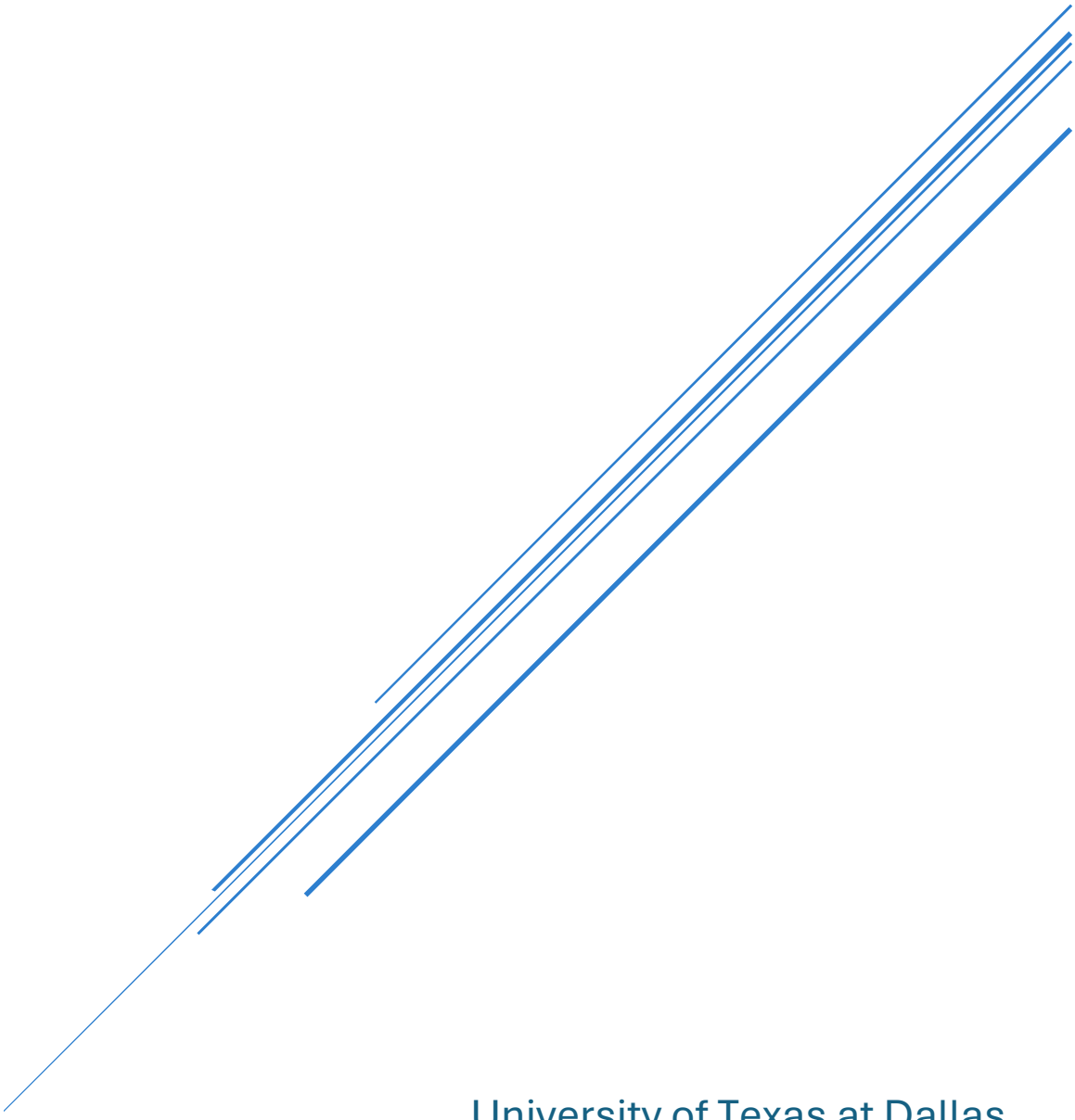
XYZ COMPANY PROJECT

XYZ Company is a business that procures certain parts from vendors to manufacture its products. The company operates through multiple departments, sites, and partners with a network of parts supply vendors. ***The project builds a database*** *for the XYZ company.*



University of Texas at Dallas
Database Systems Spring 2025

XYZ Company Project

Benny Frisella

Assumptions:

Shift–Department Relationship

Each shift belongs to one department, but a Department can have many shifts. This represents the one-to-many relationship between Department and Shift.

Employee–Shift Assignment

Each shift can be assigned to multiple people, or multiple people can work the same shift. Therefore, a junction table is used to connect Employee and Shift tables.

Interviewee Identity Ambiguity

As *interviewee* in the *interviews* table could reference either *potential employee* or *employee*, the interviewee attribute links back to the people table. We can **create a view specifically for employee applications using a join.**

Vendor Addresses

A vendor may have a couple addresses-even in the same city- but **every vendor must have at least one address and up to two are stored.**

Tracking Quantity by Part Type

As **we want to know the sums for each part type in a particular product**, and since the **totals aren't derived** - rather inserted via business logic - they are stored in the intermediate table between product and part.

Sales Granularity

Sales can record multiple products sold to the same customer by the same employee on the same day. A composite key of (Customer_id, Employee_id, Product_id, Date, Quantity) will be used in sales to allow different products to be purchased together at differing quantities.

Inventory Not Tracked

We are **currently focused on recording sales and managing operational data**, not inventory of parts or stock.

XYZ Company Project

Benny Frisella

Titles Change Infrequently

Titles in the Employee table are set to enumerate from a distinct list. This is to enforce integrity yet **assumes that company titles do not change often**. **The current title list is: ('Engineer', 'Technician', 'DBA', 'Manager', 'Sales Associate', 'Dept. Manager').**

Rank Enumeration

Assuming **Rank will not change much**, to enforce integrity, **Ranks can be chosen from a list. The current list is: ('Junior', 'Mid', 'Senior', 'Lead').**

Job Applications

Prospects are allowed to apply for more than one job simultaneously. Current data reflects only 2 maximum job applications per applicant.

Multiple People May Be Hired

Multiple people may be hired for the same job. To aid in differentiation, a **view is created** that includes the **date of the most recent interview** per candidate. This allows tracking of who completed the hiring process first and supports comparisons among selected applicants.

XYZ Company Project

Benny Frisella

Business Rules for Database Integrity

Supervisor is Separate

A supervisor column is added from an employee to **enforce integrity as a recursive relationship** in the employee table. Therefore, it is **left out of Title to ensure third normal form**.

Each Shift Has One Manager

Each shift is assigned a supervisor to help model variation between supervisors and shifts. This is enforced via a supervisor foreign key in the shift table.

Only One Supervisor per Shift

Supervisor identity is kept separate from employee roles in shifts to reduce ambiguity and enforce authority structure. **These two nuanced rules combined enforce third normal form in the shift table.**

Job_ID Auto-Incremented

Each time a new job is created, the **job_id is automatically incremented to create a globally unique job_id**. This is linked to a department via a foreign key in the Job table.

Zip-Code of Five Digits

Zip code will be kept at 5 digits as the rest can be imputed by the postal service.

Contact Record Limitations

Each **person may have up to two addresses and phone numbers** (primary and secondary). These limits help manage storage and are enforced using constraints.

Sales Quantity Tracking

Each sale records the quantity of products sold. This **supports accurate reporting and allows financial analysis at the product level.**

XYZ Company Project

Benny Frisella

Enhanced Security

By **refraining from referencing social security numbers** and instead creating an auto-incremented People_Id as a reference, we **protect user information** from a data breach.

Previous Department

As employees may only work in one department at a time, then we will **only track their most recent department worked in**.

Managers

There can be **many managers but only one department manager in each department**.

Supervisors

All employees have a supervisor **except for department managers.**

Employee Transactions

Employee salary payments are recorded monthly, including the pay date and amount. Since transaction numbers are only unique per employee and not globally, they are implemented as a **derived attribute** using a view. This approach avoids redundancy and improves efficiency, as **transaction numbers are not referenced elsewhere in the database**.

Interviews

There may only be **one interviewee per round of interviews**. (i.e. two people may not interview an interviewee in the same round.)

XYZ Company Project

Benny Frisella

Specialization and Generalization

The ability to specialize from a Person into Employee, Potential_Employee, and Customer is especially important in this context because it highlights both the common characteristics shared by all individuals in the system and the unique attributes that define their roles within the company.

These subclasses share core attributes with Person — such as name, address, phone number, and gender — but their relationships to the XYZ company differ significantly. Employees interact with internal operations, Potential_Employees interact through the hiring process, and Customers engage through transactions.

Rather than redundantly storing the same personal data across multiple tables, which would compromise both efficiency and data integrity, specialization provides a clean, normalized way to manage shared attributes in one place. This reduces data duplication and ensures consistency across all roles.

While database systems don't support inheritance in the same way object-oriented programming does (e.g., shared methods and behaviors), implementing superclass–subclass structures in data models is still highly beneficial. It improves organization, supports future flexibility, and reflects real-world relationships more accurately.

Here, the **subclasses** are **Covering** and **Overlapping**. This means that each Person **must be in at least one subclass, yet they are allowed to be in multiple subclasses**.

XYZ Company Project

Benny Frisella

Justifying the RDBMS

Relational Database Management Systems (RDBMS) are designed around the concept of structured relationships between data. Their reliance on primary and foreign keys, along with constraints, helps ensure data integrity, reduce redundancy, and enforce business rules across the system.

RDBMS platforms also support fast and reliable transactions, making them ideal for applications like this where data is frequently accessed, updated, and cross-referenced between related entities. Features like indexing, joins, and referential integrity enforcement allow for efficient querying and reporting across complex relationships.

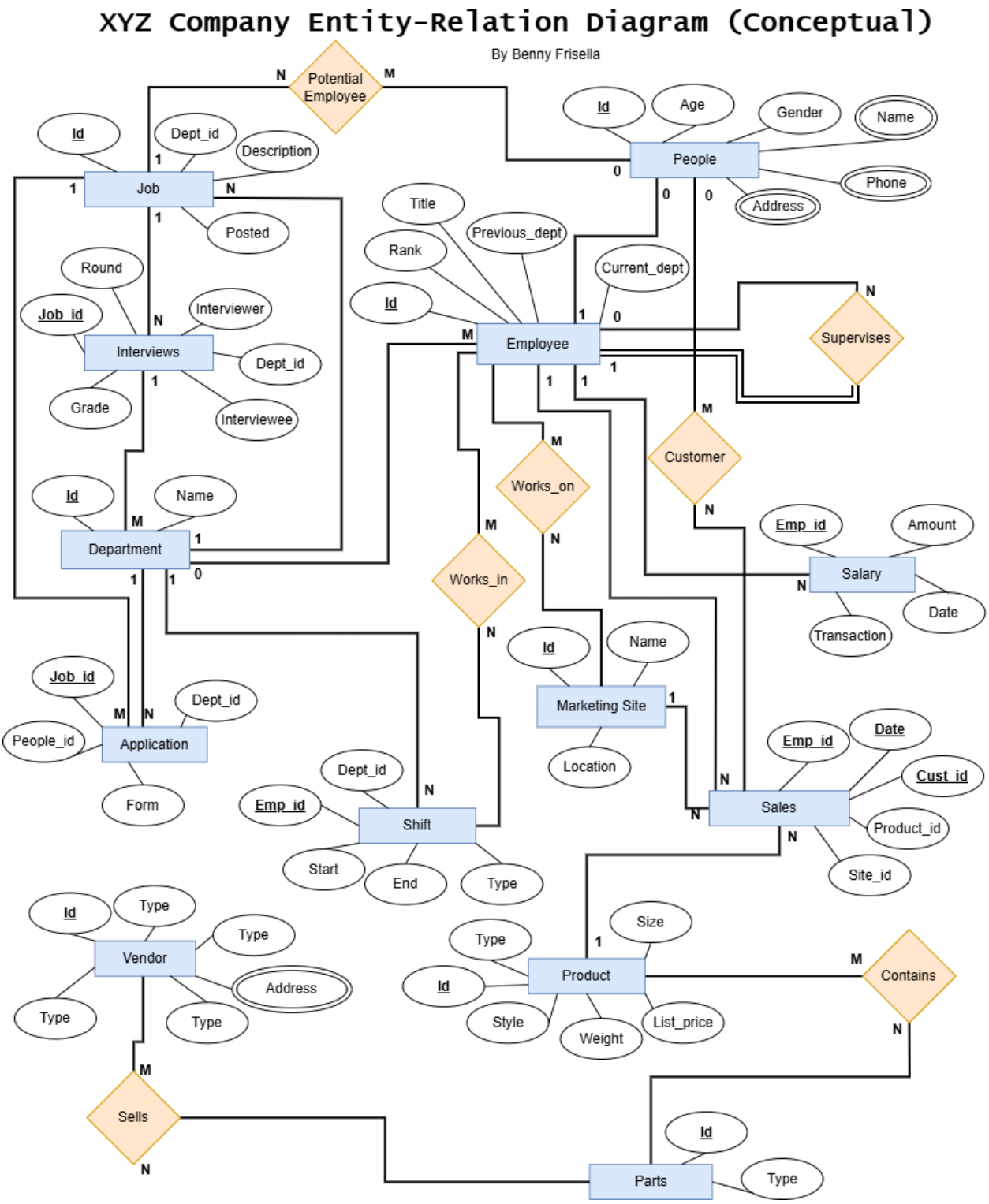
Additionally, relational systems are known for being scalable and secure, which are critical considerations for modern organizations that handle sensitive data and require consistent performance as their systems grow.

For these reasons, using a relational DBMS such as Oracle/MySQL is both a practical and reliable choice for managing the structured, highly relational data in this company's environment.

XYZ Company Project

Benny Frisella

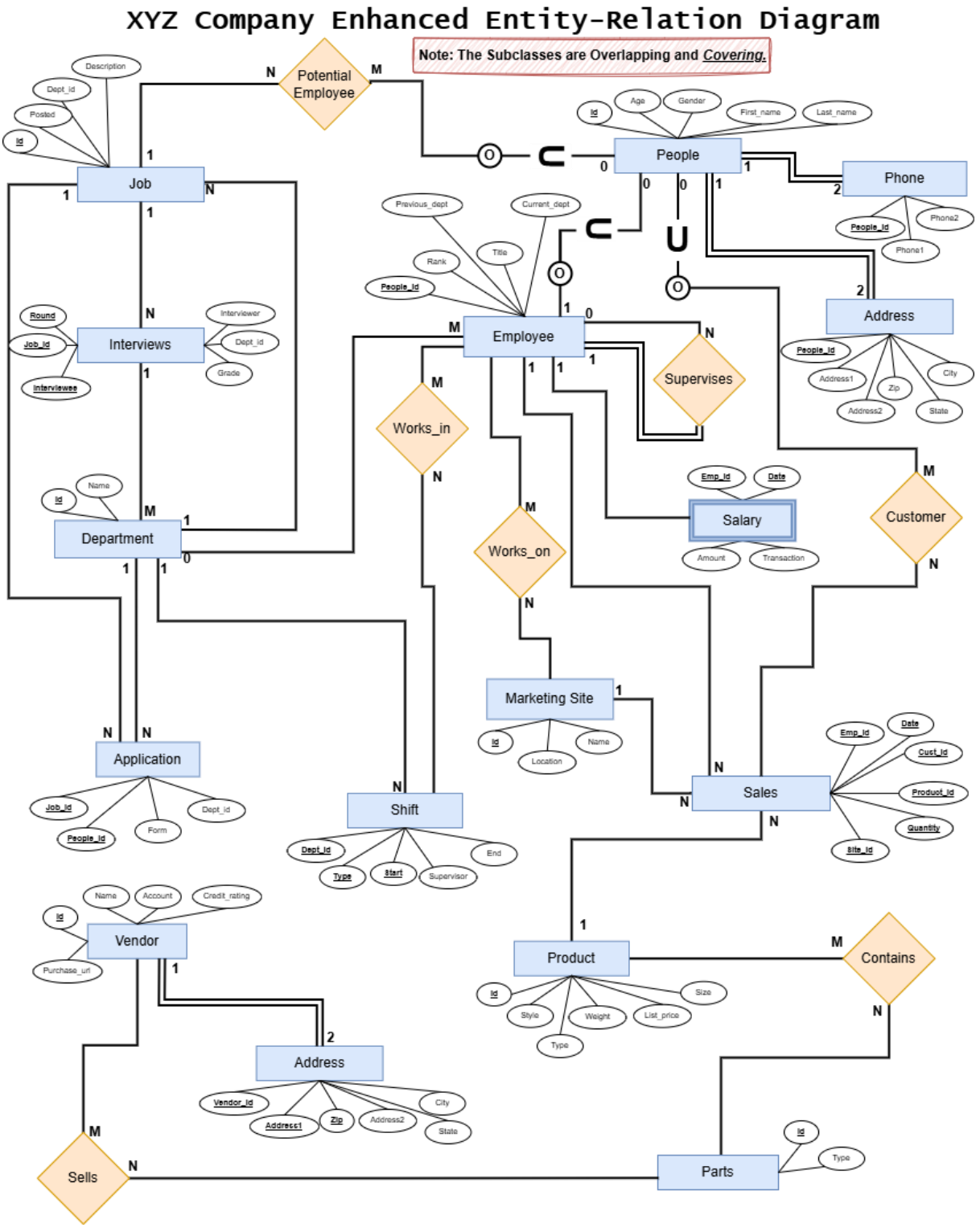
Conceptual Design



XYZ Company Project

Benny Frisella

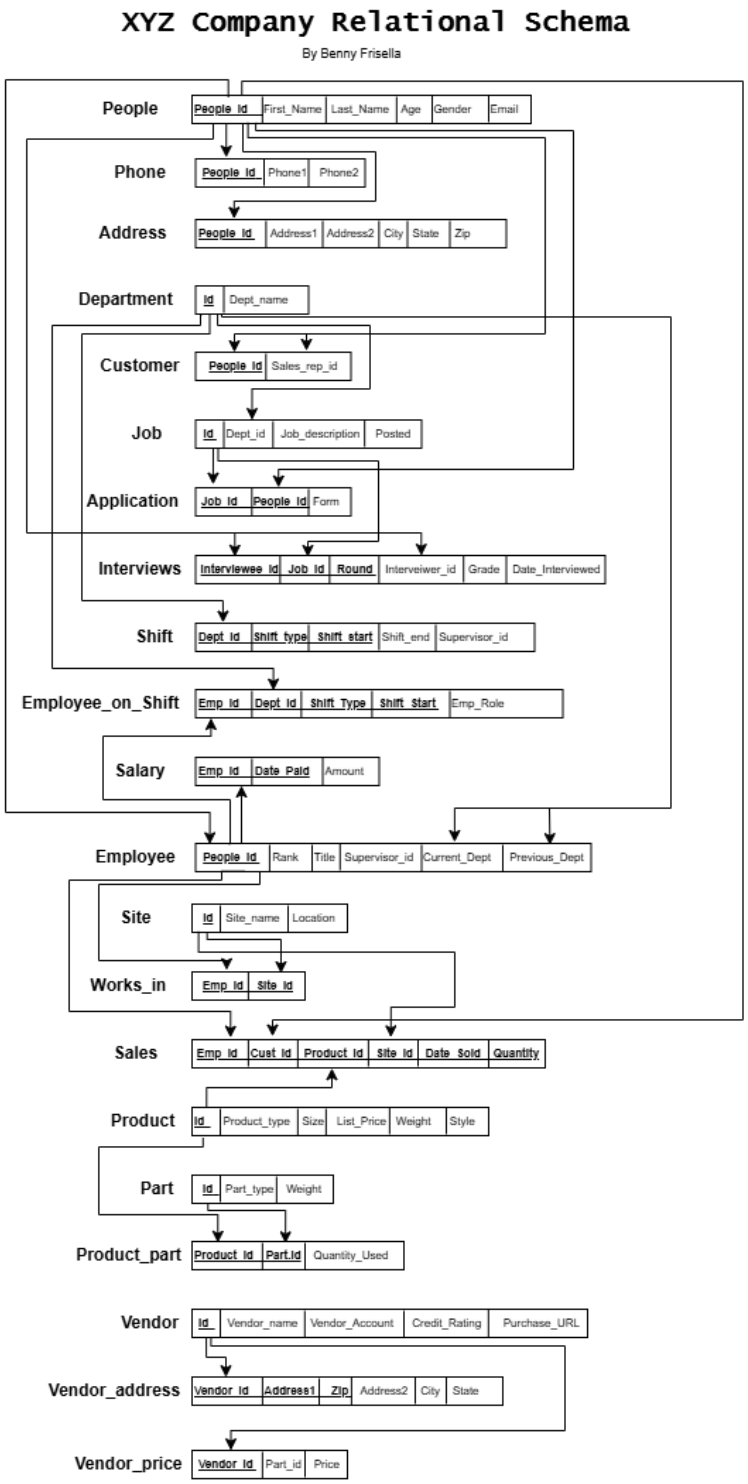
Logical Design



XYZ Company Project

Benny Frisella

Relational Schema



XYZ Company Project

Benny Frisella

Dependency Diagram

Since each table is in 3rd Normal Form (3NF), all non-key attributes are functionally dependent only on the primary key, and not on any other non-key attribute

People_id ⇒ First_Name, Last_Name, Age, Gender, Email

Employee.People_id ⇒ Emp_Rank, Title, Supervisor_id, Current_Dept, Previous_Dept

Customer.People_id ⇒ Sales_Rep_id

Department.Id ⇒ Dept_name

Job.Id ⇒ Dept_id, Job_Description, Posted

Application.(Job_id, People_id) ⇒ Form

Interviews.(Interviewee_id, Job_id, Round) ⇒ Interviewer_id, Grade, Date_Interviewed

Product.Id ⇒ Product_type, Size, List_Price, Weight, Style

Part.Id ⇒ Part_type, Weight

Product_Part.(Product_id, Part_id) ⇒ Quantity_Used

Vendor.Id ⇒ Vendor_name, Vendor_Account, Credit_Rating, Purchase_URL

Vendor_Price.(Vendor_id, Part_id) ⇒ Price

Site.Id ⇒ Site_name, Location

Works_In.(Emp_id, Site_id) ⇒ [zero non-key attributes]

Sales.(Emp_id, Cust_id, Product_id, Site_id, Date_Sold) ⇒ Quantity

Shift.(Dept_id, Shift_type, Shift_start) ⇒ Shift_end, Supervisor_id

Employee_on_Shift.(Emp_id, Dept_id, Shift_Type, Shift_Start) ⇒ Emp_Role

Salary.(Emp_id, Date_Paid) ⇒ Amount

Phone.People_id ⇒ Phone1, Phone2

Address.People_id ⇒ Address1, Address2, City, State, Zip

XYZ Company Project

Benny Frisella

Queries and Results

```
# referred to in Part e) of XYZ Company Project.docx

########## NOTE: My data generation is unique, thus I've changed the nouns but kept the syntax of your original queries to work with my data. ##########
#                          -----> indicates the syntax was changed to work with my uniquely fake data

use xyz_company;

######### 1. Interviewers for "Hellen Cole" on job 11111 -----> "Paul Wilson" on job 30013
SELECT DISTINCT i.Interviewer_ID, person.First_Name, person.Last_Name, i.Round as Interview_Round
FROM Interviews i
JOIN People person ON i.Interviewer_ID = person.ID
JOIN People target ON i.Interviewee_ID = target.ID
WHERE target.First_Name = 'Paul' AND target.Last_Name = 'Wilson'
  AND i.Job_ID = 30013;
```

Interviewer_ID	First_Name	Last_Name	Interview_Round
452	Lisa	Blackwell	1
227	John	Day	2
415	Keith	Johnson	3
394	Emily	Cunningham	4
476	Troy	Nguyen	5

```
######### 2. Job IDs posted by department "Marketing" in Jan 2011 -----> July 2024
SELECT j.ID as Job_id, Dept_name, YEAR(j.Posted) as Posted
FROM Job j
JOIN Department d ON j.Dept_ID = d.ID
WHERE d.dept_name = 'Marketing'
  AND MONTH(j.Posted) = 7 AND YEAR(j.Posted) = 2024;
```

Job_id	Dept_name	Posted
30014	Marketing	2024
30016	Marketing	2024
30091	Marketing	2024

XYZ Company Project

Benny Frisella

```
######### 3. Employees with no supervisees
SELECT e.People_id as Emp_id, person.First_Name, person.Last_Name
FROM Employee e
JOIN People person ON e.People_id = person.ID
WHERE e.People_id NOT IN (
    SELECT DISTINCT Supervisor_id FROM Employee WHERE Supervisor_id IS NOT NULL
);
```

Emp_id	First_Name	Last_Name
4	Pamela	Rodgers
5	Brian	Torres
11	Kyle	Lane
13	Richard	Harrington
20	John	Morales
25	Nathan	Davies
28	John	Long
29	Tiffany	Hardin
31	Tiffany	King
33	Shane	Mcgee
35	Jeanne	Stewart
37	Joshua	Davis
39	Ricardo	Young

Result 34

XYZ Company Project

Benny Frisella

```
######### 4. Sites with no sales in March 2011 -----> May 2025
SELECT s.ID, s.Location
FROM Site s
WHERE s.ID NOT IN (
    SELECT DISTINCT Site_ID
    FROM Sales
    WHERE MONTH(Date_Sold) = 5 AND YEAR(Date_Sold) = 2025
);
```

ID	Location
13	New Mary
36	Lake Denise
41	South Sara
66	Justinville
72	East Josephtown
88	Patriciahaven
99	North Michael

```
######### 5. Jobs that did not hire anyone one month after posting
SELECT j.ID as Job_id, j.Dept_id as Department, j.Posted
FROM Job j
WHERE NOT EXISTS (
    SELECT 1
    FROM (
        SELECT Interviewee_id, Job_id
        FROM Interviews
        WHERE Grade > 60
        GROUP BY Interviewee_id, Job_id
        HAVING COUNT(*) >= 5
    ) passed
    WHERE passed.Job_id = j.ID
);
```

Job_id	Department	Posted
30073	3	2024-08-11
30094	4	2025-02-14

XYZ Company Project

Benny Frisella

```
######### 6. Sales rep who sold all product types over $200 --------> Count of Items Over $200 Sold Per Salesman
SELECT s.Emp_id as Sales_Rep, COUNT(*) AS Sales_Over_200
FROM Sales s
JOIN Product p ON s.Product_id = p.Id
WHERE p.List_Price > 200
GROUP BY s.Emp_id
ORDER BY Sales_Over_200 DESC;
```

Sales_Rep	Sales_Over_200
401	8
306	7
15	7
136	7
496	7
334	7
328	7
214	6
419	6
131	6
289	6
166	6
24	6

Result 22

XYZ Company Project

Benny Frisella

```
######### 7. Departments with no job postings between Jan 1–Feb 1, 2011 ------> May 1–Jun 1, 2024
SELECT d.Id, d.Dept_name
FROM Department d
WHERE NOT EXISTS (
    SELECT 1
    FROM Job j
    WHERE j.Dept_Id = d.Id
      AND j.Posted BETWEEN '2024-05-01' AND '2024-06-01'
);
```

Id	Dept_name
3	HR
NULL	NULL

```
######### 8. Employees who applied to job 12345 -----> job 30002
SELECT e.People_id as Emp_id, e.Current_Dept, p.First_Name, p.Last_Name
FROM Application a
JOIN Employee e ON a.People_id = e.People_id
JOIN People p ON e.People_id = p.Id
WHERE a.Job_id = 30002;
```

Emp_id	Current_Dept	First_Name	Last_Name
429	4	William	Wyatt
435	1	Alexander	Farley
451	5	Kristy	Lamb
482	1	Jonathan	Ray
495	5	John	Shelton

XYZ Company Project

Benny Frisella

```
######### 9. Best-selling product type (by quantity)
SELECT p.Product_Type, SUM(s.Quantity) AS Total_Quantity_Sold
FROM Sales s
JOIN Product p ON s.Product_id = p.Id
GROUP BY p.Product_Type
ORDER BY Total_Quantity_Sold DESC
LIMIT 1;
```

	Product_Type	Total_Quantity_Sold
▶	Chair	577

```
######### 10. Most profitable product type (List_Price - SUM(Parts Cost))
SELECT p.Product_Type,
       ROUND(SUM(p.List_Price - COALESCE(cost.Total_Cost, 0)), 2) AS Net_Profit
FROM Product p
LEFT JOIN (
    SELECT pp.Product_id,
           SUM(pp.Quantity_Used * vp.MinPrice) AS Total_Cost
    FROM Product_Part pp
    JOIN (
        SELECT Part_id, MIN(Price) AS MinPrice
        FROM Vendor_Price
        GROUP BY Part_id
    ) vp ON pp.Part_id = vp.Part_id
    GROUP BY pp.Product_id
) cost ON p.Id = cost.Product_id
GROUP BY p.Product_Type
ORDER BY Net_Profit DESC
LIMIT 1;
```

	Product_Type	Net_Profit
▶	Monitor	2432.80

XYZ Company Project

Benny Frisella

```
######### 11. Employees who worked in all departments -----> Employees who have worked in multiple departments
SELECT e.People_id as Emp_id, p.First_Name, p.Last_Name, e.Current_Dept, e.Previous_Dept
FROM Employee e
JOIN People p ON e.People_id = p.Id
WHERE e.Previous_Dept IS NOT NULL;
```

Emp_id	First_Name	Last_Name	Current_Dept	Previous_Dept
4	Pamela	Rodgers	1	2
7	Emily	Houston	5	4
8	Anthony	Hunt	3	2
9	Robin	Jensen	4	3
10	Pamela	Hernandez	4	3
12	Alexandra	Jackson	3	2
14	Wesley	Lee	5	4
17	Vincent	Rodriguez	5	4
18	Benjamin	Wolfe	3	2
20	John	Morales	1	2
24	Thomas	Mills	3	2
25	Nathan	Davies	1	2
26	Kevin	Obrien	3	2

XYZ Company Project

Benny Frisella

```
######### 12. Selected interviewees Emails (5+ Grades ≥ 60)
SELECT DISTINCT p.First_Name, p.Last_Name, p.Email
FROM Interviews i
JOIN People p ON i.Interviewee_id = p.Id
WHERE i.Interviewee_id IN (
    SELECT Interviewee_id
    FROM Interviews
    WHERE Grade > 60
    GROUP BY Interviewee_id, Job_id
    HAVING COUNT(*) >= 5
);
```

First_Name	Last_Name	Email
Paul	Wilson	ericjames@example.org
Jamie	Wallace	laurabush@example.org
Robert	Reid	robert72@example.com
Vincent	Phillips	diana52@example.com
Sheri	Kelly	bushjamie@example.net
Jonathan	Johnson	qmartinez@example.org
David	Choi	kellysmith@example.net
Elizabeth	Cameron	kimberly43@example.com
David	Valdez	stevenromero@example.net
Cassidy	Harper	cbowman@example.net
Natasha	Brown	karen99@example.com
Keith	Johnson	kellyanthony@example.net
Rhonda	Lewis	jermainegoodman@exampl...

XYZ Company Project

Benny Frisella

```
######## 13. Contact info of selected interviewees for all their jobs ------> selected first by first date finished after 5+ interview grades of 60+ using a created view
SELECT s.Job_id, p.First_Name, p.Last_Name, p.Email, ph.Phone1, s.Selection_Date, s.Avg_Grade, s.Total_Interviews
FROM Select_Candidate_Date s
JOIN People p ON s.Interviewee_id = p.Id
LEFT JOIN Phone ph ON p.Id = ph.People_id
ORDER BY s.Job_id;
```

Job_id	First_Name	Last_Name	Email	Phone1	Selection_Date	Avg_Grade	Total_Interviews
30001	Robert	Howell	simonbarry@example.net	8278311354	2024-07-21	77.17	6
30002	Alexander	Farley	russellmiller@example.com	646.728.8107	2025-04-17	80.00	5
30002	Sally	Lynch	april88@example.com	+1-229-980-9740x1391	2025-04-17	79.80	5
30002	Alexander	Myers	jmoore@example.org	+1-756-998-8818	2025-04-22	73.67	6
30003	Margaret	English	steveallen@example.net	001-794-972-5058x871	2024-08-07	87.80	5
30003	Laura	Martinez	eric61@example.org	+1-548-679-3221x5955	2024-08-09	76.20	5
30004	Tammy	Bonilla	johnsonhannah@example.org	600.739.5265x31905	2025-01-23	70.50	6
30004	Michael	Leonard	adamsaaron@example.com	(766)227-4304x329	2025-01-26	72.17	6
30004	Matthew	Avery	sallymoore@example.org	625-253-9267	2025-01-20	79.00	5
30004	Sheila	Newman	andrewzuniga@example.net	001-205-707-4273x889	2025-01-26	75.33	6
30005	John	Tyler	brian92@example.org	878.255.2923x38140	2024-07-05	78.83	6
30005	Jessica	Jenkins	dicksonjonathan@example.net	6534471485	2024-06-27	88.00	5
30005	Tammie	Levine	robinsonbryan@example.net	+1-993-477-9438x7714	2024-07-06	78.20	5

[View can be found in Views Section.]

```
########## 14. Employee with highest average monthly salary
SELECT s.Emp_id, p.First_Name, p.Last_Name, ROUND(AVG(s.Amount), 2) AS Avg_Salary
FROM Salary s
JOIN People p ON s.Emp_id = p.Id
GROUP BY s.Emp_id, p.First_Name, p.Last_Name
ORDER BY Avg_Salary DESC
LIMIT 1;
```

Emp_id	First_Name	Last_Name	Avg_Salary
48	Jesse	Smith	7336.67

XYZ Company Project

Benny Frisella

```
######### 15. Vendor offering “Cup” part under 4 lb at lowest price ------> ...offering "Motor 9"
SELECT v.Id, v.Vendor_name
FROM Vendor v
JOIN Vendor_Price vp ON v.Id = vp.Vendor_id
JOIN Part p ON vp.Part_id = p.Id
WHERE p.Part_type = 'Motor 9'
  AND p.Weight < 4
  AND vp.Price = (
      SELECT MIN(vp2.Price)
      FROM Vendor_Price vp2
      JOIN Part p2 ON vp2.Part_id = p2.Id
      WHERE p2.Part_type = 'Motor 9'
        AND p2.Weight < 4
  );

  select * from part;
```

Id	Part_type	Weight
1	Bolt 1	3.91
2	Module 2	4.14
3	Gear 3	7.71
4	Nozzle 4	3.65
5	Hinge 5	8.06
6	Cover 6	5.55
7	Bearing 7	2.55
8	Capacitor 8	4.67
9	Motor 9	2.29
10	Casing 10	6.96
11	Blade 11	1.07
12	Blade 12	2.55
13	Pin 13	2.92

XYZ Company Project

Benny Frisella

Created Views

```
# referred to in d) of XYZ company.docx

# Quick Views #

# 1. Average Monthly Salary per Employee
SELECT * FROM Average_Monthly_Salaries;

# 2. Rounds Passed Per Interviewee Per Job
SELECT * FROM Interviewee_Rounds_Passed ORDER BY Job_id;

# 3. Number of Items Sold per Product Type
SELECT * FROM Items_Sold_Per_Product_Type ORDER BY Total_Quantity_Sold DESC;

# 4. Part Purchase Cost Per Product
SELECT * FROM Minimum_Parts_Cost_Per_Product ORDER BY Total_Part_Cost DESC;

# View for Transaction Numbers
SELECT * FROM Salary_With_Transaction;
```

XYZ Company Project

Benny Frisella

```
################## View Creations ##################

# 1. Average Monthly Salary per Employee

CREATE VIEW Average_Monthly_Salaries AS
SELECT
    Emp_id,
    ROUND(AVG(Amount), 2) AS Avg_Monthly_Salary
FROM Salary
GROUP BY Emp_id;

SELECT * FROM Average_Monthly_Salaries;
############################################################
```

Emp_id	Avg_Monthly_Salary
1	6355.09
2	6325.06
3	6083.39
4	5438.62
5	6814.43
6	6275.45
7	5950.06
8	5794.67
9	6181.34
10	5768.43
11	6097.51
12	6066.02
13	6319.72

XYZ Company Project

Benny Frisella

```
# 2. Rounds Passed Per Interviewee Per Job

CREATE VIEW Interviewee_Rounds_Passed AS
SELECT
    Interviewee_id,
    Job_id,
    COUNT(*) AS Rounds_Passed
FROM Interviews
WHERE Grade > 60
GROUP BY Interviewee_id, Job_id
ORDER BY Job_id;

SELECT * FROM Interviewee_Rounds_Passed ORDER BY Job_id;
############################################################
```

Interviewee_id	Job_id	Rounds_Passed
550	30001	2
861	30001	4
867	30001	5
964	30001	6
991	30001	4
429	30002	4
435	30002	5
451	30002	3
482	30002	4
495	30002	5
571	30002	5
589	30002	6
861	30002	5

XYZ Company Project

Benny Frisella

```
############################################################

# 3. Number of Items Sold per Product Type
## sums quantity of parts for each product

CREATE VIEW Items_Sold_Per_Product_Type AS
SELECT
    p.Product_type,
    SUM(s.Quantity) AS Total_Quantity_Sold
FROM Sales s
JOIN Product p ON s.Product_id = p.Id
GROUP BY p.Product_type;

SELECT * FROM Items_Sold_Per_Product_Type ORDER BY Total_Quantity_Sold DESC;
############################################################
```

Product_type	Total_Quantity_Sold
Chair	577
Sneakers	567
Notebook	451
Headphones	394
Backpack	362
Monitor	358
Watch	333
Keyboard	324
Lamp	320
Table	263
Desk	253
Jacket	241
Water Bottle	197

XYZ Company Project

Benny Frisella

```
# 4. Part Purchase Cost Per Product
## sums quantity used x minimum(vendor price)

CREATE VIEW Minimum_Parts_Cost_Per_Product AS
SELECT
    p.Id AS Product_id,
    p.Product_Type,
    ROUND(SUM(pp.Quantity_Used * vp.MinPrice), 2) AS Total_Part_Cost
FROM Product p
JOIN Product_Part pp ON p.Id = pp.Product_id
JOIN (
    SELECT Part_id, MIN(Price) AS MinPrice
    FROM Vendor_Price
    GROUP BY Part_id
) vp ON pp.Part_id = vp.Part_id
GROUP BY p.Id, p.Product_Type;

SELECT * FROM Minimum_Parts_Cost_Per_Product ORDER BY Total_Part_Cost DESC;
############################################################
```

Product_id	Product_Type	Total_Part_Cost
81	Chair	568.39
57	Table	474.27
88	Jacket	440.35
91	Notebook	412.49
16	Keyboard	384.26
154	Watch	371.01
73	Headphones	366.76
71	Watch	353.70
13	Monitor	346.73
30	Backpack	343.87
191	Watch	342.83
87	Sofa	341.25
22	Watch	332.80

XYZ Company Project

Benny Frisella

```
#################### Views For Derived Attributes ####################

# View for Transaction Numbers
CREATE VIEW Salary_With_Transaction AS
SELECT
    ROW_NUMBER() OVER (PARTITION BY Emp_id ORDER BY Date_Paid) AS Transaction_no,
    Emp_id,
    Date_Paid,
    Amount
FROM Salary;

SELECT * FROM Salary_With_Transaction ORDER BY Transaction_no;
############################################################
```

Transaction_no	Emp_id	Date_Paid	Amount
1	1	2024-06-05 00:00:00	7460.03
1	168	2024-06-05 00:00:00	7348.18
1	126	2024-06-05 00:00:00	6099.77
1	251	2024-06-05 00:00:00	6311.18
1	442	2024-06-05 00:00:00	8618.50
1	390	2024-06-05 00:00:00	6786.39
1	22	2024-06-05 00:00:00	6519.58
1	188	2024-06-05 00:00:00	6497.80
1	157	2024-06-05 00:00:00	4903.29
1	459	2024-06-05 00:00:00	4376.77
1	304	2024-06-05 00:00:00	8412.96
1	260	2024-06-05 00:00:00	6850.34
1	2	2024-06-05 00:00:00	7564.71

XYZ Company Project

Benny Frisella

```
############################################################

# Current number of sales for each product
CREATE VIEW Product_Sales AS
SELECT p.Product_Type, COUNT(*) AS Times_Sold
FROM Sales s
JOIN Product p ON s.Product_id = p.Id
WHERE p.List_Price > 100
GROUP BY p.Product_Type;

SELECT * FROM Product_Sales ORDER BY Times_Sold DESC;
############################################################
```

Product_Type	Times_Sold
Sneakers	140
Notebook	126
Watch	116
Chair	111
Backpack	109
Headphones	103
Lamp	96
Monitor	94
Keyboard	93
Jacket	89
Desk	85
Table	79
Sofa	66

XYZ Company Project

Benny Frisella

```sql
# Selected Candidates by soonest date of interviews completion
CREATE VIEW Select_Candidate_Date AS
SELECT
    i.Interviewee_id,
    i.Job_id,
    MAX(i.Date_Interviewed) AS Selection_Date,
    ROUND(AVG(i.Grade), 2) AS Avg_Grade,
    COUNT(*) AS Total_Interviews
FROM Interviews i
GROUP BY i.Interviewee_id, i.Job_id
HAVING
    COUNT(*) >= 5 AND
    MIN(i.Grade) >= 60 AND
    AVG(i.Grade) > 70;

Select * from Select_Candidate_Date;
```

Interviewee_id	Job_id	Selection_Date	Avg_Grade	Total_Interviews
401	30078	2025-03-15	87.33	6
402	30064	2025-05-22	82.40	5
403	30016	2024-08-02	75.60	5
403	30028	2024-08-02	80.00	6
404	30036	2024-07-06	71.50	6
405	30007	2025-01-15	86.20	5
405	30013	2025-02-19	87.20	5
406	30068	2025-02-03	89.33	6
408	30018	2024-11-16	86.00	5
408	30037	2025-05-07	76.80	5
409	30054	2025-01-08	81.40	5
411	30076	2025-03-27	78.67	6
412	30086	2024-12-28	82.00	6

XYZ Company Project

Benny Frisella

Project Summary

This project involved the **end-to-end design, implementation, and population of a comprehensive relational database** for a simulated company. The **schema** was carefully structured to **support a wide range of business functions**, including employee management, job applications, interviews, product-part tracking, vendor pricing, customer sales, and shift scheduling. **Key challenges** included **maintaining normalization** while handling complex relationships such as recursive supervisor assignments, **multi-stage interview evaluation**, **composite primary keys across operational tables, and creating fairly complex queries and views**. Using Python and the Faker library, I **generated thousands of *realistic***, **constraint-respecting records**, and **overcame issues related to profitability logic, referential integrity,** and diagramming limitations in Word in the Relational Schema.

Through this experience, I deepened my understanding of database design, functional dependencies, normalization up to 3NF, and the practical application of SQL constraints, views, and data modeling principles at scale. This project reflects not only technical capability but also the ability to model real-world business rules with data integrity and clarity.