Tech ABC Corp - HR Database

[Junghoon Suk & 3 April 2021]

Business Scenario

Business requirement

Tech ABC Corp saw explosive growth with a sudden appearance onto the gaming scene with their new AI-powered video game console. As a result, they have gone from a small 10 person operation to 200 employees and 5 locations in under a year. HR is having trouble keeping up with the growth, since they are still maintaining employee information in a spreadsheet. While that worked for ten employees, it has becoming increasingly cumbersome to manage as the company expands.

As such, the HR department has tasked you, as the new data architect, to design and build a database capable of managing their employee information.

Dataset

The HR dataset you will be working with is an Excel workbook which consists of 206 records, with eleven columns. The data is in human readable format, and has not been normalized at all. The data lists the names of employees at Tech ABC Corp as well as information such as job title, department, manager's name, hire date, start date, end date, work location, and salary.

IT Department Best Practices

The IT Department has certain Best Practices policies for databases you should follow, as detailed in the Best Practices document.

Step 1

Data Architecture Foundations

Step 1: Data Architecture Foundations

Hi,

Welcome to Tech ABC Corp. We are excited to have some new talent onboard. As you may already know, Tech ABC Corp has recently experienced a lot of growth. Our AI powered video game console WOPR has been hugely successful and as a result, our company has grown from 10 employees to 200 in only 6 months (and we are projecting a 20% growth a year for the next 5 years). We have also grown from our Dallas, Texas office, to 4 other locations nationwide: New York City, NY, San Francisco, CA, Minneapolis, MN, and Nashville, TN.

While this growth is great, it is really starting to put a strain on our record keeping in HR. We currently maintain all employee information on a shared spreadsheet. When HR consisted of only myself, managing everyone on an Excel spreadsheet was simple, but now that it is a shared document I am having serious reservations about data integrity and data security. If the wrong person got their hands on the HR file, they would see the salaries of every employee in the company, all the way up to the president.

After speaking with Jacob Lauber, the manager of IT, he suggested I put in a request to have my HR Excel file converted into a database. He suggested I reach out to you as I am told you have experience in designing and building databases. When you are building this, please keep in mind that I want any employee with a domain login to be able to access the database. I just don't want them having access to salary information. That needs to be restricted to HR and management level employees only.

I also want to make sure you know that am looking to turn my spreadsheet into a live database, one I can input and edit information into. I am not really concerned with reporting capabilities at the moment. Since we are working with employee data we are required by federal regulations to maintain this data for at least 7 years; additionally, since this is considered business critical data, we need to make sure it gets backed up properly.

I am looking forward to working with you and seeing what kind of database you design for us.

Thanks,

Sarah Collins
Head of HR

Data Architect Business Requirement

- **Purpose of the new database**

  The two main purpose of the new database is to improve data integrity and security. Currently the entire record of employee information is maintained in a spreadsheet where the data is arbitrarily shared. This will cause a serious data integrity issue especially when the user is granted with write access and starts to enter or modify the data without stringent constant rules.

  Furthermore, HR information normally contains confidential information like salary and secure management of the data is important.

  The new database will help improve on these current challenges. Constraints will ensure the consistency in data input and the security by limiting the access to the database to certain authorized people.

- **Data to be stored**

  Before normalizing the data, the while data set will be staged with columns that include employee id, employee name, email, hired date, job title, salary, department, manager name, start date, end date, location, address, city, state and education level.

  Each row entry is uniquely identified by employee id. No missing data is expected except end date which cannot be entered if an employee has not left the company.

- **Who will own/manage data**

  The owner of the database is thought to be the head of HR while the overall database management is designed and maintained by the database team.

- **Who will have access to database**

  The overall database access is granted to any employee with a username. However, the access to salary data must be restricted and only HR and management level employee can query.

Data Architect Business Requirement

- **Estimated size of database**

  The size of the initial database will be 205 rows by 15 columns. 205 rows represent unique number of employees.

- **Estimated annual growth**

  It is projected to grow 20% a year for the next 5 years.

- **Is any of the data sensitive/restricted**

  The entire database should be protected within the company. Each employee with a domain login is able to access the database.

  However, the access must be restricted to authorized people only for salary data due to its sensitivity.

- **Data retention and backup requirements**

  The company is required by federal regulations to maintain employee data at least 7 years.

  The data is considered ‘critical’ business data so backup must be done properly. As per the IT Best Practices Guide, full backup will then need to be done once a week, together with incremental backup daily.

Data Architect Technical Requirement

- ## Justification for the new database

  The new database will significantly improve the current concern of data integrity and security by schema constraints and data governance.

  Moreover, given the expectation of the projected annual growth, the database can be designed to ensure scalable operations and flexibility in use across different departments.

- ## Database objects

  Outputs from this database are following:

Type	Name	Description
Tables (7)	Employee_record, Employee, Education, Location, Job_title, Salary and Department	
View (1)	original_dataset	to replicate the original dataset provided as `hr-dataset.csv` which promotes more readable representation of the data. Callable with a query as below: SELECT * FROM original_dataset;
Function (1)	employee_job_history	to search for a job history given a parameter employee name. Callable with a query with an example as below: SELECT employee_job_history ('Toni Lembeck')

  [NOTE] In step 2 of this document, logical / physical ERD are available which visually represents the understanding of the database structure and relationship.

- ## Data ingestion

  At the current stage, we will adopt ETL data ingestion method. As instructed by IT Best Practices Guide, it is the current best practice for working with flat files like the csv file that we are provided with ('hr-dataset.csv').

Data Architect Technical Requirement

- ## Data governance (Ownership and User access)

  Ownership: any employee with a domain username can access to the database.

  User access: sensitive data such as salary will be restricted. Only HR and the management level employees will be granted to access.

- ## Scalability & Flexibility considerations

  Scalability : consideration for the growth of data and users is needed. The current data is relatively small with 205 rows and the projected growth of 5 so scalability is not a big concern yet. As users grow we may consider deploying replicated databases which allows the main database to be offloaded to several copied databases and improves read time.

  Flexibility : The expected current use of the database is for reading and modifying the employee records. The need for reporting capability is expected in the future where we need considering the compatibility of the database with reporting tools.

- ## Storage & retention

  Storage (disk or in-memory): by each server group, a standard partition of 1GB will be given to the database. When expecting larger than 10k rows of data must be asked.

  Retention: the federal regulations require at least 7 years of retention of employment data.

- ## Backup

  The HR employment information will be considered as critical business data, which then will require a backup schedule as following:

  Backup schedule: full backup 1x per week, incremental backup daily

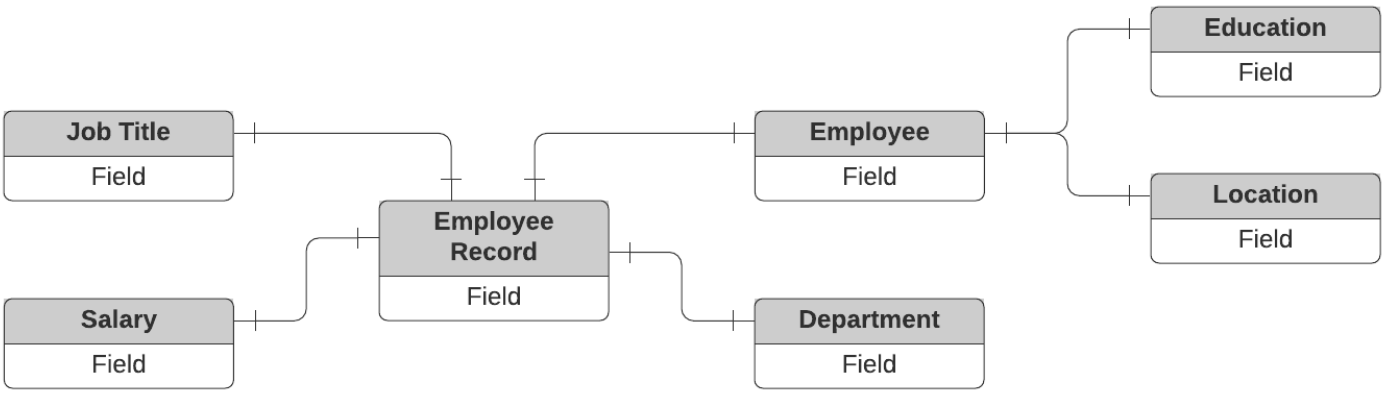
Step 2

Relational Database Design

ERD

- ## Conceptual

  This is the most general level of data modeling. At the conceptual level, you should be thinking about creating entities that represent business objects for the database. Think broadly here. Attributes (or column names) are not required at this point, but relationship lines are required (although Crow's foot notation is not needed at this level). Create at least three entities for this model; thinking about the 3NF will aid you in deciding the type of entities to create.

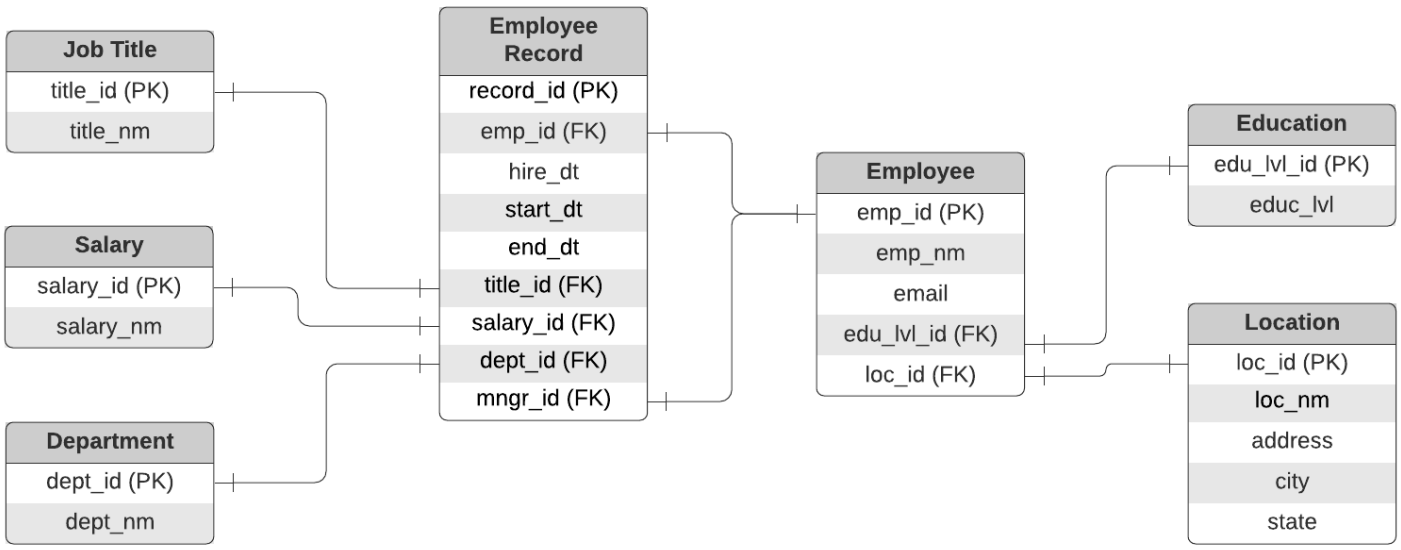


Conceptual ERD

ERD

- ## Logical

  The logical model is the next level of refinement from the conceptual ERD. At this point, you should have normalized the data to the 3NF. Attributes should also be listed now in the ERD. You can still use human-friendly entity and attribute names in the logical model, and while relationship lines are required, Crow's foot notation is still not needed at this point.

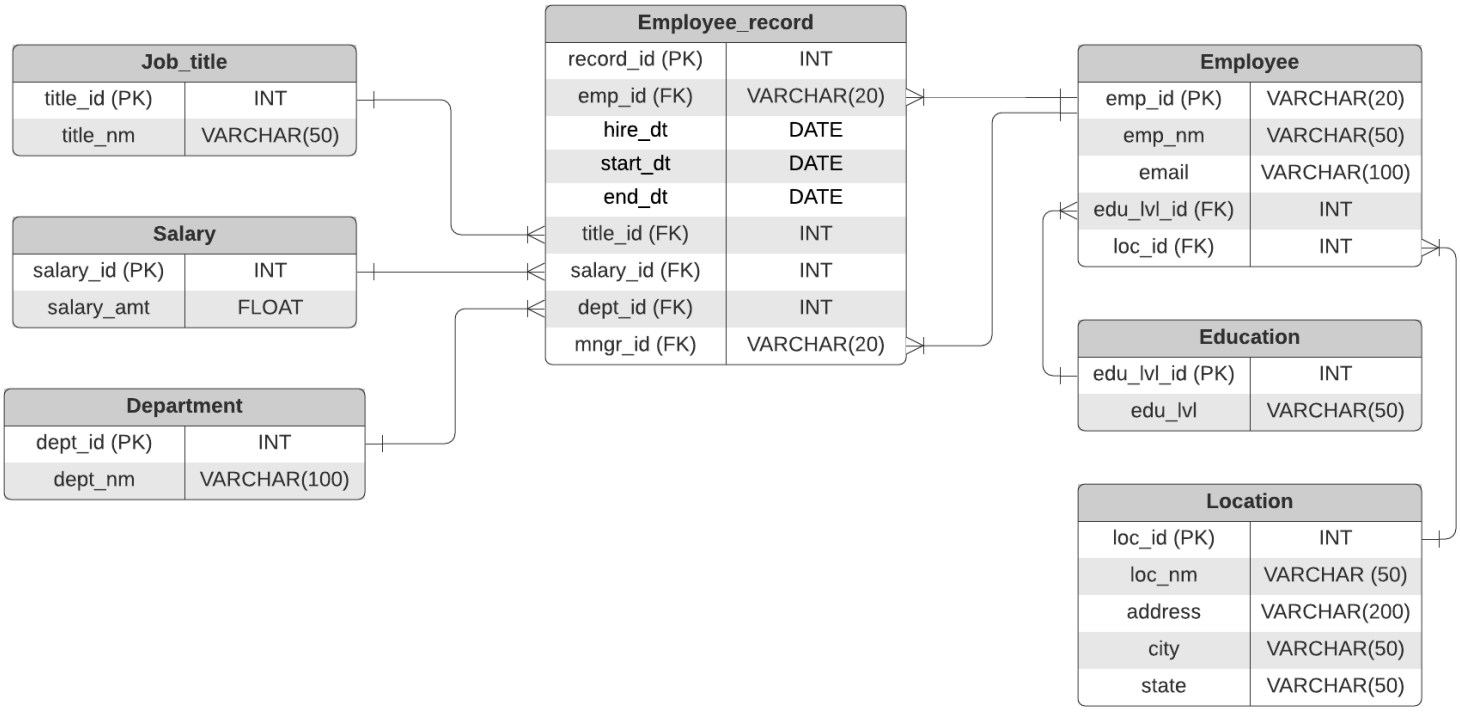


Logical ERD

ERD

- ## Physical

The physical model is what will be built in the database. Each entity should represent a database table, complete with column names and data types. Primary keys and foreign keys should also be represented here. Primary keys should be in bold type with the (PK) designation following the field name. Foreign keys should be in normal type face, but have the designation (FK) after the column name. Finally, in the physical model, Crow's foot notation is important.



Physical ERD

Step 3

Create A Physical Database

DDL

Create a DDL SQL script capable of building the database that is designed in Step 2

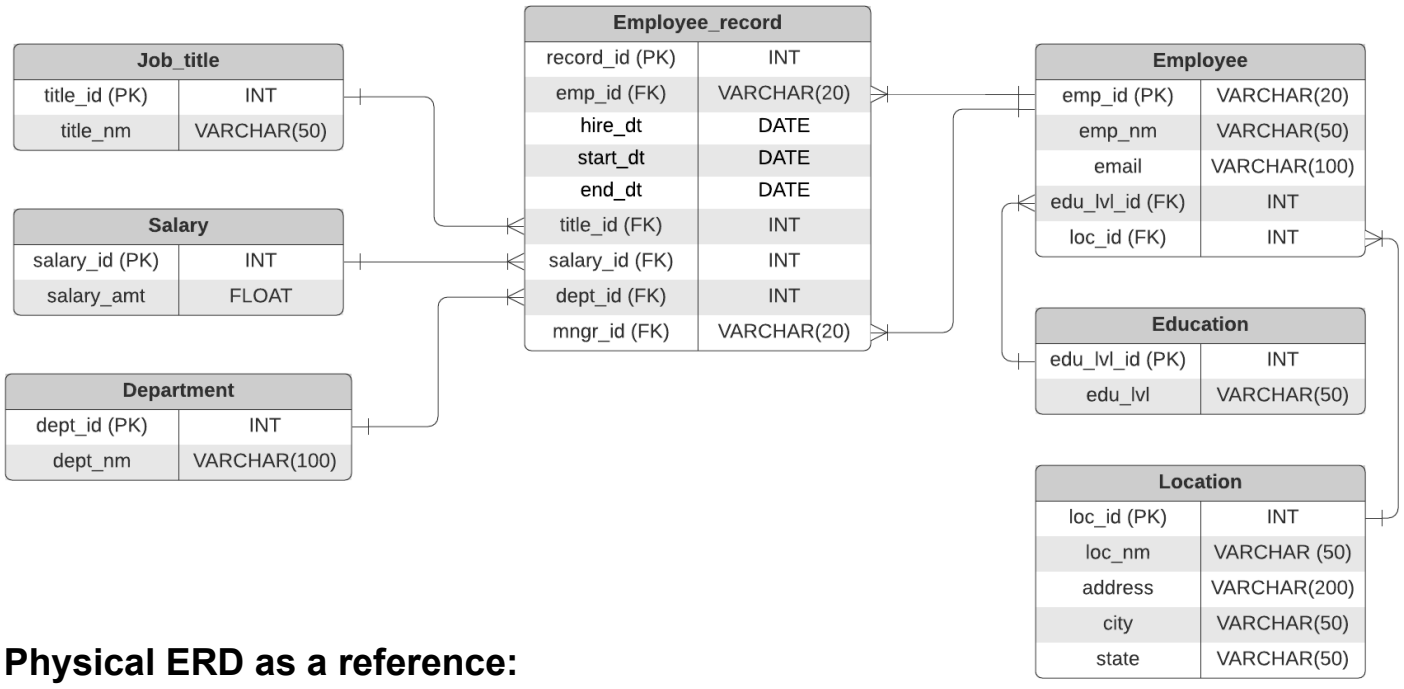
SQL files can be found in ‘data/db’ folder, named as:

- **StageTableLoad.sql** : Load data into a staging table
- **ddl_dml.sql** : Create tables from the staging table and insert relevant data following the database design

CRUD

- **Question 1: Return a list of employees with Job Titles and Department Names**

```
postgres=# SELECT rec.emp_id, emp.emp_nm, job.title_nm, dep.dept_nm
postgres-# FROM employee_record as rec
postgres-# JOIN employee as emp ON emp.emp_id = rec.emp_id
postgres-# JOIN job_title as job ON job.title_id = rec.title_id
postgres-# JOIN department as dep ON dep.dept_id = rec.dept_id;
 emp_id |        emp_nm         |         title_nm          |       dept_nm
--------+-----------------------+---------------------------+---------------------
 E87822 | Anil Padala           | Software Engineer         | Product Development
 E17054 | Tyrone Hutchison      | President                 | HQ
 E97273 | Justin Hayes          | Sales Rep                 | Product Development
 E88667 | Jennifer De La Garza  | Manager                   | Sales
 E22785 | Tami Smith            | Sales Rep                 | Sales
 E64494 | Patricia DuBois       | Software Engineer         | IT
 E27621 | Wendell Mobley        | Administrative Assistant  | Distribution
 E11678 | Colleen Alma          | Network Engineer          | Product Development
 E94552 | Geraldine Staler      | Software Engineer         | IT
 E47655 | Cody Holland          | Legal Counsel             | IT
 E54196 | Phil  Wisneski        | Sales Rep                 | Product Development
 E73518 | Cheryl Pike           | Administrative Assistant  | Product Development
```



Physical ERD as a reference:

CRUD

- **Question 2: Insert Web Programmer as a new job title**

```
postgres=# INSERT INTO Job_title (title_nm)
postgres-# VALUES ('Web Programmer');
INSERT 0 1
postgres=# SELECT * FROM job_title;
 title_id |         title_nm
----------+--------------------------
        1 | Manager
        2 | President
        3 | Database Administrator
        4 | Network Engineer
        5 | Shipping and Receiving
        6 | Legal Counsel
        7 | Sales Rep
        8 | Design Engineer
        9 | Administrative Assistant
       10 | Software Engineer
       11 | Web Programmer
(11 rows)
```

CRUD

- **Question 3: Correct the job title from web programmer to web developer**

```
postgres=# UPDATE Job_title
postgres-# SET title_nm = 'Web Developer'
postgres-# WHERE title_nm = 'Web Programmer';
UPDATE 1
postgres=# SELECT * FROM Job_title;
 title_id |         title_nm
----------+--------------------------
        1 | Manager
        2 | President
        3 | Database Administrator
        4 | Network Engineer
        5 | Shipping and Receiving
        6 | Legal Counsel
        7 | Sales Rep
        8 | Design Engineer
        9 | Administrative Assistant
       10 | Software Engineer
       11 | Web Developer
(11 rows)
```

CRUD

- **Question 4: Delete the job title Web Developer from the database**

```
postgres=# DELETE FROM Job_title
postgres-# WHERE title_nm = 'Web Developer';
DELETE 1
postgres=# SELECT * FROM Job_title;
 title_id |         title_nm
----------+--------------------------
        1 | Manager
        2 | President
        3 | Database Administrator
        4 | Network Engineer
        5 | Shipping and Receiving
        6 | Legal Counsel
        7 | Sales Rep
        8 | Design Engineer
        9 | Administrative Assistant
       10 | Software Engineer
(10 rows)
```

CRUD

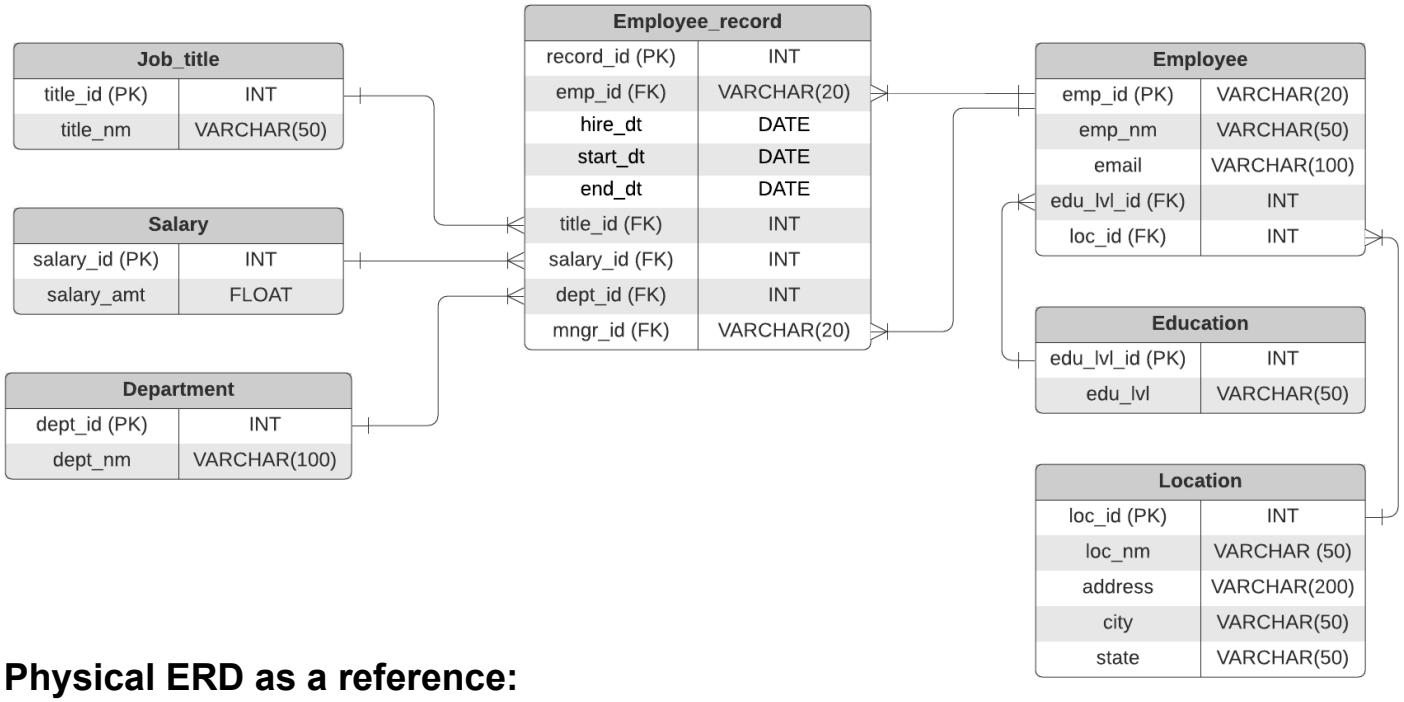
- **Question 5: How many employees are in each department?**

```
postgres=# SELECT dep.dept_id, dep.dept_nm, COUNT(*)
postgres-# FROM Employee_record as rec
postgres-# JOIN Department as dep ON dep.dept_id = rec.dept_id
postgres-# GROUP BY 1,2;
 dept_id |       dept_nm       | count
---------+---------------------+-------
       1 | Distribution        |    27
       3 | HQ                  |    13
       4 | Sales               |    41
       2 | Product Development |    70
       5 | IT                  |    54
(5 rows)
```

CRUD

- **Question 6: Write a query that returns current and past jobs (include employee name, job title, department, manager name, start and end date for position) for employee Toni Lembeck.**

```
postgres=# SELECT
postgres-#     emp.emp_nm as employee_nm,
postgres-#     job.title_nm as job_title,
postgres-#     dep.dept_nm as department,
postgres-#     mng.emp_nm as manager_nm,
postgres-#     rec.start_dt,
postgres-#     rec.end_dt
postgres-# FROM Employee_record as rec
postgres-# JOIN Employee as emp ON emp.emp_id = rec.emp_id
postgres-# JOIN Job_title as job ON job.title_id = rec.title_id
postgres-# JOIN Department as dep ON dep.dept_id = rec.dept_id
postgres-# JOIN Employee as mng ON mng.emp_id = rec.mngr_id
postgres-# WHERE emp.emp_nm = 'Toni Lembeck';
 employee_nm  |       job_title        | department |  manager_nm  |  start_dt  |   end_dt
--------------+------------------------+------------+--------------+------------+------------
 Toni Lembeck | Network Engineer       | IT         | Jacob Lauber | 1995-03-12 | 2001-07-18
 Toni Lembeck | Database Administrator | IT         | Jacob Lauber | 2001-07-18 | 2100-02-02
(2 rows)
```



Physical ERD as a reference:

CRUD

- **Question 7: Describe how you would apply table security to restrict access to employee salaries.**

  [Note] Using PostgreSQL for the database.

  We should establish database level security so the public are not permitted to access the database and only employee with a valid domain username is able to view the information.

  More importantly, a particular security restriction should be made for **Salary** table so only HR and manager-level employee can access the employee's salary information, which can be done with table-level security.

  PostgreSQL provides some queries to GRANT / REVOKE access. An example of security application can be found in Slide 25.

  More details can be found in official PostgreSQL documents following at the links below :

  - **Grant:** https://www.postgresql.org/docs/9.0/sql-grant.html
  - **Revoke:** https://www.postgresql.org/docs/9.1/sql-revoke.html

Step 4

Above and Beyond

(optional)

Standout Suggestion 1

Create a view that returns all employee attributes; results should resemble initial Excel file

Create view

```
postgres=# CREATE VIEW original_dataset AS
postgres-# SELECT
postgres-#     emp.emp_id,
postgres-#     emp.emp_nm,
postgres-#     emp.email,
postgres-#     rec.hire_dt,
postgres-#     job.title_nm as job_title,
postgres-#     sal.salary_amt as salary,
postgres-#     dep.dept_nm as department,
postgres-#     mng.emp_nm as manager,
postgres-#     rec.start_dt,
postgres-#     rec.end_dt,
postgres-#     loc.loc_nm as location,
postgres-#     loc.address,
postgres-#     loc.city,
postgres-#     loc.state,
postgres-#     edu.edu_lvl as education_level
postgres-# FROM employee_record as rec
postgres-# LEFT JOIN job_title as job ON job.title_id = rec.title_id
postgres-# LEFT JOIN salary as sal ON sal.salary_id = rec.salary_id
postgres-# LEFT JOIN department as dep ON dep.dept_id = rec.dept_id
postgres-# LEFT JOIN employee as emp ON emp.emp_id = rec.emp_id
postgres-# LEFT JOIN employee as mng ON mng.emp_id = rec.mngr_id
postgres-# LEFT JOIN education as edu ON edu.edu_lvl_id = emp.edu_lvl_id
postgres-# LEFT JOIN location as loc ON loc.loc_id = emp.loc_id;
CREATE VIEW
```

Select (load) view

```
postgres=# SELECT * FROM original_dataset LIMIT 3;
 emp_id |     emp_nm      |           email            |  hire_dt   |     job_title     | salary |     department      |   manager    |
 start_dt  |   end_dt   | location |       address       |  city  | state |  education_level
--------+-----------------+----------------------------+------------+-------------------+--------+---------------------+--------------+-
----------+------------+----------+---------------------+--------+-------+------------------
 E10033 | Jermaine Massey | Jermaine.Massey@TechCorp.com | 2016-03-07 | Software Engineer | 111681 | Product Development | Conner Kinch |
2016-03-07 | 2100-07-08 | HQ       | 1 Tech ABC Corp Way | Dallas | TX    | Bachelors Degree
 E10407 | Darshan Rathod  | Darshan.Rathod@TechCorp.com  | 2018-10-08 | Sales Rep         | 180692 | Product Development | Conner Kinch |
2018-10-08 | 2100-04-05 | HQ       | 1 Tech ABC Corp Way | Dallas | TX    | Bachelors Degree
 E11678 | Colleen Alma    | Colleen.Alma@TechCorp.com    | 2001-12-26 | Network Engineer  |  76913 | Product Development | Conner Kinch |
2001-12-26 | 2100-03-27 | HQ       | 1 Tech ABC Corp Way | Dallas | TX    | Associates Degree
(3 rows)
```

Standout Suggestion 2

Create a stored procedure with parameters that returns current and past jobs (include employee name, job title, department, manager name, start and end date for position) when given an employee name.

We are creating and executing the database on PostgreSQL environment. Stored procedure is not available in PostgreSQL. Therefore we use function instead:

Syntax to create a function returning table

```
postgres=# CREATE FUNCTION employee_job_history (employee_name varchar(50))
postgres-# RETURNS TABLE (
postgres(#     employee_name varchar(50),
postgres(#     job_title varchar(100),
postgres(#     department varchar(50),
postgres(#     manager varchar(50),
postgres(#     start_dt date,
postgres(#     end_dt date
postgres(# ) AS $$
postgres$# SELECT
postgres$#     emp.emp_nm,
postgres$#     job.title_nm,
postgres$#     dep.dept_nm,
postgres$#     mng.emp_nm,
postgres$#     rec.start_dt,
postgres$#     rec.end_dt
postgres$# FROM employee_record as rec
postgres$# LEFT JOIN employee as emp ON emp.emp_id = rec.emp_id
postgres$# LEFT JOIN employee as mng ON mng.emp_id = rec.mngr_id
postgres$# LEFT JOIN job_title as job ON job.title_id = rec.title_id
postgres$# LEFT JOIN department as dep ON dep.dept_id = rec.dept_id
postgres$# WHERE emp.emp_nm = employee_name;
postgres$# $$ LANGUAGE SQL;
CREATE FUNCTION
```

Calling the function

```
postgres=# SELECT employee_job_history ('Toni Lembeck');
                              employee_job_history
-------------------------------------------------------------------------------
 ("Toni Lembeck","Network Engineer",IT,"Jacob Lauber",1995-03-12,2001-07-18)
 ("Toni Lembeck","Database Administrator",IT,"Jacob Lauber",2001-07-18,2100-02-02)
(2 rows)
```

Reference:
https://stackoverflow.com/questions/7945932/how-to-return-result-of-a-select-inside-a-function-in-postgresql

Standout Suggestion 3

Implement user security on the restricted salary attribute.

Create a non-management user named NoMgr. Show the code of how your would grant access to the database, but revoke access to the salary data.

Create user NoMgr & grant access to the database

```
postgres=# CREATE USER NoMgr;
CREATE ROLE
postgres=# GRANT connect ON DATABASE postgres TO NoMgr;
GRANT
```

Revoke access to the database from public

```
postgres=# REVOKE CONNECT ON DATABASE postgres FROM PUBLIC;
REVOKE
```

Revoke access to the database from public

```
postgres=# REVOKE ALL ON TABLE salary FROM NoMgr;
REVOKE
```

References:

- https://www.postgresqltutorial.com/postgresql-administration/postgresql-revoke/
- https://dba.stackexchange.com/questions/17790/created-user-can-access-all-databases-in-postgresql-without-any-grants/