Databases Application

Logical Database Design of HR/Employment Management
Database Management System (DBMS)

Professor Haytham Qushtom
CP363 Databases
Group Assignment 10
Friday, April 7th, 2023

Group 38:

Abdul Basit [200757150]
Lubna Al Rifaie [200821590]
Harriharan Sivakumar[200676770]
Lashani Dharmasena [212378870]

Table of Contents

Assignment 1 - Updated System Overview

Introduction / System Overview:

This project revolves around the design and implementation of an employee management system web application that is composed of two sub-components: employee payroll management and employee performance management. These sub-components and the system as whole assist organisations and companies in the processes of evaluating and tracking employee performance relative to a particular product or project and performing payroll-related activities (i.e., calculation of gross wages, deductions, net pay, and generating the pay stub).

To accommodate the time constraint imposed on this project, certain assumptions regarding the scope of the system's functionalities were made:

- The performance of each employee is evaluated per project/product.
    - KPIs will be based on the project/product type.
- Each employee will be paid bi-monthly.
    - The payment cycle date is fixed and common for all employees in the organisation. This assumption is valid as it reflects general accounting principles.
- ● Each employee will have a fixed wage.

User Experience / User Interface:

To utilise the functionalities of the system, users will have to initially register an organisation in the web application and create departments. Following the latter, the superuser (the user who created the organisation) can then invite other users (i.e., employees) to join newly created departments. Certain departments, as well as users with the appropriate access level, can create projects, set payroll details, and enter performance appraisal results into the system via forms in the web application. Additionally, the employees can also download their pay stubs or their appraisal results as a PDF.

Entities:

To provide and support the functionalities discussed above, the employee management system will be composed of the following entities:

- Organisation: (Strong)
    - Creating an organisation entity is the entry point to our system. It will contain basic details about the organisation, such as the business registration and payment cycle information. The attributes of this entity, such as the payroll/payment cycle date, are important for the payroll management functionalities as they determine when and how much each employee will be paid.

Databases Application 3 CP363 - Group 38

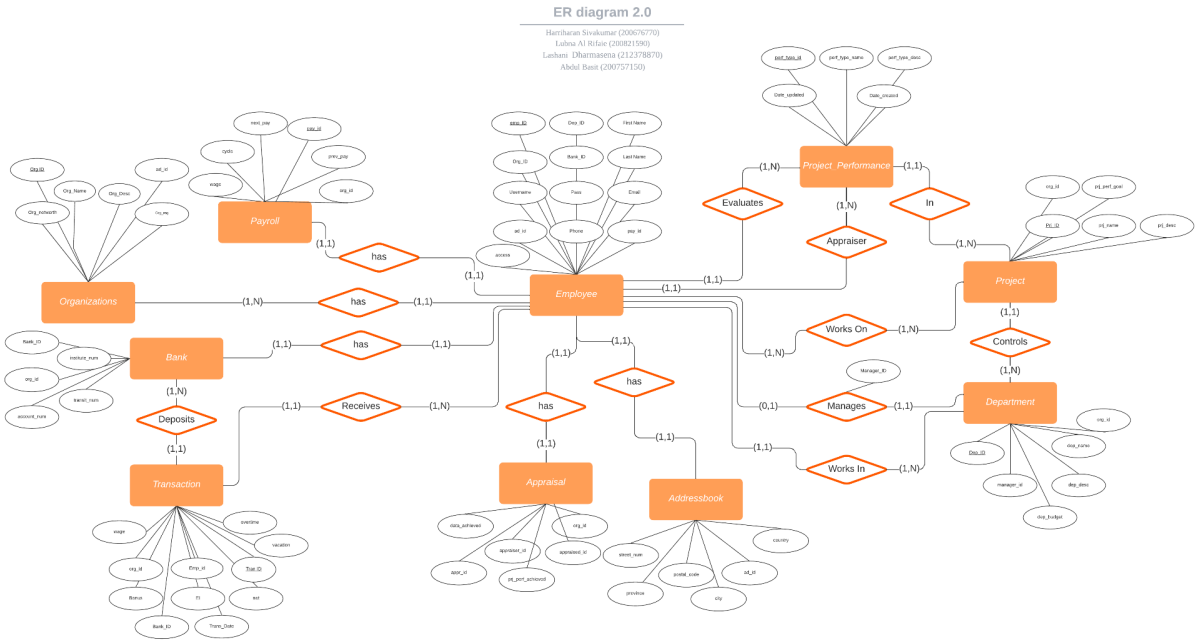
- Employee (Weak)
    - This entity contains information about each employee of an organisation, including the bank and wage details. This information will be available for the employee themselves to be accessed when necessary.
- Bank: (Weak)
    - This entity contains the bank details (e.g., branch number, account number, and bank name) of each employee. Although this entity is not accessed by any other entity, its information is used in the computation and entry of records into the payroll entity.
- Payroll: (Weak)
    - This entity consists of details such as the hours worked by an employee, their gross pay, and any applicable deductions required for calculating their net pay and generating the pay stub. The latter is made possible by this entity's relationship with the employee and organisation entity.
- Project_Performance: (Weak)
    - Whenever an employee in the organisation has their performance evaluated, the results are stored in this entity. This entity is dependent on information from the employee and project entity. Users can access information from this entity by viewing their performance results/evaluation.
- Project: (Weak)
    - This entity contains unique information about each product/project that is being undertaken by the organisation and its employees. Additionally, the evaluation of employees' performance is done using key performance indicators (“KPIs”) relative to the product/projects category. It aids in managing expectations from which employee appraisals will be entered into the performance table according to each project or product ID.
- Department: (Weak)
    - This entity contains information regarding the different departments within the organisation, the number of employees per department, as well the description, budget, and access scope of each department. Employees' access levels are based on the department entity they are associated with.
- Addressbook: (Weak)
    - This entity contains information about the employee, and organizatin address details. This includes various attributes such as their street, city, province, etc.
- Appraisal: (Weak)
    - This entity contains information about the appraised project. The attributes include, the organization, project, the appraiser and the employee who is appraised.

Relationships:

1. Employee → Department
    a. A department can manage multiple employees.
    b. One employee will be managed by one department (1:N relationship)
2. Employee → Position
    a. An employee can have one position
    b. Multiple Employees can have the same Position (1:N relationship)
3. Employee → Bank
    a. Each employee has one bank ID
    b. A single bank ID will belong to one employee (1:1 relationship).

4. Employee → Payroll
    a. Each employee will be submitted to one payroll
    b. One payroll will be submitted by one employee (1:1 relationship)
5. Employee → Project_Performance
    a. Each employee can have multiple records in the performance entity (1:N relationship)
    b. One performance record can only belong to one employee ID (1:1 relationship)
6. Project_Performance→ Project
    a. Each project will have multiple performance reports (one per employee) (1:N relationship)
    b. Each performance entity record will have one project associated with it. Performance → Employee (1:1 relationship)
7. Organization → Payroll
    a. Each organisation will be part of one payroll.
    b. One payroll will be part of a single organisation (1:1 relationship)
8. Department → Project
    a. One department controls many projects.
    b. Many projects can be controlled by one department (1:N relationship)

Assignment 2 - Updated ER Diagram



Assignment 3

Assignment 3 - Video Transcript

Hello. We're going to talk about our database project for this semester, which is the Employee Payroll and Performance Database system. I'm going to briefly talk about the project overview and the main goals of this project. Explain the entities we have in our Er diagram, all the relationships we came up with, all the attributes mentioned. At the end, we're going to show the implementation of the workbench used to create tables from the Er diagram. The focus of this project is on designing and implementing a web application for an employee management system that has two subcomponents employee Payroll Administration and Employee Performance Management. These parts and the system as a whole lets business and organisations evaluate and track employee performance in relation to a specific project or a product. To carry out payroll related tasks such as calculating gross wages, deduction, and net pay, as well as creating pay stubs. We came up with seven main entities: performance, employee, project, department, transaction, organisation and bank. Now we're going to move on to the Er diagram.

Starting from the entity Organization, which is a strong entity whose functionality does not depend on the existence of another. This entity includes attributes such as Organization ID, which is a primary key here organisation name, description, net worth address, and Payment cycle date. The attribute payment cycle date refers to the section in the first assignment where we mentioned for the purpose of this project that we assumed all employees will be paid on a single date. In order to simplify the database system, we can go ahead and look at.

The relationship with the employee table, which the organisation shares a one to many relationship with. Since one employee will work under one organisation and one organisation will have many employees. The employee table also shares a relationship with the department and therefore has Department ID and Organization ID as foreign keys. Along with other basic attributes needed to identify an employee of a company. The entity performance is used to analyse the performance of employees in projects by their manager. So the relationship is as one employee can be evaluated under one or many performance records, whereas one performance record will evaluate one employee. The same type of one too many relationship applies to the praiser, in this case, the manager who will have access to the performance table in order to evaluate employees working under them. Moving on to the project entity, the relationship with the employee table will be as one employee working on one or more projects, and one project can have many employees working on it. These projects will be used to evaluate their performance. Therefore, each project will have multiple performance reports and one performance record will be for one project an employee worked on. The project entity also has a relationship with departments, since one department can control many projects and one project will be under a single department's control.

The department entity has relations with the employee table for the managers managing each department where the manager ID is a composite key and also employees working under each department. We have also included a disjoint relationship for the employees working part time and full time by differentiating their pay as hourly wage and salary, focusing on how employees are paid. Over here, each employee will have one bank ID that is unique to a single employee. And the transaction table includes all the details necessary to pay the employee by taking into account bonuses, insurance, and vacation pay. To calculate the final net pay, we can now go ahead and see how the Er diagram will be implemented using MySQL.

So, our database management system consists of seven different tables. I'll be going over the SQL code we use to create these tables, how we alter tables to add foreign keys, as well as specific domain constraints and our disjoint constraint. So we use the Create database command to create our database, then the Create table command to create multiple entities. So, to add foreign keys to our tables, we first made sure that the reference tables did in fact exist. And then what we did is we went over to the selected table, which is an employee. We clicked on this little tool icon, which brought us to the employee table. Over here, we made sure that the attribute or the foreign key attribute we wanted to use was an int data type and it had a not null constraint. Then we went down to foreign keys and we gave it the foreign key name, the reference table. So a bank would be referenced from the bank table. And then here we once again chose the referenced data attribute from our employee table. We hit apply and then that's how we got all our foreign keys to be in the proper place, moving on to our disjoint constraint.

So we decided that this was the best option to implement our disjoint by using a domain constraint, specifically a check domain constraint. So basically what's happening here is that if the employee is a full time employee, their salary value will be not no, otherwise it would be no. And the same goes for a part time employee if their salary would be no, but their hourly wage would not be no. And then finally, just going over some of the more simple domain constraints that we use. So this one right here makes sure that the values entered are positive integers. And this is very important for net pay, which could not be or which shouldn't be in the negatives.

And here's just a copy of the same video without audio. And these are just screenshots of the tables we created.

Assignment 4

Assignment 4 - Video Transcript

Ah, hi, this is Harry and I am representing group number 38 and this is assignment four. So we created a database that is for performance analysis of employees as well as an employee management database. So to start off, I'm going to show you the create commands that we're using. So we're going to start off by creating a database and we're just going to call it CP 363 for now, but we're going to change the name and we're developing the UI. The first table we create is the organisation table as it has no foreign keys and no other entity in our system can

exist without the organisation entity since all of them are a weak entity or they depend on the organisation.

So we start off with the organisation as a primary organisation. ID is a primary key and it increments on its own so users don't have to add that every time they create an organisation. Followed by that we have the date created which is also added on its own. And then we have the organisation name which is of our char, 80 characters long and it can't be null and it must be unique. And the reason is we don't want to have organisations that have the same name.

And also in terms of legality, I'm pretty sure organisations with the same name can't be incorporated or you can't have two organisations that are incorporated with the same name. Following that we have the address description and net worth. The address and description are just text and then the net worth to the decimal. And the reason we cap it off at ten is because we want to make sure that we have a small sized organisation rather than a large organisation because that's what we're building the system towards. Lastly, we have the payment cycle which can't be null and since we're going to assume that general accounting principles are used so the payment cycle is when everybody in the organisation gets paid and the reason it can't be null is we're enforcing people to enter the payment cycle when their organisation is created.

Next up we have the project table. The reason we create this next rather than the employee is because the project table has no foreign keys which makes it easier to create first. So with the project table we have the project ID which just like the organisation increments on its own and is the primary key. And then we have the day created which is also similar to the organisation they created. Excuse me.

And then we have the project name which can be null, the description which also can't be null. Notice that we don't make the project names unique because it's possible that you can have two products of the same name. And then we have KPI goals which are also checked here. And we want to cap off the KPI goal or we want to set the KPI goal to be a percentage since we're assuming that we can have organisations coming from different industries or sectors. So the KPI for each one of those industries or sectors is going to be different.

But by making it a percentage, we can kind of ensure that the KPIs are normalised throughout the database and the KPIs cannot be nulled when the products are created because we're assuming that software engineering processes are followed when the product is created or even general engineering principles are followed. Engineering process principles are followed when products are created. Next up, we have the department which is similar to all the other tables mentioned for this, this also has a department ID, and they created which auto Increment. And update on their own. Then we have the department name, which can't be null and which must be unique because you rarely see two departments with the same name.

And then we have the department description as well as the budget. And we also ensure that the department budget is greater than or equal to zero because we don't want to have a department that's bankrupt because there's probably no point in that department existing. And then we have the bank table, which is created once again with a bank ID that's auto Incremented. On its own and which also represents a primary key. And then we have the date created, followed by that we have the Institute number, which is set to be not null and is a VAR chart of 50.

And then we have the transit number and account number, which are also VAR charts of 50. The reason we choose VAR charts rather than integers is because we want to ensure that we accept values.

It's to avoid errors that are created when users are entering numbers with hyphens in them. And then one more thing that you notice is the Institute number is not unique, whereas the transit number and the account numbers are unique because we can have multiple employees bank, multiple employees using the same bank, which would make their Institute number the same. Before I proceed, I also want to say that the bank table essentially contains the details, the account bank account details for each employee. Next up, we have the employee tables creation command, which is one of the biggest ones over here. Walking down, we see that the first section is essentially the same as before or not before, the same as the same logic as the other tables attributes or columns.

So we have the employee ID, auto Increment primary key date created, which is timestamp and default, current_timestamp on update current_timestamp. And then the first name, last

name, address of the employee, username, email password. And then we have the disjoint constraint as seen in two. We have a disjunction for the type of the employee. So we have a full time employee or part time employee and depending on that, they either have an hourly wage or a salary.

So we set these to be default as null and they're updated later on when you're inserting. But the full time employees have a salary, whereas the part time employee contract employees are on hourly wage. We also want to check that this condition is held. So if the employee type is a full time employee, make sure that they have an employee salary and not an hourly wage. The inverse is also true for the part time employees.

Then we create the foreign keys for the organiser. We reference the foreign keys, which are the organisation ID, department ID. And bank ID. And one thing I want to point out is we do set the bank ID to be default null and that's because employees who are on their onboarding process might not have their bank account information entered into the organisation yet, but they can have it entered in later.

Once this is done, we alter the department table and we create the Manager ID foreign key and we reference it with the employee ID. The reason this is done later on rather than within the Create Department table is because when we are creating the Department table, there is no employee table created yet. So if you try to run it with the foreign key inside the department, it would create a circular foreign key error. Next up we have the performance table. The first section of that is essentially the same and we want the KPI achieved to be a percentage as well, as mentioned earlier.

And we also associate the performance appraisals with projects. And another thing is we have the Appraiser ID and Appraised ID which are essentially employee IDs. But we don't want these two to be the same because you can't have a person who appraises who does their own performance appraisal. Next up we're going to look at the transaction tables, create commands. So the first two columns, just the ID and the data, are essentially the same.

And then the initial wage is kind of your gross pay, which must be greater than zero if you're going to pay them. And then we have the deductions, which have no conditions on them

because they could be negative, and then the additions which must be positive or zero. Finally we have the net pay, which cannot be null because it could be zero, but it could not be null. And then we have the foreign keys associated with this. So when we run this, you're going to notice that everything works fine, no errors pop up.

And then if you refresh the schemas, we're going to see the CPT 60 database and all the tables associated with it. I'm just going to pop open all of them just to show that they created what we expected of them, what we expected from them. So open all of that. Starting from transactions, everything is correct. Project Performance Organization Employee I'm going to scroll so you can see the rest of it.

Department and bank so once we have the tables created in our database, we're going to insert values into them. So we created a bunch of sample organisations. Following this, we create the department with the null as the manager ID as I showed before. This allows us to create the department first before we create the employee, which would also be the sequence that our software is going to follow. And then we create the employee.

We create multiple employees here. Two of them are full time, one of them is part time. And as you can see here, the null values are set based on that. So the full time employees have a null for the hourly wage and have a value for the salary. And it's the inverse is true for the part time employee.

And also notice that their bank accounts are set to null because we haven't added the entries for them on the bank table yet, which is done right here. Institute numbers are set and the transit numbers are set and the account numbers are set. And once we have the bank account details entered into the system, we then update the employee table and we match the bank entry. And also once we have the employees in their bank set up, we can match the department, we can set the managers for the department or we could also do this before we enter the bank staff. Next up we create two sample projects and we set their KPIs to be six and 8%, which is really lowballing it, but we're assuming that's going to be the case in this sample.

And then we create performance entries. Notice that the appraiser ID and appraised ID are different and then we set the products for them. As you can see, the product IDs are automatically created for these as they are set as auto Increment. And they're also the primary key so it's just easier to refer to them by that. And then we also create two sample transactions.

When I run this you should see all the inserts work. Beautiful. So now when I open up the employee table, you should see the employees we just created with all their personal information, the nulls matching up where they're supposed to based on the employee type and their associated bank IDs. So if you go look at the bank table, you see the bank details for them and you could also go look at some things, just the transaction table which in this case we put null for overtime pay. And then we also have the employee IDs, bank IDs.

We could go look at the performance appraisals. As you can see here, everything seemed to be working. So since we already covered creating an insert, we're now going to look at joins. So these are two of our joint queries. Each of them aims to achieve different things.

So the first thing over here is we're joining the project, the employee and the performance table. And we aim to see the KPI goal versus the KPI achieved per employee per project. And then the second one, we paired the performance transactions project and Employee table, and we essentially aim to see the KPI achieved and bonuses associated for each employee based on their first name. So if I run these, you should see two tables pop up. So the result one contains the KPI goal versus KPI achieved for each employee.

And then result two contains the KPI achieved and the bonus pay associated with it for each employee.

And as you can see here, we match based on the project ID and the Performance ID and the project ID. I believe.

Yes, perfect. Over here as well. Employee ID and appraise ID and product ID.

Next up we have the queries. So these queries are rather simpler compared to the joint statements that we wrote. The joint queries that we wrote. Okay, I'll explain the first one. So we select all everything from organisations, so we select all the organisations and we order them in alphabetical order based on their organisation name.

The second table which will be generated from the table. The second query's result would be the number of appraisals per appraiser. And finally we have the number of appraisals per appraiser and we filter it to ensure that the result only shows appraisers who appraised more than two people or employees.

So the group ensures that it's grouped by the appraiser ID. So you don't have repetitive appraiser IDs and you're ensuring that you're seeing it per appraiser. So if I run this, three tables are created. The first one contains all the organisations sorted in order. Next up, we have the appraiser ID and the number of appraisers.

Finally we have the count of the appraised ID and the appraiser. And it's essentially the same as the previous table, but it's filtered to be only greater than two. So in this one, we see that you have an appraiser ID. Three who appraised only one. That one gets removed in this one not removed, but that one gets ignored in this one because it doesn't meet our having condition.

So let's close all of that. And next up, we're going to look at the views. So I have my views written down on a notepad here for convenience.

I'm just going to pull up the correct file real quick.

Perfect. So we have the HR view at first, which shows the employee performance in a project relative to the project's expected performance. So this essentially is sort of a join. You join the Performance Project and Employee table and you match it based on the appraised ID and the project ID. So if I run this, you should see what we expect, and we order it by the first name in alphabetical order.

So running this, you're going to see the view that's generated here. So if I open that up, you see the Performance ID department ID. So you can kind of see what the performance of each employee is by the department. And, yeah, you see all their appraisals and you see the notes related to them. This is kind of the gauge overall what department is doing good and who is responsible for that department doing good in their respective projects.

Next up, we have a view, which is the manager's view. And one more thing I want to point out before I PROCEED is in all of these views that we create, we ensure that the personal information of employees, such as their phone number, et cetera, is kept private so it's not displayed on the view, which is the same for the case of the manager view. So I'm going to create the manager view right now, so you can see this. So I created the manager view. As you see, we only grab the first name, last name and the type of the employee.

Everything else is left as is. We don't really put it in this view. And then we have the department name, we sort it by the department name to be in alphabetical order and then we pair it based on the department ID.

Perfect. So when we apply this and run it, you should see the table. That is the view that we just created. So Manage view gives you an overall view of who's working in which department. Finally, we have the department body, which also shows you that, which shows you the number of employees per department and the budget allocated for that department.

So this is more of an accounting type table, or we can call it the table, forgive me, the Accountant View. So this one pairs up the employee and the department and it shows you the department name, the number of employees in that department, and the budget allocated towards that department. So we're running it, everything works fine. So if I go to the accountant's view, you should see the number of employees in each of those departments and the budget allocated to them. I could explain the commands we used.

Like I said, this is basically you're joining them and you're pairing them up based on the department IDs. And then you also get the count of the usernames and you set the column name for that to be the number of employees and you grab the budget from the department table.

So, yeah, like I said before, this gives an accountant in the organisation a rough idea of how much money is being spent or how much money is being allocated for each department. We can also further make this better by adding budget spent, which could be created by joining the account sorry, the transaction stable and subtracting that from the budget, essentially. So, yeah, that's basically it for all our queries, our Joins join commands, our create commands, and our insert commands, as well as the views that we created. And all of these views have two or more tables associated with them as you can see in this file over here, you have the employee in the department. You have an employee in the department.

You have the employee performance and the project. So I really hope you enjoyed our presentation and hope you have a good one. Thank you.

SQL Code for Assignment 4 submitted as separate files.

Assignment 5

Assignment 5 - Video Transcript

The query that I have for the keyword exists represents an employee who is also a manager of our department. So if you go back to the Er diagram, we see that an employee could manage a department and they will be the manager of that department. Going back to the department table, we see manager ID and as one and three and employee ID. One and three. So employee ID one john Doe and employee ID three elon Musk. They are the managers of the department's Marketing and Finance. So that's what is represented here. In this query, select Distinct Employee first name, employee last name from Employee table where the employee ID is equal to the manager ID in department tables. If you run this query, we see John Doe and Elon Musk, which is what we want. The second query that we have, it counts the number of employees currently working on a project that is in progress. So select count distinct press ID. We don't want the Press ID to be repeated from the Performance Table where the Employee ID is equal to a Press ID in the Performance Table and the performance nodes say in Progress. So if you go back to the Performance Table we see that there are two employees working in a project that is in progress, one and two, and we want the count to be two.

So if you run this, we see the count as two. So the third query lists the employees who have been evaluated more than once. So basically, select Employee first Name and count the Performance ID from Performance Table as it will be represented under a number of evaluations from Performance Table and we join the tables Employee and Performance using the appraise ID as a foreign key, which is in Performance Table and Employee ID, which is the primary key. Employee Table and you group by first name and count the number of Performances performance Evaluations, which is greater than one. So if you go back to the Performance Table we see the Employee One employee, employee One who has Employee ID One, they've been evaluated once and the second employee with Employee ID number Two, they've been evaluated three times and Employee Three they've also been evaluated once. So if you run this so this.

Query finds the total number of transactions and the total net pay for each bank, including banks with no transaction. The query uses two left joins to include banks with no transactions and employees who have not set up a bank account, and a sub query to include a row for banks with no transactions. It also uses the distinct keyword to avoid duplicate rows in the sub query. Moving on. This query here finds the number of projects each department is responsible for, including departments with no projects, and it uses two left joins to include departments with no projects and employees who are not assigned to any projects and a union to include a row for the total number of projects in the organisation. And then finally this BigQuery, what it does is it finds the top performing employees in each department and the overall top performing employees in the organisation based on their KPIs achieved. So the way it works is it starts with a select statement separated by union. So the first statement finds the top performing employees in each department by joining the employee department and performance tables. And then we use the group by clause to group the results by having by department and employee and then the having clause to filter for employees whose KPIs achieved are greater than or equal to the average KPIs achieved in their department.

Then the second statement finds the top overall performing employee by joining the employee and performance tables and then uses group by and having to filter for employees whose KPIs achieved are greater than or equal to the overall average KPIs achieved in the organisation. I'm going to go ahead and run all the queries now. So starting with this one. So

right now the output is nothing because we don't have any performance that is inputted after or before sorry, after 2022. Moving on to the second query. As you can see this shows the number of projects each department has had sorry, has and then finally the final query. Here we go. It finds the total number of transactions and the total net pay for each bank including the banks that have no transactions.

Of two parts separated by the minus keyboard keyword. Sorry. The first part is the select statement that joins the bank employee and transaction tables and groups the results of the institution number column of the bank table. It uses left join to include all the uses left join to include all the bank banks even if they don't have any employees or transactions. The resulting table includes the institution number column as well as the count of transactions and the sum of the net pay of each bank. The second part is a select statement that joins the bank transaction tables that groups the results by the institution number column of the bank table. This query only includes banks that have at least one transaction. The resulting table also includes the counts of transactions and the sum of the net pay of each bank and the minus keyword is used to subtract the second query to the first query resulting in a table that includes only the bank that doesn't have any employees who have completed training. The second query is employees who have not received any bonuses. This cure uses a minus keyboard to retrieve the employees who have not received any bonuses.

The first select statement retrieves all the employees from the employee table and the second select statement retrieves all the employees who have received bonus by joining the bonus and employees tables. The minus keyword is used to retrieve employees who have the first select statement but not in the second select statement. So first select the Employee ID, first name and last name. And then from Employees, select the Employee ID, first name and last name from the employee table. Minus the minus operator returns all the rows returned to the first select statement that are not also returned by the second select statement. And then here we have the selected employee. Employee ID first name, last name from Bonus join Employee on Employee ID, which will equal Employee ID, the first name and last name from the bonus table, but only for employees who have received bonus. Received a bonus. The Join condition links the Employee ID column in the Bonus table to the ID column in the Employee table. By taking the minus of these two select statements, we get all employees

who are in the employee table but not in the bonus table, for example, who haven't received any bonuses.

These employees have their Employee ID, first name and last name that was returned by the curie.

All right, so before we start discussing more about each of the queries and showing them in particular, we're just going to briefly go over the user interface. So let's launch it off. You did this one completely in Python, and when you get started, you have an option to either update an existing organisation or create a new one. For this example, let's update an existing one. Let's pick one from the list that's given. It shows you which organisation you're handling, the different details about it, and then you can browse in here. View different departments. Add a new department. Remove a new department. Let's add a new one. For instance, the test department. Let's say some values there. So now we see it has a new added department. If you try to remove the department, you just have to enter the ID of the department. So we created Department Six. As you can see, it's removed a couple of the functions that are handy in here. Let's see you're trying to manage your payroll. Compute the different payroll entries. You can edit or remove the payroll entry. For instance, let's try to remove the payroll entry that was lastly made.

Boom. As you can see, it's gone. Or you could edit one or add a new payroll entry. Let's say you're trying to add one for today's spam. Bunch of values in there. To get this through real quick, ask you for your Employee ID. Let's say your employee has one bank ID. Let's say one adds it to payroll entries. As you can see, the entry that we just created, let's exit back more. Let's look at performance. View the different performance evaluations. You could add or remove an evaluation. For now, we're just going to leave that one in there so we could go look at it later. The other stuff is pretty similar. I could showcase the code after I'm done showing you the user interface. But I want to highlight special queries because these are the extra long queries that were specifically for this assignment. So one of them is View. Managerial employees. So these are employees which have a managerial status and then count of different employees, count of employees with multiple valuations, top performers, projects by department and as well as your transaction summary and seven exit or go Back.

As you can see, it's a basic user interface, but the back end is very dynamic, allowing you to pile up all of these without having an excessively large file.

So I can also show you how it is to create a new organisation. So let's say test organisation. Let's just throw in something new, give it a balance of XYZ, throw in a date, then boom, you have a new organisation that was just created. And you can double check this by going and seeing the Update Organization. Our organisation is still here. So let's say test organisation. Then it lets you back into the menu to go do your admin stuff for this organisation. So that's about it. For our user interface, I can show you the backside, the back end of it. So we have our special query stored in a separate file. That's just to keep the code tidy. And then we have our own database component which was built to handle the MySQL stuff. It does it through the MySQL package, but it has its own handy dandy functions that we use which are custom for our back end. So our system is the biggest file over here, which contains different menus and it handles different database aspects within the system. So you're adding your payroll stuff, viewing different payroll entries as well as your special queries.

The reason they're stored here is because they're just extremely long. And then domain driver code is simply your different input. So definitely the hosts and passwords are stored in a private file which are not actually tracked by the repo. And then you have your different menus that are stored as constants for you to edit the code whenever you choose to. And then you have your range of While loops which hold each different menu section and call the organisation object related to it. Thank you so much for watching.

Assignment 6

Addressbook

{ad_id} → {date_created, date_updated, street_num, unit_num, street_name, city, province, postal_code, country }

Appraisal

{appr_id} → {date_created, date_updated, date_achieved, prj_perf_acheived}

Payroll

{pay_id} → {date_created, date_updated, wage, cycle, prev_pay, next_pay}

Project_performance

{**perf_type_id**} → {date_created, date_updated, perf_type_name, perf_type_desc}

Project
{**perf_type_id**} → {date_created, date_updated, prj_name, prj_desc, prj_perf_goal}

Organizations
{**org_id**} → {date_created, date_updated, org_date_created, org_name, org_address, org_desc, org_netWorth, org_reg}

Employee
{**emp_id**} → {date_created, date_updated, access, firstname, lastname, username, email, phone, pass}

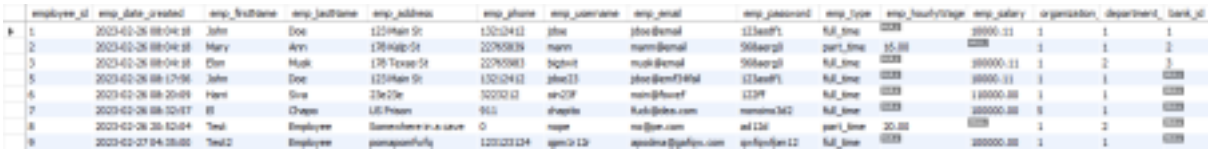
Department
{**Dep_id**} → {date_created, date_updated, dep_budget, dept_name, dept_desc, dept_budget}

Bank
{**bank_id**} → {date_created, date_updated, insititute_num, transit_num}

Transactions
{**tran_id**} → {date_created, date_updated, transaction_date, wage, EI, vacation, bonus, overtime, net}

Assignment 7- Normalisation in 3NF

Assignment 7



We have made employee ID to be the unique identifier (determinant) of the table employee along with other attributes such as employee date created, first name, last name and so on.

Employee(employee_id, emp_date_created, emp_firstName, emp_lastName, emp_address, emp_phone, emp_username, emp_email, emp_password, emp_type, emp_hourlyWage, emp_salary)
FDs: {employee_id → emp_date_created,
emp_username → emp_email, emp_password,
emp_type → emp_hourlyWage, emp_salary}

bank_id	bnk_date_created	institute_number	transit_number	account_number
1	2023-02-26 08:04:18	001	11242	123123123123
2	2023-02-26 08:04:18	002	11363	11231442342256
3	2023-02-26 08:04:18	003	12363	1123144231256

Another table created with bank ID as a primary key, uniquely identifies an employee's banking information along with the creation date of the entry.

Bank(bank_id, bnk_date_created, institute_number, transit_number, account_number) FDs: { bank_id → bnk_date_created, institute_number, transit_number, account_number}

In the case where we want to add the date the employee got paid for, we can add a new attribute called transaction date to the bank table. But, it will introduce a transitive functional dependency because the transaction date depends on the bank ID attribute, which in turn depends on the employee ID attribute.

To avoid this, we have created a third table called transaction for the relationship between these two tables, that keeps track of the payments made for each employee.

transaction_id	transaction_date	wage	EI_pay	vacation_pay	bonus_pay	overtime_pay	net_pay	employee_id	bank_id
1	2023-02-16	100.00	0.00	0.00	0.00	NULL	100.00	1	1
2	2022-01-09	100.00	0.00	0.00	50.00	NULL	150.00	2	2
8	2023-02-26	12312.00	124.00	421.00	12.00	NULL	412.00	1	1

Here we have used the transaction ID as the primary key, which is a unique identifier for each transaction, and we have also included other attributes such as the net pay.
We have also modified the table to include employee ID and bank ID as foreign keys which will ensure that each transaction is unique to avoid the transitive functional dependency.

Transaction(transaction_id, transaction_date, wage, EI_pay, vacation_pay, bonus_pay, overtime_pay, net_pay)
FDs: { transaction_id → transaction_date
transaction_id, transaction_date → wage, EI_pay, vacation_pay, bonus_pay, overtime_pay, net_pay}

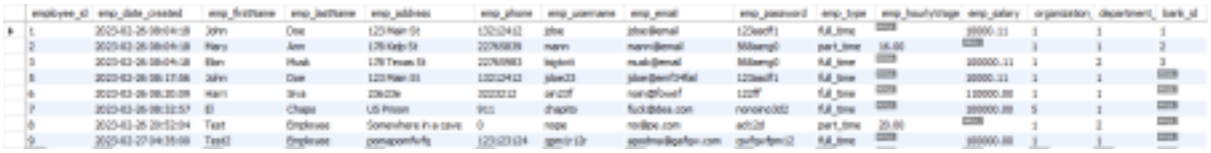
2) Decomposition to 2NF and 3NF:

1. Organisations (organization_id, org_date_created, org_name, org_address, org_desc, org_netWorth, payment_cycle)

organization_id	org_date_created	org_name	org_address	org_desc	org_netWorth	payment_cycle
1	2023-02-27 00:34:03	AmazonPrime	123 sre st	New Random Desc	1021.65	2023-06-22
2	2023-02-26 08:04:18	Microsoft	45 erb st	software development company	2554.50	2023-02-20
3	2023-02-26 08:04:18	Netflix	256 bvl st	internet entertainment services	1200.45	2023-06-18
4	2023-02-26 08:04:18	Apple	556 brown st	software development company	1054.50	2023-01-06
5	2023-02-26 08:30:11	DaBouf	Trap	Narcotics	1000000.00	2069-04-20
6	2023-02-27 04:38:26	Test Organization	423 qwerqoinqwr	New Orgaization	10000.00	2023-06-24

This table is already in 3NF/BCNF because there are no partial dependencies or transitive dependencies.

2. Employee (employee_id, emp_date_created, emp_firstName, emp_lastName, emp_address, emp_phone, emp_username, emp_email, emp_password, emp_type, emp_hourlyWage, emp_salary)



This table is already in 3NF/BCNF because there are no partial dependencies or transitive dependencies.

3. Department (department_id, dep_date_created, dept_name, dept_desc, dept_budget)

department_id	dep_date_created	dept_name	dept_desc	dept_budget	organization_id	manager_id
1	2023-02-26 08:04:18	Marketing	Responsible in identifying customer	2000.00	1	1
2	2023-02-27 00:44:47	Finance	Acquiring and utilizing money for financing	2030.00	1	3
4	2023-02-26 08:30:30	Supply	Sinolans	10000000.00	5	NULL

This table is already in 3NF/BCNF because there are no partial dependencies or transitive dependencies.

4. Bank (bank_id, bnk_date_created, institute_number, transit_number, account_number)

bank_id	bnk_date_created	institute_number	transit_number	account_number
1	2023-02-26 08:04:18	001	11242	123123123123
2	2023-02-26 08:04:18	002	11363	11231442342256
3	2023-02-26 08:04:18	003	12363	1123144231256

This table is already in 3NF/BCNF because there are no partial dependencies or transitive dependencies.

5. Transactions (transaction_id, transaction_date, wage, El_pay, vacation_pay, bonus_pay, overtime_pay, net_pay)

transaction_id	transaction_date	wage	El_pay	vacation_pay	bonus_pay	overtime_pay	net_pay	employee_id	bank_id
1	2023-02-16	100.00	0.00	0.00	0.00	NULL	100.00	1	1
2	2022-01-09	100.00	0.00	0.00	50.00	NULL	150.00	2	2
8	2023-02-26	12312.00	124.00	421.00	12.00	NULL	412.00	1	1

This table has a partial dependency of {transaction_id, transaction_date} on {wage, El_pay, vacation_pay, bonus_pay, overtime_pay, net_pay}, because the wage and pay columns are functionally dependent on the transaction_id and transaction_date.

To decompose this table to 2NF, create two tables:

Transactions1 (transaction_id, transaction_date, net_pay)

Transactions2 (transaction_id, transaction_date, wage, El_pay, vacation_pay, bonus_pay, overtime_pay)

Table Transactions1 is in 2NF and Transactions2 is in 3NF.

Transactions have the primary compound key of (transaction_id, transaction_date) because transaction_id and transaction_date functionally determine the net_pay column.

Transactions has the primary compound key of (transaction_id, transaction_date) because the combination of transaction_id and transaction_date functionally determines the other columns (wage, El_pay, vacation_pay, bonus_pay, and overtime_pay).

6. Performance (performance_id, perf_date_created, perf_notes, date_Achieved, KPI_Achieved)

performance_id	perf_date_created	perf_notes	date_Achieved	KPI_Achieved	appraiser_id	appraised_id	project_id
27	2023-02-27 03:45:04	completed	2022-04-20	50	1	2	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

This table has a transitive dependency of {performance_id} on {date_Achieved, KPI_Achieved}, because date_Achieved and KPI_Achieved are functionally dependent on performance_id through the perf_notes column.

To decompose this table to 3NF, create two tables:

Performance1 (performance_id, perf_date_created, perf_notes)
Performance2 (performance_id, date_Achieved, KPI_Achieved)

Both tables are in 3NF/BCNF.

Overall, all the tables are now in 3NF/BCNF.

Performance1 has the primary key of performance_id because it functionally determines the other columns.

Performance2 has the primary key of performance_id because it functionally determines the date_Achieved and KPI_Achieved columns.

Assignment 8- Normalisation in BCNF

Normalisation is a process of organising data in a database to reduce data redundancy and improve data integrity. Normalisation involves dividing a table into two or more tables and defining relationships between the tables. The goal of normalisation is to create tables that are in a specific normal form.

BCNF (Boyce-Codd Normal Form) is a higher level of normalisation than 3NF (Third Normal Form). A table is in BCNF if and only if every determinant in the table is a candidate

key. In other words, a table is in BCNF if there are no non-trivial functional dependencies between attributes in the table.

To verify whether a table is in BCNF, you need to identify all the functional dependencies (FDs) that exist between the attributes in the table. A functional dependency is a relationship between two attributes in which the value of one attribute determines the value of another attribute.
Once you have identified all the FDs in the table, you need to check if any of the determinants (the attributes on the left side of the FD) are not candidate keys. If there is at least one determinant that is not a candidate key, the table is not in BCNF.

If a table is not in BCNF, you need to decompose the table into two or more tables to eliminate the non-trivial FDs. Each new table should have a primary key, and you need to define relationships between the tables using foreign keys.
To modify a table to make it BCNF, you may need to add new tables and define relationships between the tables. If you add new data to the modified tables, you need to make sure that the data satisfies the BCNF requirements.

Based on the given functional dependencies(FDs), we can identify the primary keys and non-key attributes for each table.

- For the **Organizations table**, the primary key is org_id. The non-key attributes are org_date_created, org_name, org_address, org_desc, org_netWorth, and payment_cycle.

organization_id	org_date_created	org_name	org_address	org_desc	org_netWorth	payment_cycle
1	2023-02-27 00:34:03	AmazonPrime	123 sre st	New Random Desc	1021.65	2023-06-22
2	2023-02-26 08:04:18	Microsoft	45 erb st	software development company	2554.50	2023-02-20
3	2023-02-26 08:04:18	Netflix	256 bvl st	internet entertainment services	1200.45	2023-06-18
4	2023-02-26 08:04:18	Apple	556 brown st	software development company	1054.50	2023-01-06
5	2023-02-26 08:30:11	DaBouf	Trap	Narcotics	1000000.00	2069-04-20
6	2023-02-27 04:38:26	Test Organization	423 qwerqoinqwr	New Orgaization	10000.00	2023-06-24

- For the **Employee table**, the primary key is emp_id. The non-key attributes are emp_date_created, emp_firstName, emp_lastName, emp_address, emp_phone, emp_username, emp_email, emp_password, emp_type, emp_hourlyWage, and emp_salary.

employee_id	emp_date_created	emp_firstName	emp_lastName	emp_address	emp_phone	emp_username	emp_email	emp_password	emp_type	emp_hourlyWage	emp_salary	organization_	department_	bank_id
1	2023-02-26 08:04:18	John	Doe	123 Main St	13212412	jdoe	jdoe@email	123asdf1	full_time	NULL	10000.11	1	1	1
2	2023-02-26 08:04:18	Mary	Ann	178 Kelp St	22765839	mann	mann@email	568aerg0	part_time	16.00	NULL	1	1	2
3	2023-02-26 08:04:18	Elon	Musk	178 Texas St	22765983	bigtwit	musk@email	568aerg0	full_time	NULL	100000.11	1	2	3
5	2023-02-26 08:17:56	John	Doe	123 Main St	13212412	jdoe23	jdoe@emf34fail	123asdf1	full_time	NULL	10000.11	1	1	NULL
6	2023-02-26 08:20:09	Harri	Siva	23e23e	3223212	oin23f	noin@fowef	122ff	full_time	NULL	110000.00	1	1	NULL
7	2023-02-26 08:32:57	El	Chapo	US Prison	911	chapito	fuck@dea.com	nonoino3d2	full_time	NULL	100000.00	5	1	NULL
8	2023-02-26 20:52:04	Test	Employee	Somewhere in a cave	0	nope	no@pe.com	ad12d	part_time	20.00	NULL	1	2	NULL
9	2023-02-27 04:35:00	Test2	Employee	pomapomfwfq	123123124	qpm1r12r	apodma@gafqw.com	qwfqwfpm12	full_time	NULL	100000.00	1	1	NULL

Both tables are in at least 2NF (Second Normal Form) as there are no partial dependencies in the functional dependencies listed. However, we need to check for transitive dependencies to determine if they are in 3NF (Third Normal Form) or BCNF (Boyce-Codd Normal Form).

Assuming that emp_type refers to the type of employment (e.g., full-time, part-time, etc.), we can see that there is a transitive dependency in the Employee table.
emp_type → emp_hourlyWage, emp_salary, since the type of employment determines the wage/salary. To eliminate this transitive dependency, we can create a new table called EmploymentType with emp_type as the primary key and emp_hourlyWage and emp_salary as non-key attributes.

After normalisation, the Organizations table would remain unchanged. The Employee table would be split into two tables:

1. Employee (emp_id, emp_date_created, firstname, lastname, username, phone, email, pass)
2. EmploymentType (emp_type, emp_hourlyWage, emp_salary).

Both new tables would be in 3NF as there are no transitive dependencies or partial dependencies. The Employee table would also be in BCNF since the determinant employee_id is a candidate key and there are no non-trivial functional dependencies.

- For the **Performance table**, the primary key is performance_id. The non-key attributes are perf_date_created, perf_notes, date_Achieved, and KPI_Achieved.

performance_id	perf_date_created	perf_notes	date_Achieved	KPI_Achieved	appraiser_id	appraised_id	project_id
27	2023-02-27 03:45:04	completed	2022-04-20	50	1	2	3
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

- For the **Project table**, the primary key is project_id. The non-key attributes are prj_date_created, prj_name, prj_desc, and KPI_goal.

project_id	prj_date_created	prj_name	prj_desc	KPI_goal
3	2023-02-26 00:00:00	Test	Random Test	20
6	2023-02-27 00:46:24	NewProj	Will Be Edited	89
8	2023-03-23 00:00:00	NewProj2	Newfwef	0

Both tables are in at least 2NF (Second Normal Form) as there are no partial dependencies in the functional dependencies listed. However, we need to check for transitive dependencies to determine if they are in 3NF (Third Normal Form) or BCNF (Boyce-Codd Normal Form). There are no apparent transitive dependencies in either table. Thus, both tables are already in 3NF and BCNF.

Therefore, no further normalisation is required.

- For the **Department table**, the primary key is Department_id. The non-key attributes are dep_date_created, dept_name, dept_desc, and dept_budget.

department_id	dep_date_created	dept_name	dept_desc	dept_budget	organization_id	manager_id
1	2023-02-26 08:04:18	Marketing	Responsible in identifying customer	2000.00	1	1
2	2023-02-27 00:44:47	Finance	Acquiring and utilizing money for financing	2030.00	1	3
4	2023-02-26 08:30:30	Supply	Sinolans	10000000.00	5	NULL

- For the **Bank table**, the primary key is bank id. The non-key attributes are bnk_date_created, institute_number, transit_number, and account_number.

bank_id	bnk_date_created	institute_number	transit_number	account_number
1	2023-02-26 08:04:18	001	11242	123123123123
2	2023-02-26 08:04:18	002	11363	11231442342256
3	2023-02-26 08:04:18	003	12363	1123144231256

- For the **Transactions table**, the primary key is transaction id. The non-key attributes are transaction_date, wage, El_pay, vacation_pay, bonus_pay, overtime_pay, and net_pay.

transaction_id	transaction_date	wage	EI_pay	vacation_pay	bonus_pay	overtime_pay	net_pay	employee_id	bank_id
1	2023-02-16	100.00	0.00	0.00	0.00	NULL	100.00	1	1
2	2022-01-09	100.00	0.00	0.00	50.00	NULL	150.00	2	2
8	2023-02-26	12312.00	124.00	421.00	12.00	NULL	412.00	1	1

Both the **Department and Bank tables** are in at least 2NF as there are no partial dependencies in the functional dependencies listed. However, we need to check for transitive dependencies to determine if they are in 3NF or BCNF.

For the **Department table**, there are no transitive dependencies. Thus, it is already in 3NF and BCNF.

For the **Bank table**, there is a transitive dependency between the bank id and the non-key attributes institute_number, transit_number, and account_number. This means that the table is not in 3NF or BCNF.

To normalise the Bank table, we can split it into two tables:
Bank_1 {bank_id} -> {bnk_date_created}
Bank_2 {bank_id} -> {institute_number, transit_number, account_number}
Now, both Bank_1 and Bank_2 tables are in 3NF and BCNF.
Therefore, the Transactions table is already in 3NF and BCNF, and the Bank table is now in 3NF and BCNF after normalisation.

Here is the summary for all the given tables:

1. **Organisations:** The table has a primary key **`org_id`** and contains information about organisations including their creation date, name, address, description, net worth, and payment cycle. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.
2. **Employee:** The table has a primary key **`emp_id`** and contains information about employees including their creation date, first name, last name, address, phone, username, email, password, type, hourly wage, and salary. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.
3. **Performance:** The table has a primary key **`perf_id`** and contains information about performance including the creation date, notes, date achieved, and KPI achieved. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.
4. **Project:** The table has a primary key **`proj_id`** and contains information about projects including the creation date, name, description, and KPI goal. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.

5. **Department:** The table has a primary key **`dep_id`** and contains information about departments including the creation date, name, description, and budget. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.
6. **Bank:** The table has a primary key **`bank_id`** and contains information about banks including the creation date, institute number, transit number, and account number. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.
7. **Transactions:** The table has a primary key **`trans_id`** and contains information about transactions including the transaction date, wage, EL pay, vacation pay, bonus pay, overtime pay, and net pay. The table is in BCNF since there are no non-trivial functional dependencies where a determinant is not a superkey.

Assignment 9

Assignment 9 - Video Transcript

I will cover the section of the assignment that requires the database to be normalised and explain how our database is in their normalisation form. So normalisation eliminates redundancies, modifies anomalies and ensures dependencies are logical. We have individually gone over the steps of Normalising from first NF to third NF for each individual table created. I will go over the general steps of normalising a table by taking the Table employee as an example. For a table to be in the first normal form, it is necessary for all columns to contain atomic values, meaning attributes cannot be broken down further without any repeating groups of columns and for them to uniquely identify each record. So looking at the initial table that we created before Normalisation, we see that the attributes address is included in it, which is not entirely atomic since it can be broken down as city, state, country and so on. If these attributes were to be made atomic within the table, they will conflict with the rules of second NF and third normalisation form, which requires all non key columns to be dependent on the table's primary key, eliminating partial dependencies. And it will include transitive dependency since attributes like city and country will indirectly depend on the primary key through the attribute address.

In order to normalise the Table, we have created a new table called Address Book that breaks down the attribute address, making it atomic without any partial or transitive dependencies. Since these attributes directly depend on the address ID, which is the foreign key of the table employee, removing any redundancies that could occur. So this was just an example for the table employee and these steps were followed for all the tables to normalise the database.

Assignment 10 - Relational Algebra Statements

Select

Find all projects in motion to train current employees and achieved a performance goal of over 50%

$$\sigma_{prj\_name = \text{“Training”} \ \hat{} \ prj\_perf\_goal>50}(project)$$

Project

Find the address Id of all employees living in waterloo

$$\Pi_{ad\_id}(\sigma_{city = \text{“Waterloo”}}(addressbook))$$

Union

List all employees who completed projects in the month of February 2023

$$\Pi_{appraised\_id}(\sigma_{prj\_perf\_achieved = 100 \ \hat{} \ MONTH(date\_achieved) = \text{“February”}}(appraisal)) \cup \Pi_{emp\_id}(employee)$$

Set difference

All employees who have not been paid a salary yet

$$\Pi_{emp\_id}(Transactions) - \Pi_{emp\_id}(employee)$$

Cartesian product

Find the names of all employees working under the security department together with their leading manager’s ID

$$\Pi_{firstname,manager\_id}(\sigma_{employee.dep\_id = department.dep\_id}(\sigma_{dep\_name = \text{“Security”}}(employee \ X \ department)))$$

Rename

Find the names of all employees working under the security department together with their leading manager's ID and name the resultant relation as Dep_Secure

$$\rho_{Dep\_Secure}\left(\Pi_{firstname,manager\_id}\left(\sigma_{employee.dep\_id = department.dep\_id}\left(\sigma_{dep\_name = \text{“Security”}}(employee \times department)\right)\right)\right)$$

Set intersection

Find all the employees who are managers of the networking department

$$\Pi_{manager\_id}(\sigma_{dep\_name = \text{“Networking”}}(department)) \cap \Pi_{emp\_id}(employee)$$

Assignment operation

Find all the employees who are managers of the networking department and assign the resultant relation to a variable named manageNetwork

$$manageNetwork = \Pi_{manager\_id}(\sigma_{dep\_name = \text{“Networking”}}(department)) \cap \Pi_{emp\_id}(employee)$$

Join operation

Find the details of the managers in each department

$$MDep \leftarrow \Pi_{dep\_id,dep\_name,\ manager\_id}(department)$$

$$MEmp \leftarrow \Pi_{emp\_id,\ dep\_id,\ firstname,\ lastname}(department)$$

$$DepMgr \leftarrow MDep \ ><_{(manager\_id = emp\_id)}\ MEmp$$

Theta join

Find all employees who were paid a net total of more than the wage amount entered

$$EmpPay \leftarrow \Pi_{pay\_id,wage}(payroll)$$

$$MonthTrans \leftarrow \Pi_{tran\_id,net}(transactions)$$

$$EmpPay \ ><_{net\ >\ wage}\ MonthTrans$$

Natural join

Find the managers leading projects categorised under each department and rename the resulting table as Proj_Dept

$$Temp \leftarrow \Pi_{dep\_id}(department)$$

$$AddDepInfo \leftarrow (Temp \times project)$$

$$ManageDep \leftarrow \Pi_{dep\_id,\ manager\_id}(department)$$

$$DepProject \leftarrow \Pi_{prj\_id,\ prj\_name,dep\_id}(AddDepInfo)$$

$$Proj\_Dept \leftarrow DepProject * \rho_{(dep\_id,\ manager\_id)}(ManageDep)$$

Division

To retrieve the employee ID of the employees working on all projects

$$EmpProject \leftarrow \Pi_{prj\_id,\ appraised\_id}(appraisal)$$

$$ProjID \leftarrow \Pi_{prj\_id}(project)$$

$$Result \leftarrow EmpProject \div ProjID$$

Aggregate functions

To retrieve the number of employees and their average salary in each department and rename the resultant table as avg_sal

$$Temp \leftarrow department \times transactions$$

$$\rho_{avg\_sal(dep\_is,\ no\_of\_employees,\ ang\_sal)} dep\_id$$

$$F_{COUNT(emp\_id),AVERAGE(net)}(temp)$$

Final Tables

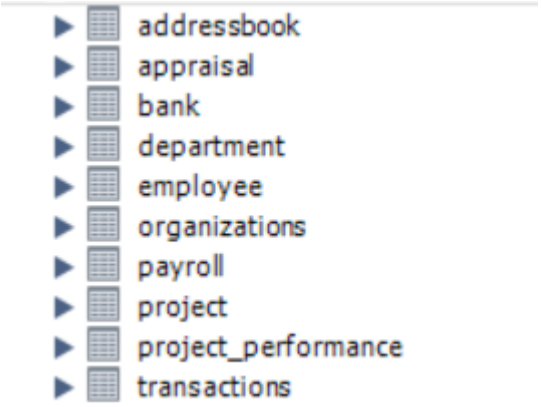


Table: addressbook

Columns:

date_created	timestamp
date_updated	timestamp
ad_id	int AI PK
street_num	smallint
unit_num	smallint
street_name	tinytext
city	tinytext
province	tinytext
postal_code	varchar(20)
country	varchar(80)

Table: appraisal

Columns:

date_created	timestamp
date_updated	timestamp
date_achieved	date
org_id	int
prj_id	int
appr_id	int AI PK
appraiser_id	int
appraised_id	int
prj_perf_achieved	int

Table: bank

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int
bank_id	int AI PK
institute_num	varchar(50)
transit_num	varchar(50)
account_num	varchar(50)

Table: department

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int
dep_id	int AI PK
dep_name	varchar(50)
dep_desc	text
dep_budget	decimal(15,2)
manager_id	int

Table: employee

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int
emp_id	int AI PK
dep_id	int
access	tinyint
firstname	varchar(50)
lastname	varchar(50)
username	varchar(50)
email	varchar(50)
phone	bigint
pass	varchar(50)
ad_id	int
bank_id	int
pay_id	int

Table: transactions

Columns:

date_created	timestamp
date_updated	timestamp
tran_id	int AI PK
org_id	int
emp_id	int
bank_id	int
wage	decimal(10,2)
EI	decimal(10,2)
vacation	decimal(10,2)
bonus	decimal(10,2)
overtime	decimal(10,2)
net	decimal(10,2)

Table: organizations

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int AI PK
org_reg	varchar(25)
org_name	tinytext
org_networth	decimal(15,2)
ad_id	int
org_desc	text

Table: project

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int
prj_id	int AI PK
prj_name	varchar(50)
prj_desc	text
perf_type_id	int
prj_perf_goal	int

Table: payroll

Columns:

date_created	timestamp
date_updated	timestamp
org_id	int
pay_id	int AI PK
wage	decimal(15,2)
cycle	int
prev_pay	timestamp
next_pay	timestamp

Table: project_performance

Columns:

date_created	timestamp
date_updated	timestamp
perf_type_id	int AI PK
perf_type_name	varchar(20)
perf_type_desc	text