247032 Stanisław Marciniak	247033 Wojciech Mokwiński
247035 Krzysztof Muszyński	247036 Bartosz Nowacki
247038 Zuzanna Pająk	247039 Tymoteusz Piwowarski

course Information Technology

semester 4

academic year 2023 / 2024

Thursday
day of the week

10:15 – 11:30
time

Database Programming
Digital Agency

Final Project Documentation

May 27, 2024
Date of submission

Result: ___

1. Project Description

1.1. General Description

The database schema for running a digital agency is designed to streamline and optimize the communication and workflow between service providers and customers. The primary objective is to structure any service offering into a process consisting of simple, well-defined steps, ensuring efficient task management and completion. The system supports the creation, management, and monitoring of tasks assigned to both customers and employees, enhancing collaboration and service delivery.

1.2. Workload Division

Table 1: Division of responsibilities between group members.

Functionality	Progress	Author
Defining Tables	implemented	Zuzanna Pająk
Providing Sample Data	implemented	Bartosz Nowacki
SQL Queries for Data preview	implemented	Krzysztof Muszyński
Defining Procedures	implemented	Stanisław Marciniak
Defining Functions	implemented	Tymoteusz Piwowarski
Defining Triggers	implemented	Bartosz Nowacki
Additional Database Structures	implemented	Wojciech Mokwiński

1.3. Project Scope

The scope of this project encompasses the design and implementation of a robust database schema that supports the aforementioned functionalities. The key components include:

1. **Service Definition and Management**: Development of tables and relationships to define and manage services and their respective steps.
2. **Order Processing**: Implementation of mechanisms to handle the creation, tracking, and management of service orders.
3. **Task Assignment and Tracking**: Design of structures to assign tasks to employees or customers, track task progress, and manage task dependencies.
4. **Input Data Handling**: Creation of tables and relations to manage various input types, store input data, and ensure data accessibility for dependent tasks.
5. **Client and Company Management**: Structuring client and company data to support collaboration and communication.

2. Functionalities

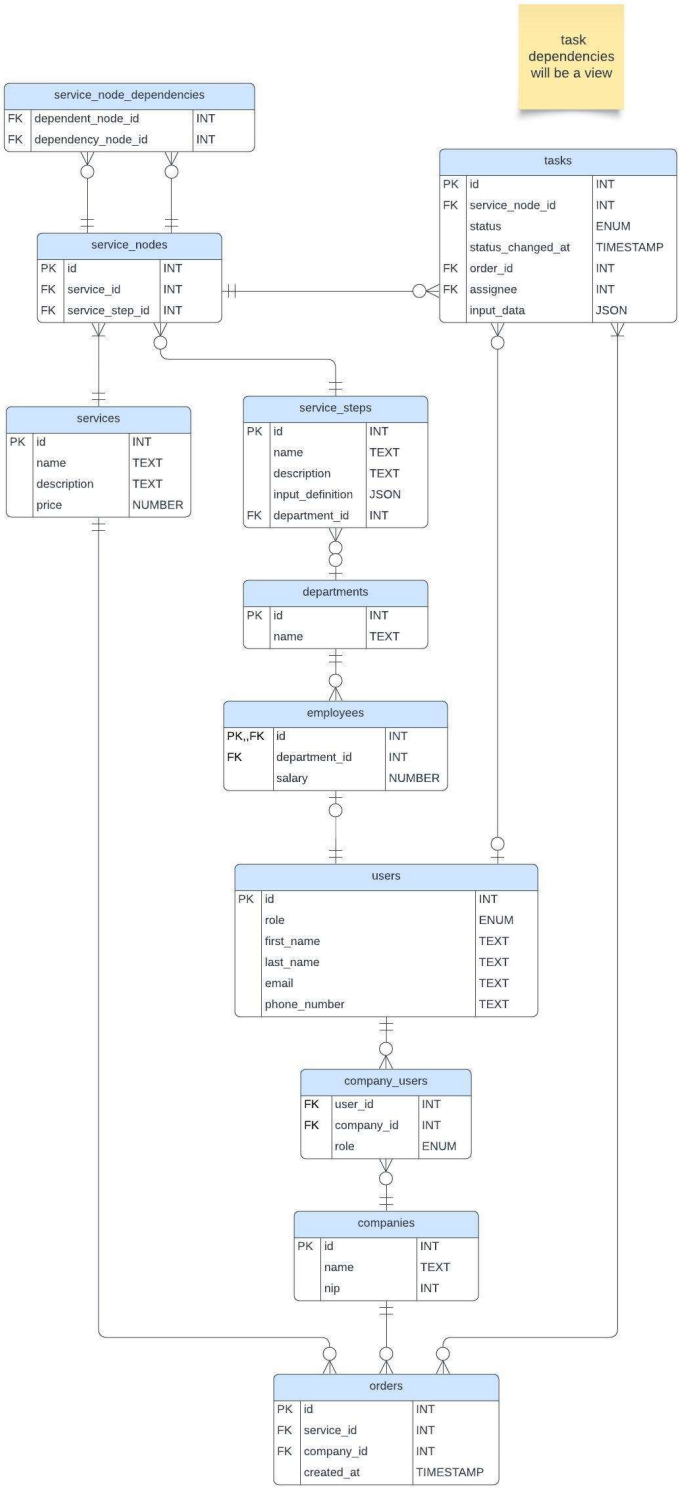

Figure 1: Relational diagram of the implemented solution.

2.1 Tables

2.1.1 Users

Table contains information about the users of the system. Each user can have either an 'admin' or 'basic' role. The table ensures uniqueness for email and phone number fields and utilizes a sequence to auto-generate user IDs.

- id: Primary key, a unique identifier for each user.
- role: Role of the user, constrained to 'admin' or 'basic'.
- first_name: The first name of the user.
- last_name: The last name of the user.
- email: The email address of the user, must be unique.
- phone_number: The phone number of the user, must be unique.

The users_id_seq sequence is defined to generate unique user IDs starting from 100.

2.1.2 Departments

Table stores information about the various departments within the digital agency.

- id: Primary key, a unique identifier for each department.
- name: The name of the department.

The departments_id_seq sequence is defined to generate unique department IDs starting from 100.

2.1.3 Employees

Table links users to their respective departments and stores their salary information.

- user_id: Primary key, references the users table.
- department_id: Foreign key, references the departments table.
- salary: The salary of the employee, with a default value of 0.

2.1.4 Companies

Table holds information about the companies associated with the digital agency's clients.

- id: Primary key, a unique identifier for each company.
- name: The name of the company.
- nip: The tax identification number of the company.

The companies_id_seq sequence is defined to generate unique company IDs starting from 100.

2.1.5 Company_users

Table establishes a relationship between users and companies, allowing for collaboration.

- user_id: Foreign key, references the users table.
- company_id: Foreign key, references the companies table.
- role: Role of the user within the company, constrained to 'owner', 'admin', or 'basic'.

The primary key is a composite of user_id and company_id.

2.1.6 Services

Table contains information about the services offered by the digital agency.

- id: Primary key, a unique identifier for each service.
- name: The name of the service.
- description: A description of the service.
- price: The price of the service, with a default value of 0.

The services_id_seq sequence is defined to generate unique service IDs starting from 100.

2.1.7 Service_steps

Table defines the steps involved in each service, specifying which department is responsible for each step.

- id: Primary key, a unique identifier for each service step.
- name: The name of the service step.
- description: A description of the service step.
- department_id: Foreign key, references the departments table.
- input_definition: A JSON object defining the inputs required for the step.

The service_steps_id_seq sequence is defined to generate unique service step IDs starting from 100.

2.1.8 Service_nodes

Table links service steps to specific services.

- id: Primary key, a unique identifier for each service node.
- service_id: Foreign key, references the services table.
- service_step_id: Foreign key, references the service_steps table.

The service_nodes_id_seq sequence is defined to generate unique service node IDs starting from 100.

2.1.9 Service_node_dependencies

This table captures the dependencies between different service nodes, indicating which nodes must be completed before others can begin.

- dependent_node_id: Foreign key, references the service_nodes table.
- dependency_node_id: Foreign key, references the service_nodes table.

The primary key is a composite of dependent_node_id and dependency_node_id.

2.1.10 Orders

This table records the orders placed for services by companies.

- id: Primary key, a unique identifier for each order.
- service_id: Foreign key, references the services table.
- company_id: Foreign key, references the companies table.
- created_at: Timestamp indicating when the order was created.

An index is created on the company_id column to improve query performance. The orders_id_seq sequence is defined to generate unique order IDs starting from 100.

2.1.11 Tasks

Table manages tasks generated from service steps, tracking their status and assigning them to users.

- id: Primary key, a unique identifier for each task.
- created_at: Timestamp indicating when the task was created.
- service_node_id: Foreign key, references the service_nodes table.
- status: The status of the task, constrained to 'to_do', 'in_progress', 'hold', 'cancelled', or 'done'.
- status_changed_at: Timestamp indicating when the task status last changed.
- order_id: Foreign key, references the orders table.
- assignee: Foreign key, references the users table.
- input_data: A JSON object containing the input data for the task.

An index is created on the order_id column to improve query performance. The tasks_id_seq sequence is defined to generate unique task IDs starting from 100.

2.2 Views

task_dependencies

Description: Displays dependencies between tasks, showing which tasks depend on others within the same order.

Reason for Implementation: This view is implemented to prevent data duplication by consolidating task dependencies into a single view, making it easier to manage and query task dependencies without redundant data storage.

Columns:

- `dependent_task_id`: The ID of the task that is dependent on another task.
- `dependency_task_id`: The ID of the task on which another task depends.

Usage:

```
SELECT * FROM task_dependencies;
```

tasks_joined

Description: Provides comprehensive details about tasks, including related service step, service, assignee, company, and whether the task is blocked.

Reason for Implementation: This view consolidates all relevant information about tasks into a single view, simplifying complex queries that require joining multiple tables to get a full picture of each task.

Columns:

- `id`: Task ID.
- `created_at`: Task creation timestamp.
- `service_node_id`: ID of the associated service node.
- `status`: Current status of the task.
- `status_changed_at`: Timestamp of the last status change.
- `order_id`: ID of the associated order.
- `assignee`: ID of the user assigned to the task.
- `input_data`: JSON input data for the task.
- `name`: Name of the service step.
- `description`: Description of the service step.
- `service_name`: Name of the service.
- `assignee_email`: Email of the assignee.
- `company_id`: ID of the associated company.
- `company_name`: Name of the associated company.
- `is_blocked`: Indicates whether the task is blocked (determined by the function `is_task_blocked`).

Usage:

```
SELECT * FROM tasks_joined;
```

employees_joined

Description: Provides detailed information about employees, including their personal details and department information.

Reason for Implementation: This view combines employee data with user details and department information, simplifying queries that need comprehensive employee information.

Columns:

- `user_id`: ID of the user (employee).
- `department_id`: ID of the department the employee belongs to.
- `salary`: Salary of the employee.
- `role`: Role of the user (employee).
- `first_name`: First name of the user (employee).
- `last_name`: Last name of the user (employee).
- `email`: Email of the user (employee).
- `phone_number`: Phone number of the user (employee).
- `department_name`: Name of the department the employee belongs to.

Usage:

```
SELECT * FROM employees_joined;
```

These views provide consolidated data from multiple related tables, facilitating easier and more efficient queries for commonly needed information, while ensuring data consistency and preventing redundancy.

2.3 Sample Data

The provided sample data populates all tables in the database, representing a digital agency's organizational structure and operations. This includes users, departments, employees, companies, services, service steps, service nodes, and orders. Below is a summary of how different entities are inserted and what users can expect from them, with a detailed focus on the sample services and their steps.

Users

The `users` table is populated with sample data representing both administrative and basic users, including their roles and contact information.

Example Data:

```
INSERT INTO users (id, role, first_name, last_name, email,
phone_number) VALUES
(1, 'admin', 'John', 'Doe', 'john.doe@example.com', '1234567890'),
...
(20, 'basic', 'John', 'Adams', 'john.adams@example.com',
'6789012345');
```

Departments

The `departments` table contains data for various departments within the agency, such as Marketing, Finance, and Engineering.

Example Data:

```
INSERT INTO departments (id, name) VALUES
(1, 'Marketing'),
(2, 'Finance'),
(3, 'Human Resources'),
(4, 'Engineering'),
(5, 'Sales');
```

Employees

The `employees` table associates users with departments and their respective salaries.

Example Data:

```
INSERT INTO employees (user_id, department_id, salary) VALUES
(1, 1, 60000.00),
(2, 2, 75000.00),
...
(6, 5, 69000.00);
```

Companies

The `companies` table includes sample data for companies that interact with the agency.

Example Data:

```
INSERT INTO companies (id, name, nip) VALUES
(1, 'ABC Corporation', 1234567890),
(2, 'XYZ Ltd.', 9876543212),
...
(4, 'GHI Enterprises', 3789123456);
```

Company Users

The `company_users` table links users to companies, specifying their roles within each company.

Example Data:

```
INSERT INTO company_users (user_id, company_id, role) VALUES
(7, 1, 'owner'),
(8, 1, 'admin'),
...
(20, 1, 'admin');
```

Services and Service Steps

Service 1. Website Development

Description:
The Website Development service offers a comprehensive approach to creating a website from scratch. This includes design, development, and deployment stages, ensuring a fully functional and visually appealing website for the client.

Steps:

3. **Design Consultation**: Discuss design preferences and requirements with the client.
    - **Department**: Sales
    - **Input**: Client preferences and requirements (text)
4. **Wireframing**: Create wireframes based on the design consultation.
    - **Department**: Engineering
    - **Input**: Wireframes (image)
5. **Frontend Development**: Develop the frontend of the website based on the wireframes.
    - **Department**: Engineering
    - **Input**: Frontend code (code)
6. **Backend Development**: Develop the backend functionalities of the website.
    - **Department**: Engineering

    - **Input**: Backend code (code)
7. **Testing and Quality Assurance**: Test the website thoroughly to ensure functionality and quality.
    - **Department**: Human Resources
    - **Input**: Test results and feedback (text)
8. **Deployment and Launch**: Deploy the website to the client server and launch it.
    - **Department**: Engineering
    - **Input**: None

Service 2. Social Media Management

Description:
The Social Media Management service includes managing the client's social media presence through content creation, scheduling posts, and analyzing performance to optimize engagement.

Steps:

1. **Client Onboarding**: Onboard the client and gather necessary account access.
    - **Department**: Sales
    - **Input**: Login and password (text)
2. **Content Creation**: Create engaging content for social media platforms.
    - **Department**: Marketing
    - **Input**: None
3. **Scheduling**: Schedule posts on various social media platforms.
    - **Department**: Marketing
    - **Input**: None
4. **Analytics and Reporting**: Analyze social media performance and generate reports.
    - **Department**: Finance
    - **Input**: Analytics report (spreadsheet link)

Service 3. Search Engine Optimization (SEO)

Description:
The Search Engine Optimization service aims to improve the client's website visibility and search engine rankings through various on-page and off-page strategies.

Steps:

1. **Website Audit**: Conduct a comprehensive audit of the client website for SEO improvements.

    - **Department**: Engineering
    - **Input**: Audit report (file)
2. **Keyword Research**: Research and identify relevant keywords for the client industry.
    - **Department**: Marketing
    - **Input**: Keyword list (file)
3. **On-Page Optimization**: Optimize website content and structure for improved search rankings.
    - **Department**: Engineering
    - **Input**: None
4. **Off-Page Optimization**: Implement strategies to improve website authority and backlinks.
    - **Department**: Engineering
    - **Input**: None

These services encompass different aspects of digital agency operations, covering web development, social media management, and SEO, each broken down into specific steps handled by different departments to ensure thorough and effective service delivery.

Orders

The `orders` table records service orders placed by companies.

Example Data:

```
INSERT INTO orders (id, service_id, company_id) VALUES
(1, 1, 1),
...
(10, 3, 4);
```

Tasks

Tasks are automatically created upon order insertion through triggers, and thus their specific details are not manually inserted here.

Conclusion: This sample data provides a comprehensive foundation for a digital agency's database, representing users, departments, employees, companies, services, service steps, and orders. The detailed service steps for each service illustrate the workflow and responsibilities across different departments, offering a clear and structured approach to managing the agency's operations.

2.4 SQL Queries for data preview

Query 1: List all users with their associated departments

Description: This query lists all users along with the departments they are associated with.

Usage:

```
SELECT u.first_name, u.last_name, d.name AS department_name
FROM users u
JOIN employees e ON u.id = e.user_id
JOIN departments d ON e.department_id = d.id;
```

Columns:

- `first_name`: The first name of the user.
- `last_name`: The last name of the user.
- `department_name`: The name of the department the user is associated with.

Query 2: Find the total number of employees in each department

Description: This query finds the total number of employees in each department.

Usage:

```
SELECT d.name, COUNT(e.user_id) AS employee_count
FROM departments d
JOIN employees e ON d.id = e.department_id
GROUP BY d.name;
```

Columns:

- `name`: The name of the department.
- `employee_count`: The total number of employees in the department.

Query 3: Get details of all tasks that are in 'to_do' status for a specific department

Description: This query retrieves details of all tasks that are in 'to_do' status for a specific department.

Usage:

```
SELECT t.*, d.name AS department_name
FROM tasks t
```

```
JOIN service_nodes sn ON t.service_node_id = sn.id
JOIN service_steps ss ON sn.service_step_id = ss.id
JOIN departments d ON ss.department_id = d.id
WHERE t.status = 'to_do' AND d.id = 5;
```

Columns:

- `id`: Task ID.
- `created_at`: Task creation timestamp.
- `service_node_id`: ID of the associated service node.
- `status`: Current status of the task.
- `status_changed_at`: Timestamp of the last status change.
- `order_id`: ID of the associated order.
- `assignee`: ID of the user assigned to the task.
- `input_data`: JSON input data for the task.
- `department_name`: The name of the department associated with the task.

Query 4: Display the average salary of employees in each department

Description: This query displays the average salary of employees in each department.

Usage:

```
SELECT d.name AS department_name, AVG(e.salary) AS average_salary
FROM employees e
JOIN departments d ON e.department_id = d.id
GROUP BY d.name;
```

Columns:

- `department_name`: The name of the department.
- `average_salary`: The average salary of employees in the department.

Query 5: List all services with their total number of steps

Description: This query lists all services along with the total number of steps involved in each service.

Usage:

```
SELECT s.name, COUNT(sn.id) AS total_steps
FROM services s
```

```
JOIN service_nodes sn ON s.id = sn.service_id
GROUP BY s.name;
```

Columns:

- `name`: The name of the service.
- `total_steps`: The total number of steps involved in the service.

Query 6: Show all companies with their latest order date

Description: This query shows all companies along with the latest order date for each company.

Usage:

```
SELECT c.name, MAX(o.created_at) AS latest_order_date
FROM companies c
LEFT JOIN orders o ON c.id = o.company_id
GROUP BY c.name;
```

Columns:

- `name`: The name of the company.
- `latest_order_date`: The latest order date for the company.

Query 7: Retrieve all tasks for a specific order including service step details

Description: This query retrieves all tasks for a specific order, including details about the service steps involved.

Usage:

```
SELECT t.*, ss.name AS step_name, ss.description
FROM tasks t
JOIN service_nodes sn ON t.service_node_id = sn.id
JOIN service_steps ss ON sn.service_step_id = ss.id
WHERE t.order_id = 1;
```

Columns:

- `id`: Task ID.
- `created_at`: Task creation timestamp.
- `service_node_id`: ID of the associated service node.
- `status`: Current status of the task.

- `status_changed_at`: Timestamp of the last status change.
- `order_id`: ID of the associated order.
- `assignee`: ID of the user assigned to the task.
- `input_data`: JSON input data for the task.
- `step_name`: The name of the service step.
- `description`: The description of the service step.

Query 8: Count of orders per company sorted by the total number of orders descending

Description: This query counts the number of orders per company and sorts the result in descending order based on the total number of orders.

Usage:

```
SELECT c.name, COUNT(o.id) AS order_count
FROM companies c
LEFT JOIN orders o ON c.id = o.company_id
GROUP BY c.name
ORDER BY order_count DESC;
```

Columns:

- `name`: The name of the company.
- `order_count`: The total number of orders placed by the company.

Query 9: List all active service node dependencies for a particular service

Description: This query lists all active service node dependencies for a particular service.

Usage:

```
SELECT s.name AS service_name, ssd.dependent_node_id,
ssd.dependency_node_id
FROM service_nodes sn
JOIN services s ON sn.service_id = s.id
JOIN service_node_dependencies ssd ON sn.id = ssd.dependent_node_id
WHERE s.id = 1;
```

Columns:

- `service_name`: The name of the service.
- `dependent_node_id`: The ID of the dependent node.
- `dependency_node_id`: The ID of the dependency node.

Query 10: List employees who are assigned tasks but have not completed any

Description: This query lists employees who are assigned tasks but have not completed any.

Usage:

```
SELECT u.first_name, u.last_name
FROM users u
JOIN tasks t ON u.id = t.assignee
GROUP BY u.first_name, u.last_name
HAVING SUM(CASE WHEN t.status = 'done' THEN 1 ELSE 0 END) = 0;
```

Columns:

- `first_name`: The first name of the employee.
- `last_name`: The last name of the employee.

Query 11: List users with the count of tasks 'to_do' and 'done'

Description: This query lists users along with the count of their tasks in 'to_do' and 'done' statuses.

Usage:

```
SELECT u.first_name, u.last_name,
       COUNT(CASE WHEN t.status = 'to_do' THEN 1 END) AS
tasks_to_do,
       COUNT(CASE WHEN t.status = 'done' THEN 1 END) AS tasks_done
FROM users u
LEFT JOIN tasks t ON u.id = t.assignee
GROUP BY u.first_name, u.last_name;
```

Columns:

- `first_name`: The first name of the user.
- `last_name`: The last name of the user.
- `tasks_to_do`: The count of tasks with 'to_do' status.
- `tasks_done`: The count of tasks with 'done' status.

Query 12: List all departments with their total salary expenditure

Description: This query lists all departments along with their total salary expenditure.

Usage:

```
SELECT d.name, SUM(e.salary) AS total_salary_expenditure
FROM departments d
JOIN employees e ON d.id = e.department_id
GROUP BY d.name;
```

Columns:

- `name`: The name of the department.
- `total_salary_expenditure`: The total salary expenditure of the department.

Query 13: Find all users who are assigned tasks in the 'in_progress' status

Description: This query finds all users who are assigned tasks in the 'in_progress' status.

Usage:

```
SELECT u.first_name, u.last_name, t.id AS task_id, t.status
FROM users u
JOIN tasks t ON u.id = t.assignee
WHERE t.status = 'in_progress';
```

Columns:

- `first_name`: The first name of the user.
- `last_name`: The last name of the user.
- `task_id`: The ID of the task.
- `status`: The status of the task.

Query 14: List all tasks along with the names of the services they are part of

Description: This query lists all tasks along with the names of the services they are part of.

Usage:

```
SELECT t.id AS task_id, t.status, s.name AS service_name
FROM tasks t
JOIN service_nodes sn ON t.service_node_id = sn.id
JOIN services s ON sn.service_id = s.id;
```

Columns:

- `task_id`: The ID of the task.
- `status`: The status of the task.
- `service_name`: The name of the service the task is part of.

Query 15: Find the total revenue generated from orders for each company

Description: This query finds the total revenue generated from orders for each company.

Usage:

```
SELECT c.name AS company_name, SUM(s.price) AS total_revenue
FROM companies c
JOIN orders o ON c.id = o.company_id
JOIN services s ON o.service_id = s.id
GROUP BY c.name;
```

Columns:

- `company_name`: The name of the company.
- `total_revenue`: The total revenue generated from the orders placed by the company.

2.5 Packages

A package is a schema object that groups logically related PL/SQL types, variables, procedures, and functions. This project incorporates the following packages:

2.5.1 tasks_pkg package

add_task

Description: Adds a new task to the `tasks` table.

Parameters:

- `p_service_node_id IN tasks.service_node_id%TYPE`: The ID of the service node associated with the task.
- `p_status IN tasks.status%TYPE`: The status of the task.
- `p_order_id IN tasks.order_id%TYPE`: The ID of the order associated with the task.
- `p_assignee IN tasks.assignee%TYPE`: The ID of the user to whom the task is assigned.
- `p_input_data IN tasks.input_data%TYPE`: Input data for the task, in JSON format.

Usage:

```
BEGIN
    tasks_pkg.add_task(1, 'to_do', 1001, 10, '{"key":"value"}');
END;
```

delete_task

Description: Deletes a task from the `tasks` table. Returns TRUE if the task was deleted and FALSE if no task with the given ID exists.

Parameters:

- `p_task_id IN tasks.id%TYPE`: The ID of the task to be deleted.

Returns:

- BOOLEAN: TRUE if the task was deleted, FALSE otherwise.

Usage:

```
DECLARE
    v_result BOOLEAN;
BEGIN
    v_result := tasks_pkg.delete_task(1001);
    DBMS_OUTPUT.PUT_LINE('Task deleted: ' || TO_CHAR(v_result));
END;
```

assign_task

Description: Assigns a task to a specified user.

Parameters:

- `p_task_id IN tasks.id%TYPE`: The ID of the task to be assigned.
- `p_assignee IN tasks.assignee%TYPE`: The ID of the user to whom the task is assigned.

Usage:

```
BEGIN
    tasks_pkg.assign_task(1001, 20);
END;
```

update_task_status

Description: Updates the status of a task and sets the `status_changed_at` timestamp.

Parameters:

- `p_task_id IN tasks.id%TYPE`: The ID of the task to be updated.
- `p_new_status IN tasks.status%TYPE`: The new status of the task.

Usage:

```
BEGIN
    tasks_pkg.update_task_status(1001, 'in_progress');
END;
```

calculate_avg_completion_time

Description: Calculates the average completion time for tasks for a given department. If no department ID is specified, the function calculates the average for all tasks.

Parameters:

- `p_department_id IN departments.id%TYPE`: The ID of the department. If `NULL`, the average completion time is calculated for all tasks.

Returns:

- `NUMBER`: The average completion time in minutes.

Usage:

```
SELECT tasks_pkg.calculate_avg_completion_time(NULL) FROM dual; --
Average for all tasks
SELECT tasks_pkg.calculate_avg_completion_time(10) FROM dual; --
Average for tasks in department with ID 10
```

2.5.2 companies_pkg package

add_company

Description: Adds a new company to the `companies` table.

Parameters:

- `p_name IN companies.name%TYPE`: The name of the company.
- `p_nip IN companies.nip%TYPE`: The NIP (Tax Identification Number) of the company.

Usage:

```
BEGIN
    companies_pkg.add_company('Tech Innovations', '1234567890');
END;
```

delete_company

Description: Deletes a company from the `companies` table. Returns TRUE if the company was deleted and FALSE if no company with the given ID exists.

Parameters:

- `p_id IN companies.id%TYPE`: The ID of the company to be deleted.

Returns:

- BOOLEAN: TRUE if the company was deleted, FALSE otherwise.

Usage:

```
DECLARE
    v_result BOOLEAN;
BEGIN
    v_result := companies_pkg.delete_company(101);
    DBMS_OUTPUT.PUT_LINE('Company deleted: ' || TO_CHAR(v_result));
END;
```

company_summary

Description: Prints all relevant information about a company, including the company's name, its employees, and its orders.

Parameters:

- `p_company_id IN companies.id%TYPE`: The ID of the company whose information is to be printed.

Usage:

```
BEGIN
    companies_pkg.company_summary(101);
END;
```

calculate_total_revenue

Description: Calculates the total revenue generated from a specified company. If no company ID is specified, the function calculates the total revenue for all companies.

Parameters:

- `p_company_id IN companies.id%TYPE`: The ID of the company. If `NULL`, the total revenue is calculated for all companies.

Returns:

- `NUMBER`: The total revenue.

Usage:

```
SELECT companies_pkg.calculate_total_revenue(NULL) FROM dual; --
Total revenue for all companies
SELECT companies_pkg.calculate_total_revenue(101) FROM dual; --
Total revenue for company with ID 101
```

2.5.3 departments_pkg package

add_department

Description: Adds a new department to the `departments` table.

Parameters:

- `p_name IN departments.name%TYPE`: The name of the department.

Usage:

```
BEGIN
    departments_pkg.add_department('Research and Development');
END;
```

delete_department

Description: Deletes a department from the `departments` table. Returns `TRUE` if the department was deleted and `FALSE` if no department with the given ID exists.

Parameters:

- `p_id IN departments.id%TYPE`: The ID of the department to be deleted.

Returns:

- `BOOLEAN`: `TRUE` if the department was deleted, `FALSE` otherwise.

Usage:

```
DECLARE
    v_result BOOLEAN;
BEGIN
    v_result := departments_pkg.delete_department(201);
    DBMS_OUTPUT.PUT_LINE('Department deleted: ' ||
TO_CHAR(v_result));
END;
```

update_department_salaries

Description: Updates employee salaries based on the department's performance rating. Different ratings (A, B, C, D) correspond to different bonus percentages.

Parameters:

- `p_department_id IN employees.department_id%TYPE`: The ID of the department whose employees' salaries are to be updated.
- `p_rating IN VARCHAR2`: The performance rating of the department. Allowed values are 'A', 'B', 'C', 'D'.

Usage:

```
BEGIN
    departments_pkg.update_department_salaries(201, 'A');
END;
```

sum_salaries

Description: Sums the salaries for employees in a specified department. If no department ID is specified, the function calculates the sum for all employees.

Parameters:

- `p_department_id IN departments.id%TYPE`: The ID of the department. If `NULL`, the sum is calculated for all employees.

Returns:

- NUMBER: The total sum of salaries.

Usage:

```
SELECT departments_pkg.sum_salaries(NULL) FROM dual; -- Total
salaries for all employees
SELECT departments_pkg.sum_salaries(201) FROM dual; -- Total
salaries for employees in department with ID 201
```

get_avg_salary

Description: Returns the average salary for employees in a specified department. If no department ID is specified, the function calculates the average for all employees.

Parameters:

- p_department_id IN departments.id%TYPE: The ID of the department. If NULL, the average is calculated for all employees.

Returns:

- NUMBER: The average salary.

Usage:

```
SELECT departments_pkg.get_avg_salary(NULL) FROM dual; -- Average
salary for all employees
SELECT departments_pkg.get_avg_salary(201) FROM dual; -- Average
salary for employees in department with ID 201
```

2.6 Triggers

insert_tasks_after_order

Description: Automatically inserts tasks based on the purchased service after an order is created. Tasks are created based on service's nodes and service step's definition.

Reason for Implementation: This trigger ensures that every new order automatically generates the necessary tasks associated with the purchased service, streamlining the workflow and ensuring consistency in task creation.

Trigger Details:

- **Trigger Type**: AFTER INSERT ON orders
- **Purpose**: To insert tasks related to the ordered service nodes.

- **Logic**:
    - A cursor `service_nodes_cursor` selects all service nodes associated with the newly ordered service.
    - For each service node, a random employee from the relevant department is selected and assigned the task.
    - A new task is then inserted into the `tasks` table with the status 'to_do'.

update_status_changed_at

Description: Updates the `status_changed_at` timestamp whenever the status of a task changes.

Reason for Implementation: This trigger ensures that there is an accurate timestamp reflecting the last time the status of a task was modified, which is crucial for tracking task progress and timelines.

Trigger Details:

- **Trigger Type**: `BEFORE UPDATE ON tasks`
- **Purpose**: To update the `status_changed_at` timestamp.
- **Logic**:
    - If the `status` of the task changes, the `status_changed_at` field is set to the current timestamp (`SYSTIMESTAMP`).

validate_service_id

Description: Ensures that service nodes that depend on each other belong to the same service.

Reason for Implementation: This trigger enforces data integrity by ensuring that service node dependencies are valid, meaning both nodes involved in a dependency must belong to the same service.

Trigger Details:

- **Trigger Type**: `BEFORE INSERT OR UPDATE ON service_node_dependencies`
- **Purpose**: To validate that dependent and dependency nodes belong to the same service.
- **Logic**:
    - Fetches the `service_id` for both the `dependent_node_id` and the `dependency_node_id`.
    - Raises an error if the `service_id` values do not match, ensuring that dependencies are correctly associated with the same service.

3. Conclusions

The presented database schema effectively supports the digital agency's need to manage complex service delivery processes. By defining services as a series of steps and managing these through tasks, the schema ensures that service delivery is both efficient and traceable. The use of JSON for input data allows for flexible and dynamic data handling, catering to diverse service requirements.

The inclusion of user roles, departments, and company affiliations provides a robust framework for task assignment and collaboration. This design ensures that tasks are allocated to the right personnel, thereby optimizing resource utilization and service quality.

Overall, this schema provides a solid foundation for the digital agency's operations, enabling scalable and efficient management of services, orders, and tasks. Future enhancements could focus on expanding the service step definitions to include more complex workflow rules and integrating more sophisticated reporting and analytics capabilities to further support decision-making and performance monitoring.

4. References

[1] S. Feuerstein, B. Pribyl: Oracle PL-SQL Programming, 6E, O'Reilly, 2014

[2] B. Rosenzweig, Elena Rakhimov: Oracle PL/SQL by Example, 5th Edition, Prentice Hall, 2015.

[3] M. McLaughlin, John Harper: Oracle Database 12c PL/SQL Advanced Programming Techniques, McGraw-Hill Education, 2014.

[4] M. McLaughlin: Oracle Database 12c PL SQL Programming, McGraw-Hill Education, 2014

[5] Oracle Database Documentation - Oracle Database Database PL/SQL Language Reference, 12c Release 2 (12.2) https://docs.oracle.com/en/database/oracle/oracle-database/12.2/lnpls/index.html