Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Information Management II (CSU34041) Project Report

Cornel Jonathan Cicai, Student: 19335265

March 25th, 2022

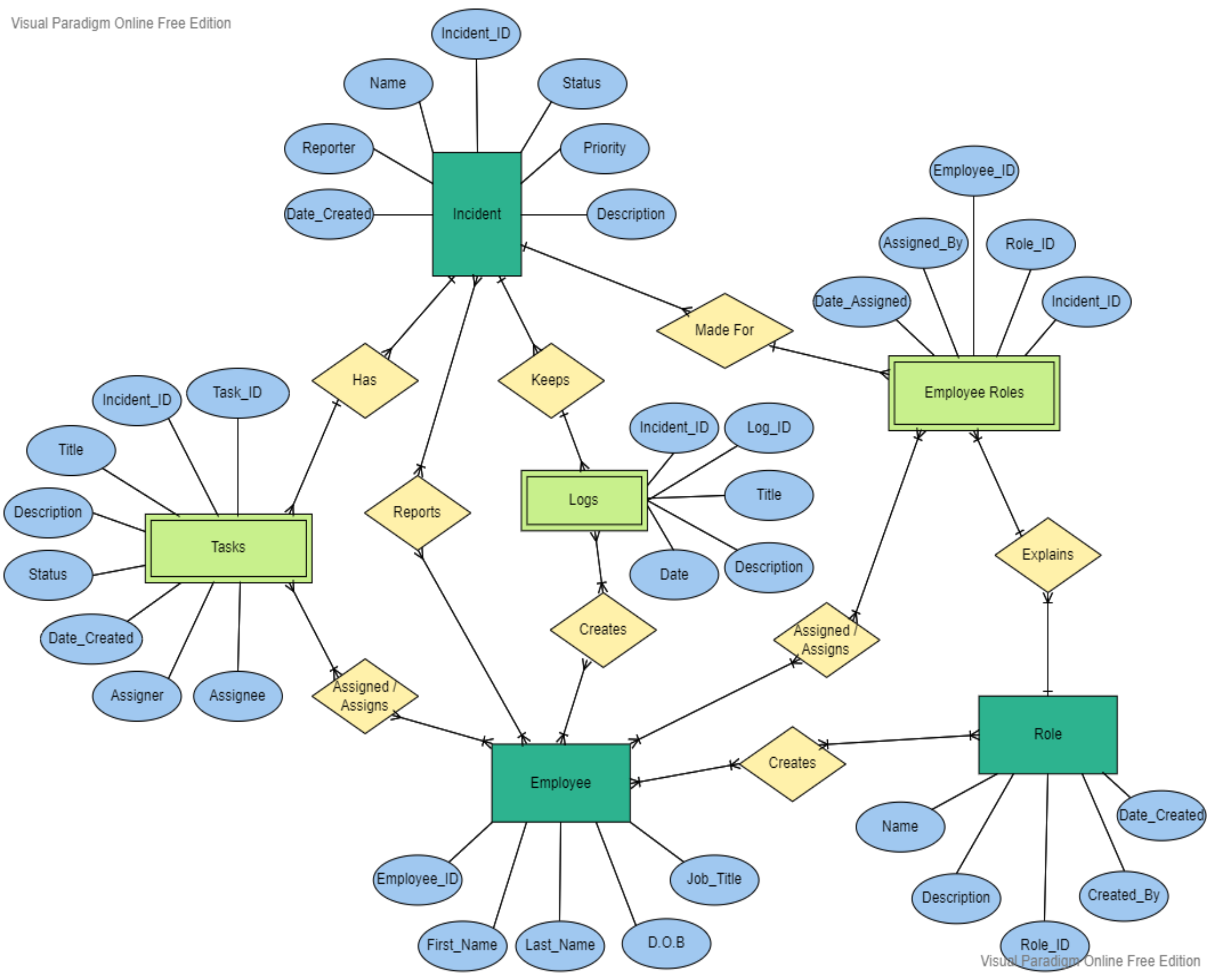
Contents

Section A: Description of Database Application Area and ER Model

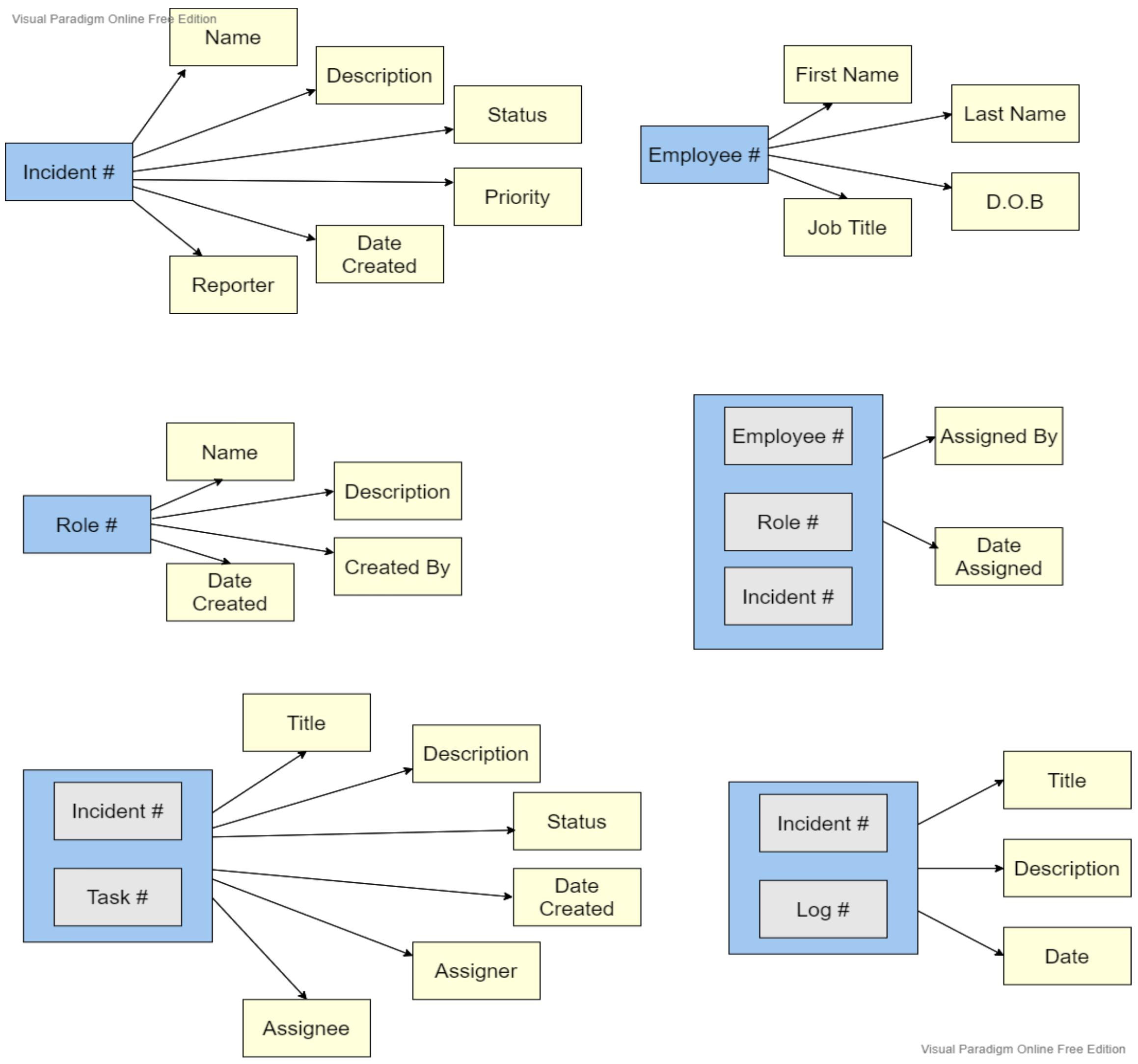
1 Application Description

I decided to implement the database that a potential incident tracking application would use. The application maintains a list of available employees, a list of incidents, and a list of roles. Each incident also has specific tasks and logs, and an employee working on a specific incident has a role assigned to them. Employees are able to “report” an incident, to add a new entry in the Incidents table. Every time an incident is created, it will begin publishing logs to the Logs table. Every time a task is assigned, or an employee is assigned/unassigned a role, a log is created, for the relevant incident.

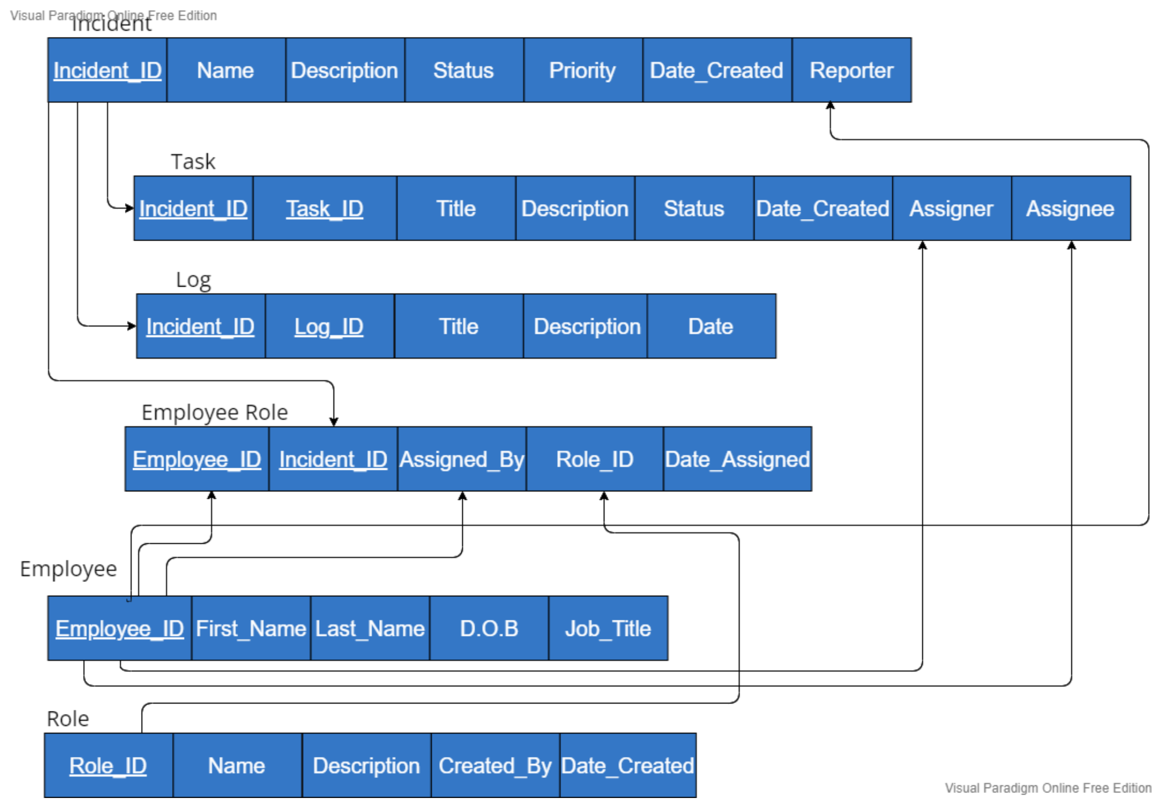
2 Entity Relationship Diagram



4 Functional Dependency Diagrams



3 Mapping to Relational Schema



Section B: Explanation of Data and SQL Code:

5 Explanation of the SQL script for creating the “Incidents” table

```
CREATE TABLE `incidents` (
  `id` int NOT NULL AUTO_INCREMENT,
  `name` varchar(45) NOT NULL,
  `description` varchar(200) NOT NULL,
  `prio` varchar(20) NOT NULL,
  `status` varchar(45) NOT NULL,
  `reporter` int NOT NULL,
  `date_created` datetime NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `id_UNIQUE` (`id`)
);
```

The code above was used to create the “Incidents” table - there is nothing too fancy going on, just declaring the columns and their types, and assigning the id as a primary, unique key.

6 Explanation and SQL Code for any Trigger operations

The following is a trigger that happens before inserting into the “Incidents” table:

```
CREATE DEFINER=`root`@`localhost` TRIGGER `incidents_BEFORE_INSERT`
BEFORE INSERT ON `incidents` FOR EACH ROW BEGIN
      call addLog(new.id, concat('Incident Reported: ',new.name),'The
incident was reported');
END
```

Before the reported incident is added to the “Incidents” table, we just also add a log to the “Logs” table, to start the log history of the newly created incident. When an incident is marked as resolved, a log is also created.

Similarly, when an employee is assigned a role (an entry into the “Employee_Roles” table), or unassigned a role (a deletion from the “Employee_Roles” table), a log is created.
A log is also created when a task is assigned to an employee, for a certain incident.

These triggers are so that each incident can keep a log of what steps are taken to help resolve the incident.

7 Explanation and SQL Code for any Views

The following is a view that shows which employees are part of which incident team.

```
CREATE
    ALGORITHM = UNDEFINED
    DEFINER = `root`@`localhost`
    SQL SECURITY DEFINER
VIEW `incidentteams` AS
    SELECT
        `i`.`id` AS `id`,
        `i`.`name` AS `name`,
        `er`.`employee_id` AS `employee_id`,
        `e`.`first_name` AS `first_name`,
        `e`.`last_name` AS `last_name`
    FROM
        ((`incidents` `i`
        JOIN `employee_roles` `er`)
        JOIN `employees` `e`)
    WHERE
        ((`i`.`id` = `er`.`incident_id`)
            AND (`er`.`employee_id` = `e`.`id`))
```

By joining “Incidents” , “Employee Roles” and “Employees” into one table, we are able to see which employees are working on which incidents.

8 Explanation and SQL Code for populating the “Incidents” table

Below is the code that populates the “Incidents” table:

```
call db.reportIncident(111, 'Dashboard Broken', 'The dashboard is not
loading', 'Major', 192);
call db.reportIncident(292, 'Server Not Responding', 'The website server is
unresponsive', 'Severe', 843);
call db.reportIncident(384, 'Broken Animation', 'The loading bar on the
homepage does not work', 'minor', 56);
call db.reportIncident(495, 'Slow Rendering', 'The website is rendering
slowly', 'Major', 0);
call db.reportIncident(684, 'Login broken', 'The users of the website cannot
log in', 'major', 2);
```

The procedure *reportIncident(id, title, description, priority, reporter)* takes in a number of parameters, which it then uses to insert the given data into the “Incidents” table.

```
CREATE DEFINER=`root`@`localhost` PROCEDURE `reportIncident`(
    in id int,
    in name varchar(45),
    in description varchar(200),
    in priority varchar(20),
    in reporter int


)
BEGIN
    INSERT INTO `db`.`incidents` (`id`, `name`, `description`, `prio`,
`status`,   `reporter`,`date_created`)
    VALUES (id,name,description,priority,'In Progress',reporter, now());
END
```