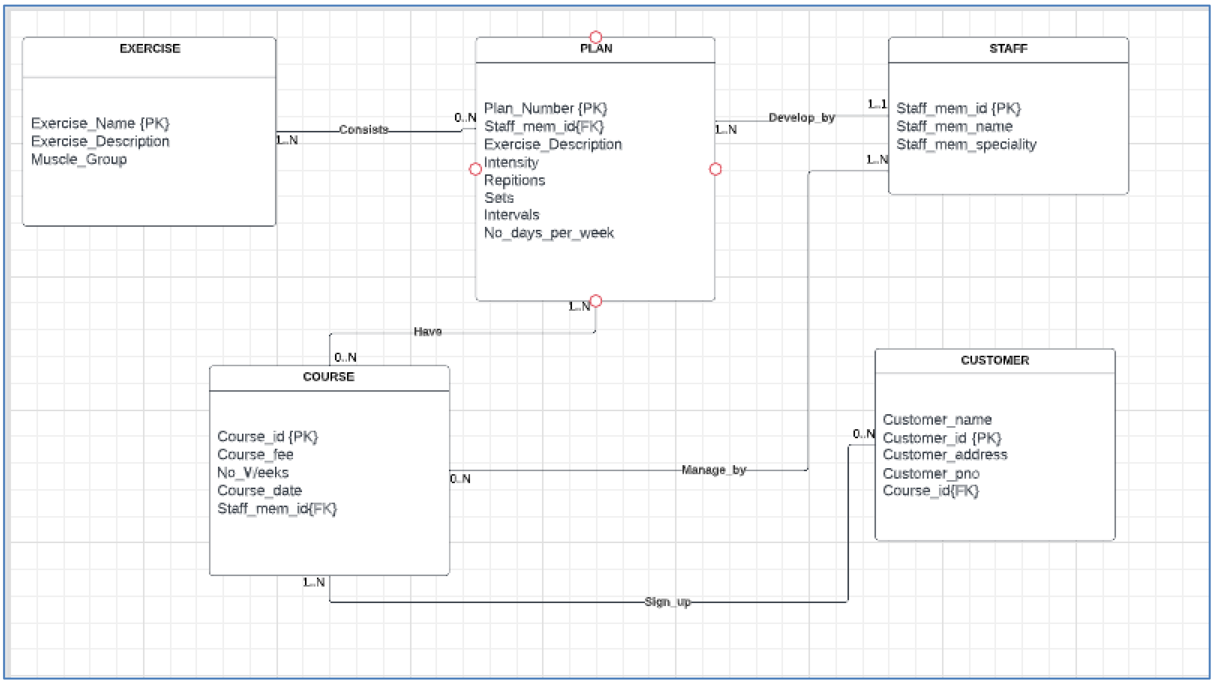
ISYS1055/1057/3412 (Practical) Database Concepts

Assessment 1 Draft : Database Design

Name:Pratham Radhakrishna

Part A: Entity-Relationship Modelling

Task 1: Designing an Entity-Relationship Model



Mapping ER Model

Step 1: Map Strong Entities

Exercise(<u>Exercise_Name</u>, Exercise_Description,Muscle_Group)

Plan(<u>Plan_Number</u>,Staff_mem_id*,Exercise_description,Intensity,Repitions,Sets,Intervals,No_days_per_week)

Staff(<u>Staff_mem_id</u>,Staff_mem_name,Staff_mem_speciality)

Course(<u>Course_id</u>,Course_Fee,No_week,Course_date, Staff_mem_id*)

Customer(<u>Customer_id</u>,Customer_name,Customer_address,Customer_pno, Course_id*)

Step 2: Map Weak Entities

Null

Step 3: Map 1:1 Relationships

- Mandatory participation on both sides
- Mandatory participation on one side
- Optional participation on both sides

Mandatory Participation on both sides
NULL
Mandatory Participation on one side
Null
Optional Participation on both sides
Null

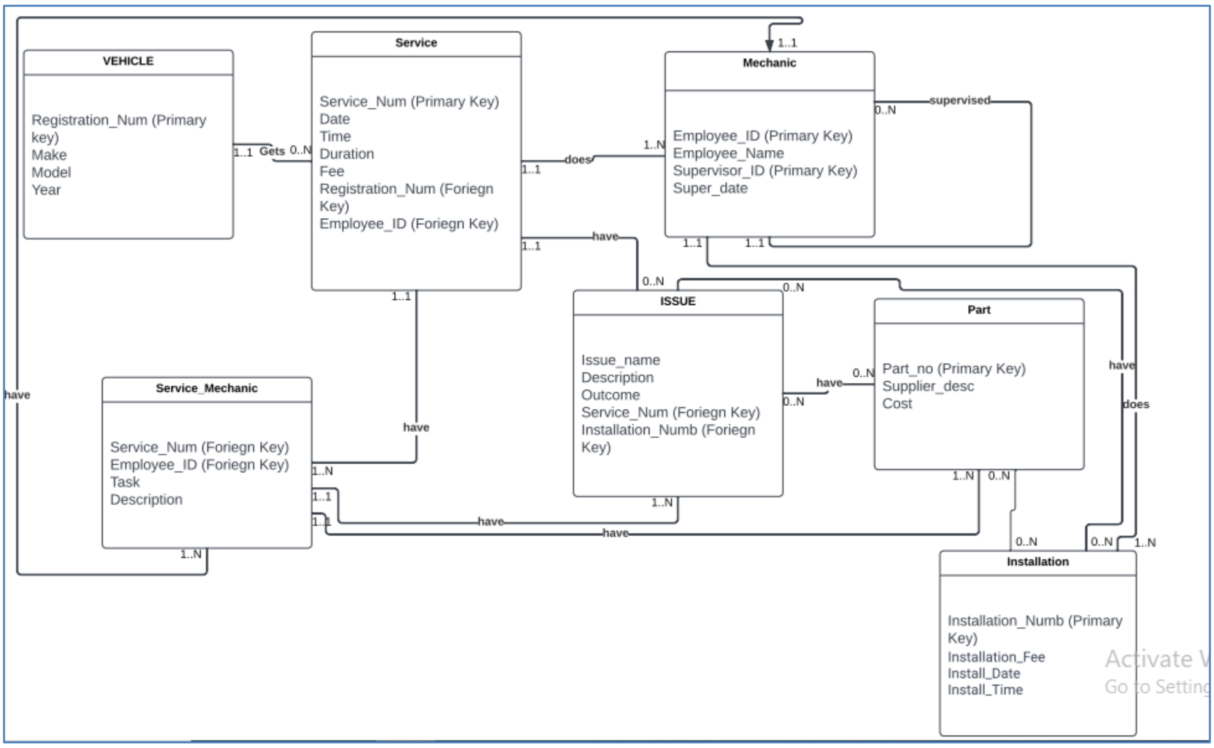
Step 4: Map 1:N Relationships
Plan(Plan_Number, Exercise_Description,Intensity,Repetitions,Sets,Intervals,No_days_per_week, Staff_mem_id*)

Step 5: Map M:N Relationships
Consists(Exercise_Name*,Plan_Number*)
Deveop_by(Plan_Number*,Staff_mem_id*)
Have(Course_id*,Plan_Number*)
Manage_by(Course_id*,Staff_mem_id*)
Sign_up(Course_id*,Customer_id*)

Step 6: Multi-valued Attributes
Null
Step 7: Map higher-degree relationships
Null

Complete Relational Model
Consists(Exercise_Name*,Plan_Number*)
Deveop_by(Plan_Number*,Staff_mem_id*)
Have(Course_id*,Plan_Number*)
Manage_by(Course_id*,Staff_mem_id*)
Sign_up(Course_id*,Customer_id*)
Plan(Plan_Number, Exercise_Description,Intensity,Repetitions,Sets,Intervals,No_days_per_week, Staff_mem_id*)
Exercise(Exercise_Name, Exercise_Description,Muscle_Group)
Staff(Staff_mem_id,Staff_mem_name,Staff_mem_speciality)
Course(Course_id,Course_Fee,No_week,Course_date, Staff_mem_id*)
Customer(Customer_id,Customer_name,Customer_address,Customer_pno, Course_id*)

Task 2: Designing an Entity-Relationship Model



Mapping ER Model

Step 1: Map Strong Entities

Vehicle(<u>Registration_Num,</u>Make,Model,Year)

Service(<u>Service_Num</u>,Date,Time,Duration,Fee,Registration_Num*,Employee_ID*)

Mechanic(<u>Employee_ID</u>,Employee_Name,<u>Supervisor_ID</u>,Super_date)

Part(<u>Part_no</u>,Supplier_desc,Cost)

Installation(Installation_Numb,Installation_Fee,Install_Date,Install_Time)

Step 2: Map Weak Entities

Service_Mechanic(Service_Num*,Employee_ID*,Task,Description)

Issue(Service_Num*,Installation_Numb*,Issue_name,Description,Outcome)

Step 3: Map 1:1 Relationships

- Mandatory participation on both sides
- Mandatory participation on one side
- Optional participation on both sides

Mandatory participation on both sides
Null

Mandatory participation on one side
Null
Optional participation on both sides
Null

Step 4: Map 1:N Relationships

Service(Service_Num,Date,Time,Duration,Fee,Registration_Num*,Employee_ID*)

Mechanic(Employee_ID,Employee_Name,Supervisor_ID,Super_date,Service_Num*)

Issue(Service_Num*,Installation_Numb*,Issue_name,Description,Outcome)

Service_Mechanic(Service_Num*,Employee_ID*,Task,Description,Supervisor_ID*)

Part(Part_no,Supplier_desc,Cost)

Installation(Installation_Numb,Installation_Fee,Install_Date,Install_Time,Employee_ID*,Supervisor_ID*)

Step 5: Map M:N Relationships

Contains(Part_no*)
Consists(Installation_Numb*)
Can_have(Part_no*,Installation_Numb*)

Step 6: Multi-valued Attributes
Null
Step 7: Map higher-degree relationships
Null
Complete Relational Model
Contains(Part_no*)
Consists(Installation_Numb*)
Can_have(Part_no*,Installation_Numb*)
Service(Service_Num,Date,Time,Duration,Fee,Registration_Num*,Employee_ID*)
Mechanic(Employee_ID,Employee_Name,Supervisor_ID,Super_date,Service_Num*)
Issue(Service_Num*,Installation_Numb*,Issue_name,Description,Outcome)
Service_Mechanic(Service_Num*,Employee_ID*,Task,Description,Supervisor_ID*)
Part(Part_no,Supplier_desc,Cost)
Installation(Installation_Numb,Installation_Fee,Install_Date,Install_Time,Employee_ID*,Supervisor_ID*)
Vehicle(Registration_Num,Make,Model,Year)

3.

Step 1: Map Strong entitites

ProjectManager(<u>employeeID</u>,name)

Client(<u>ID</u>, name)

Contract(<u>contractNo</u>, startDate, endDate, totalFee)

BankAccount(<u>BSB, accountNo</u>)

ConstructionProject(<u>address</u>, roomCount, noOfFloors)

Contractor(<u>ABN</u>, phoneNo)

Step 2: Map Weak Entities

SubContract(contractNo*, <u>task</u>, fee, balance)

Step 3: Map 1:1 Relationships

- Mandatory participation on both sides
- Mandatory participation on one side
- Optional participation on both sides

1:1 Relationship with Mandatory Participation on both sides

Contract(<u>contactNo</u>, startDate, endDate, totalFee, address*, roomCount, noOfFloors)

Client(<u>ID</u>, name, BSB*, accountNo*)

Mandatory participation on one side

Null

Optional participation on both sides

Null

Step 4: Map 1:N Relationship

Contract(<u>contractNo</u>, startDate, endDate, totalFee, address*, roomCount, noOfFloors, ID*, signDate, EmployeeID*)

SubContract(contractNo*, <u>task</u>, fee, balance, ABN*)

Step 5: Map N:N Relationship

WorksOn(<u>contractNo*, task*, ABN*</u>)

Step 6: Map Multi - Value Entity

Null

Step 7: Map higher-degree relationships

Null

Final Schema

WorksOn(contractNo*, task*, ABN*)

SubContract(contractNo*, <u>task</u>, fee, balance, ABN*)

Contract(<u>contractNo</u>, startDate, endDate, totalFee, address*, roomCount, noOfFloors, ID*, signDate, EmployeeID*)

Client(<u>ID</u>, name, BSB*, accountNo*)

Contractor(<u>ABN</u>, phoneNo)

ProjectManager(<u>employeeID</u>, name)

Part B

Task 4: Relational Database Model

4.1

Yes, there is a manager associated with each department because Departments table has a foreign key "manager_id" that has taken reference from Employees table "employee_id" . So each department have a employee as its manager. And "manager_id" in the Departments table shows that it is a primary key in the Employees table, so every manager is an existing employee.

4.2

Employees.department_id ------>Departments.department_id

Employees.empjob_id ------> Jobs.job_id

Employees(<u>employee_id,</u> first_name, last_name, phone_number, hire_date, empjob_id*, salary, department_id*)

Departments(<u>department_id,</u> department_name, manager_id*, location_id*)

Jobs(<u>job_id,</u> job_title, min_salary, max_salary,)

Locations(<u>location_id,</u> street_address, postal_code, city, state_province, country_id*)

Countries(<u>country_id</u>, country_name)

JobHistory(<u>employee_id*, start_date, end_date,</u> job_id*, department_id*)

4.3

- UPDATE Departments SET department_name='IT Support' WHERE department_id=1;

The above query is sufficient to achieve the requirements specified above.

- INSERT INTO Departments VALUES(4, 'Software Development', 12, 10);

The second statement given produces conflicts in the given conditions .
The software development sub department has been given location id as 10 and does not specify any new location id.
Secondly there is no new job title created as director in the jobs table where salary range is between 130,000 to 160,000. These two have to be updates in order to satisfy the given condition.

4.4

UPDATE Employees SET empjob_id=45, hire_date=6/06/2020 WHERE empjob_id=33;

The above query will show error because the hire_date attribute format which is of type Date is wrong . It should be 'yyyy-mm-dd' . After correcting the above error , it is possible to find all past contracts of Jonny dean . We can see that employee ID of Jonny deans is 10 . We can see from the Job History table in the Employee ID that Jonny deans has 2 past contracts. You can also use the query to retrieve the past contract:
SELECT* from JobHistory WHERE employee_id=10 ;

4.5

DELETE FROM LOCATIONS WHERE location_id=10;
The above query will delete all the rows having location id 10 in the locations table .
To successfully run this statement we have to see if location_id which is a primary key has been referenced in any other table.
We see that departments table has a foreign key location_id .So any row in this table having location_id 10 has to be deleted or updated.

4.6

CREATE TABLE JobHistory
(
emp_id INTEGER, start_date DATE, end_date DATE, job_id INTEGER, depart_id INTEGER, PRIMARY KEY (emp_id , start_date , end_date),FOREIGN KEY (emp_id) REFERENCES Employees(employee_id), FOREIGN KEY (job_id) REFERENCES Jobs(job_id), FOREIGN KEY (depart_id) REFERENCES Departments(department_id));

This creates a table called JobHistory with five columns emp_id,start_date,end_date,job_id and depart_id where emp_id,job_id and depart_id are foriegn keys referencing the primary keys of the corresponding tables .

4.7

CREATE TABLE Jobs (job_id INTEGER PRIMARY KEY, job_title TEXT NOT NULL, min_salary INTEGER NOT NULL, max_salary INTEGER NOT NULL, CHECK (min_salary <= max_salary));

This creates table called Jobs with four columns job_id,job_title,min_salary,max_salary. The CHECK constraint makes sure that the minimum salary is not greater than the maximum salary. The data types are assumed to be INTEGER for the job_id, min_salary, and max_salary columns, and TEXT for the job_title column.

4.8

UPDATE Employees SET salary = 90000, hire_date = '01/01/2021' WHERE employee_id = 50;

This updates the salary and hiring date of the employee (Adam Smith) having employee id 50 in the Employees table