

## DATABASE DESCRIPTION

This database represents the database of a gym chain. It has mainly six tables and two other tables were added to it in order to normalize the data. So, finally there is a total of eight tables namely:

branch, supplier, equipment, customer, class, employee, customerclass, daysinaweak. These are the main requirements any gym chain would require.

The table branch has the attributes: branch\_id(primary key), street, city, country, phone\_number.

The table supplier has the attributes: supplier\_id(primary key), first\_name, last\_name, phone\_number, branch\_id(foreign key-provides the link between branch and supplier table).

The table equipment has the attributes: equipment\_id(primary key), cost, name, supplier\_id(foreign key-provides the link between equipment and supplier table).

The table customer has the attributes: customer\_id(primary key), age, email, first\_name, last\_name, phone\_number, branch\_id(foreign key- links customer and the branch).

The table customerclass has the attributes: customer\_id(primary key), class\_type(primary key).

The table class has the attributes: class\_type(primary key), duration, employee\_id(foreign key- links the tables class and employee).

The table daysinaweak has the attributes: daysinaweak(primary key), class\_type(primary key).

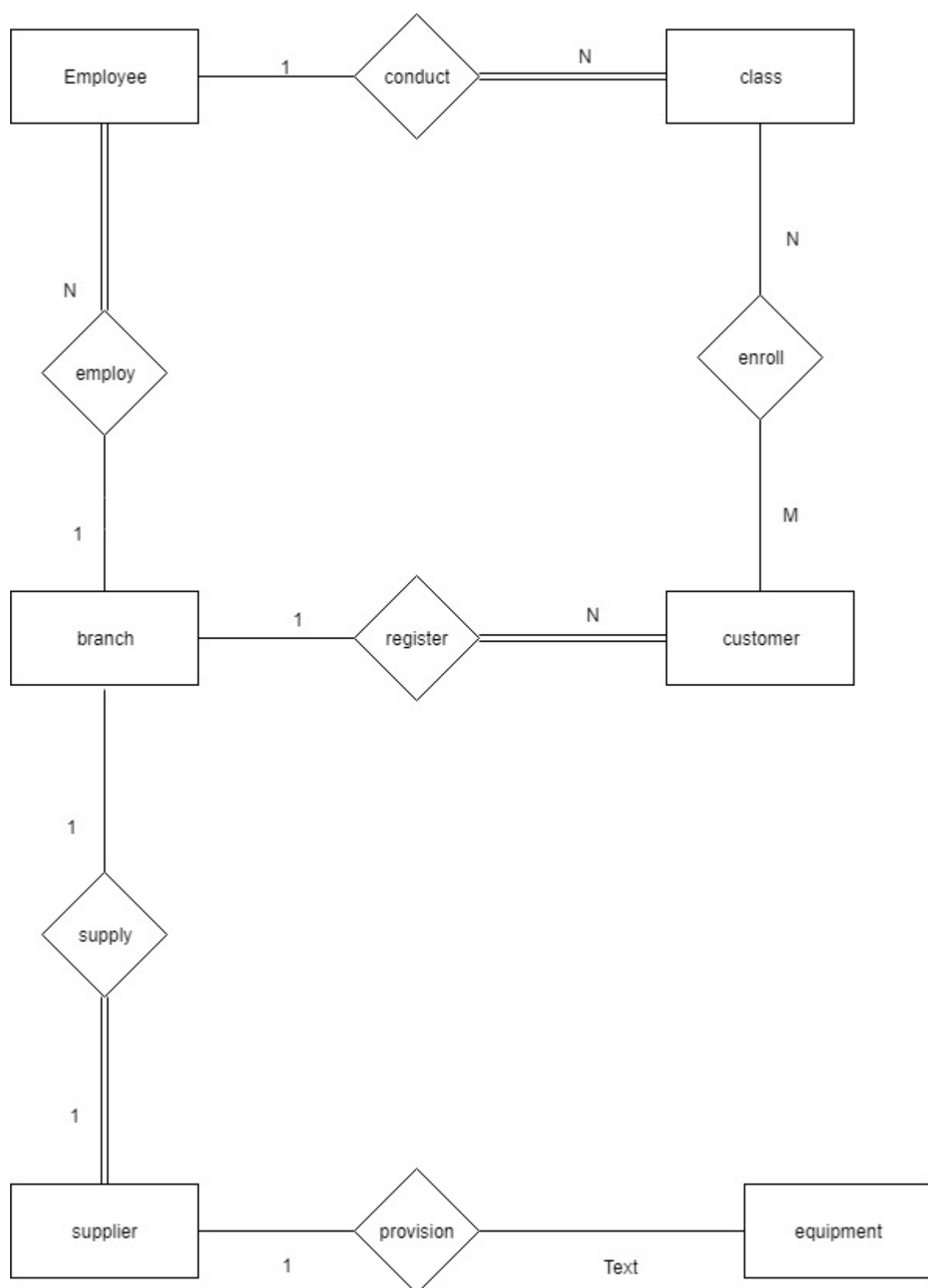
The table employee has the attributes: employee\_id(primary key), first\_name, last\_name, DOJ-month, DOJ-day, DOJ-year, salary, type, brach\_id(foreign key-links employee and the branch table).

Commands such as create, alter, insert into and a few check constraints were applied to create and modify the table data in the database. In order to make the database simple to understand minimal number of attributes were used. The attributes particular to each entity have been laid out in the entity-relation diagram along with the particular relations that connect the tables together with the type of relation being declared as either one-to-one, one-to-many, many-to-many. The participation of these relations have also been specified as either total or partial participation. How the tables are functionally dependent on each other have also been clearly shown with the help of functional dependency diagram.

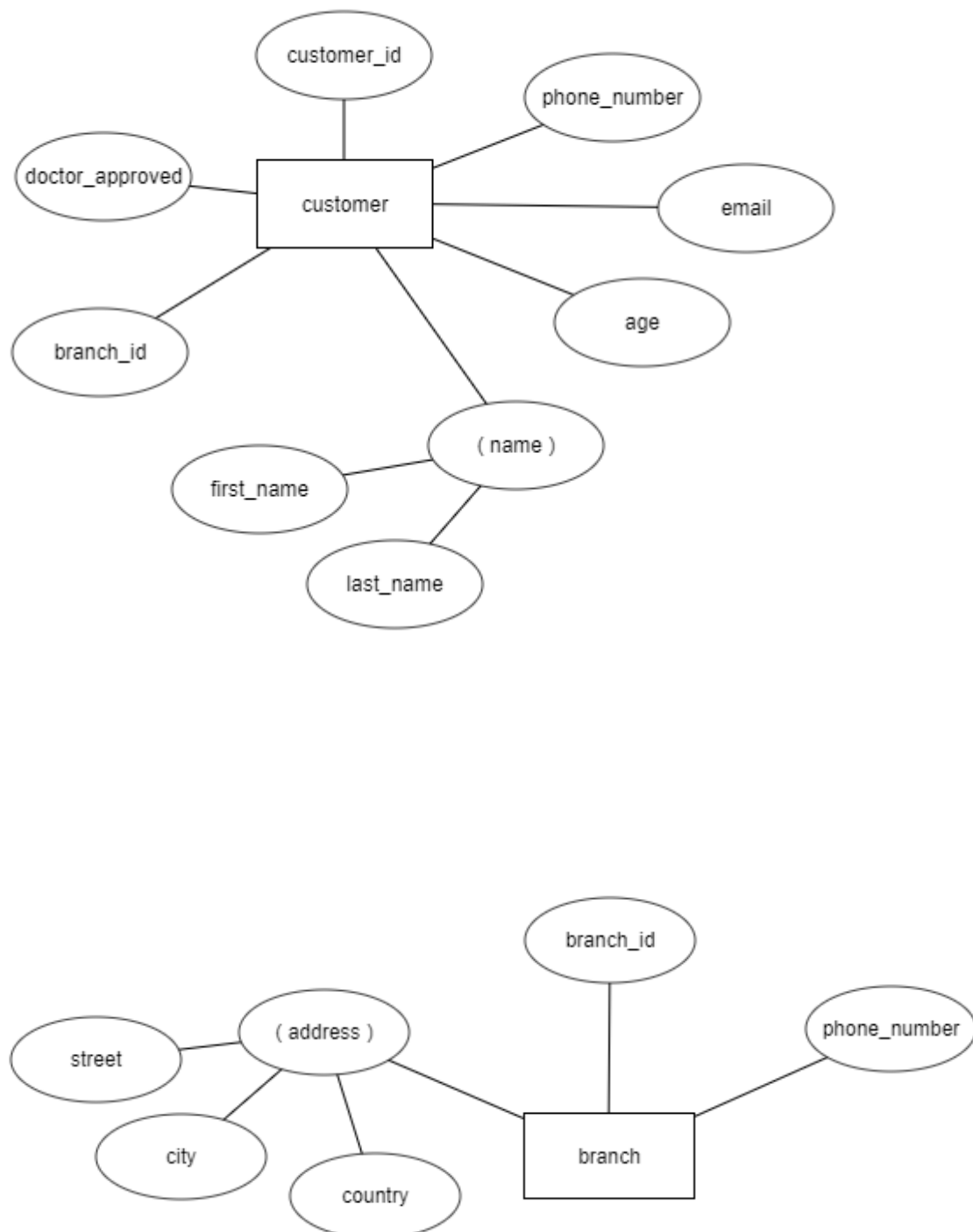
Each table has been identified by a unique primary key and in some tables like the customerclass table and daysinaweak table have a composite primary key. Initially the tables did not have any link between them but in-order to create a functional database foreign keys were introduced in the tables because of which all the tables were linked together to form a database. Semantic constraints such as NOT NULL constraints were added. In order to avoid deliberate corruption of database, database security was introduced with the help of security policy and access control. In the database security roles were created and permissions were granted and revoked. Views were created to gain partial access to the information contained in the relations. Relational select and join operations were performed to get information from particular attributes of either single or multiple tables. Update operations were performed of the data to change the values that were initially assigned.

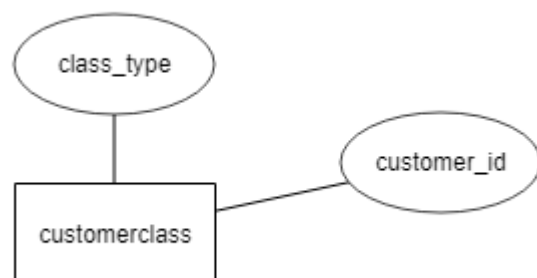
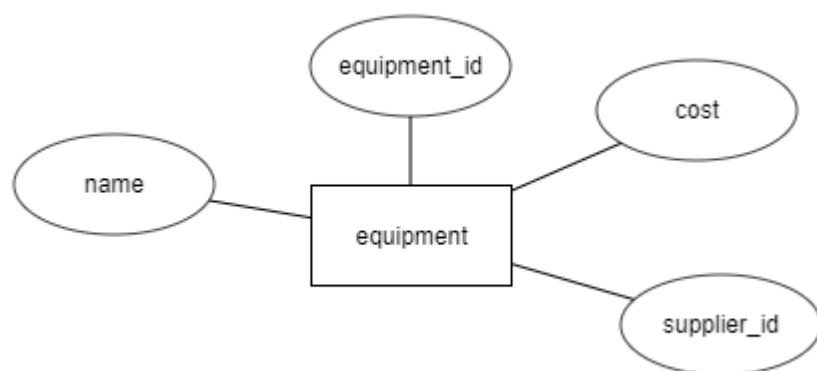
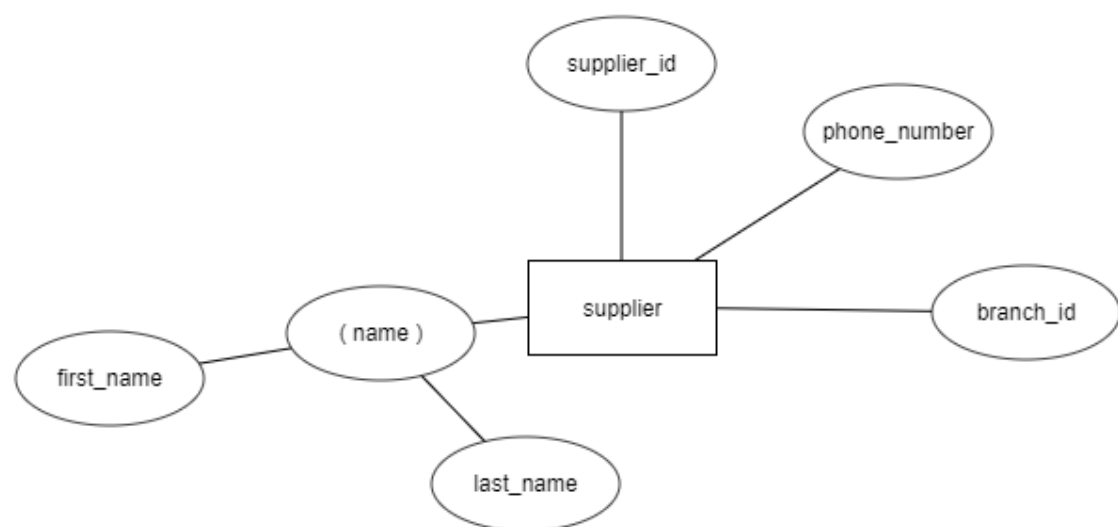
Triggers were also created in the database so as to allow certain constraints to be checked on occurrence of specific events and resulting actions to be invoked.

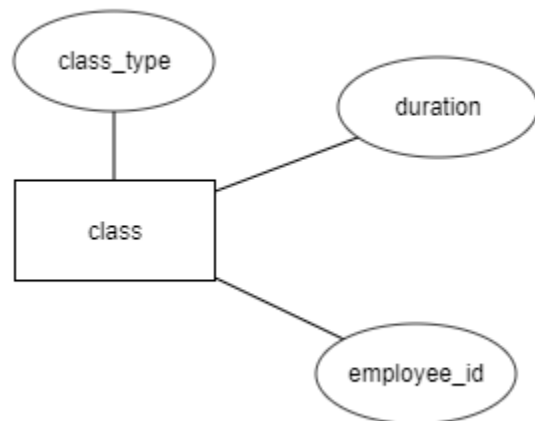
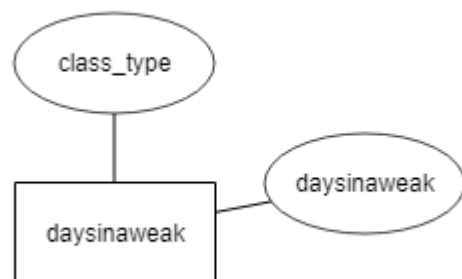
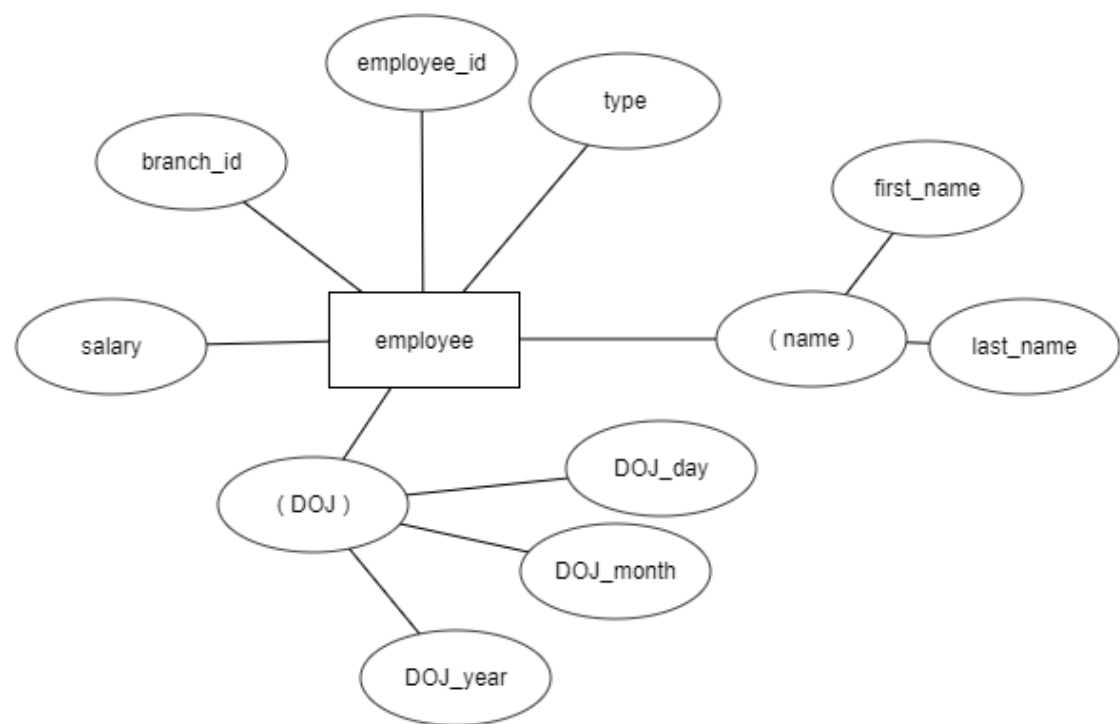
## Entity Relationship diagram



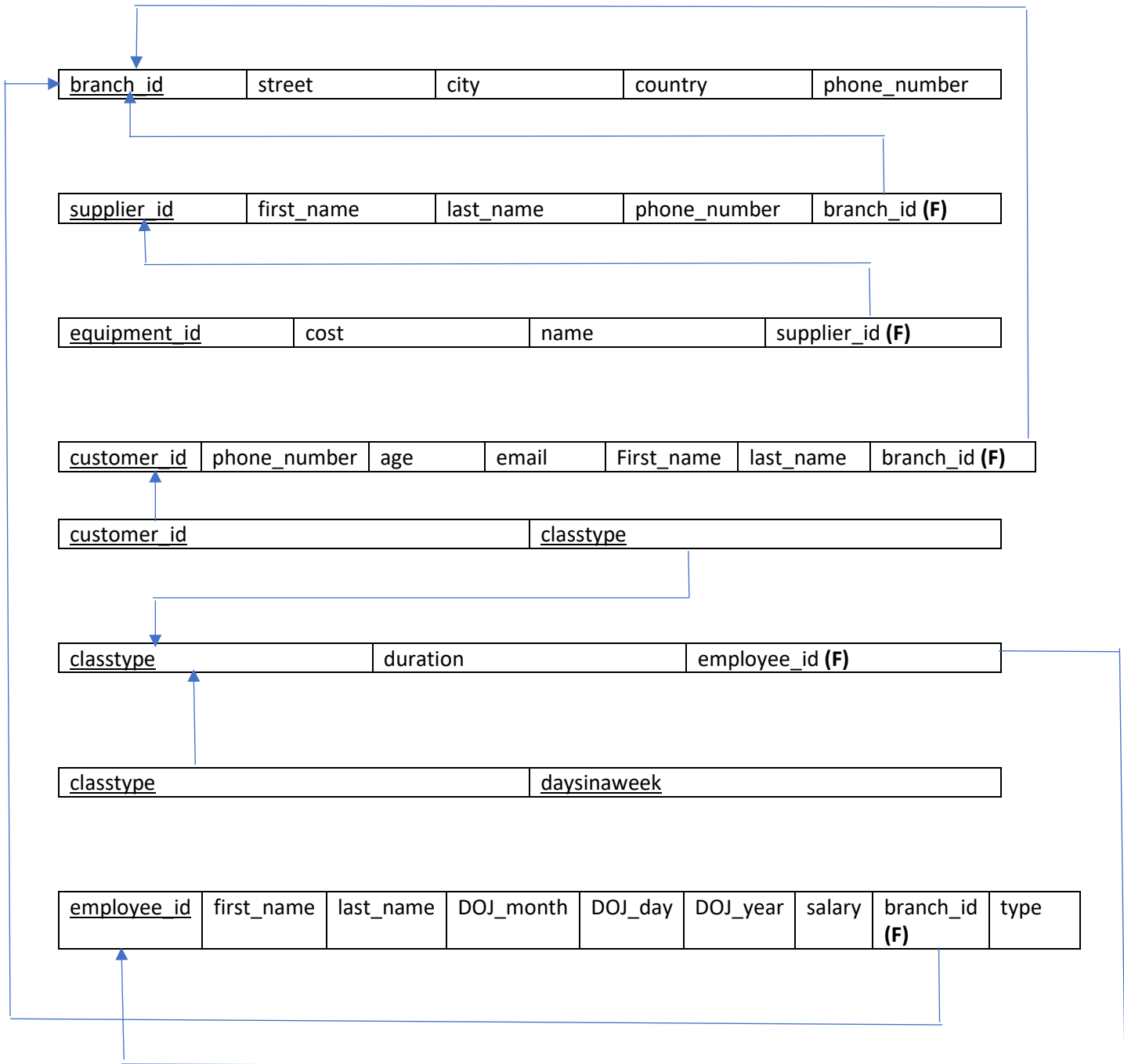
## Entities along with attributes








## Mapping to Relational Schema




## Functional Dependency Diagram

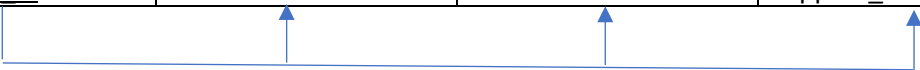
<u>branch_id</u>	street	city	country	phone_number
------------------	--------	------	---------	--------------




<u>supplier_id</u>	first_name	last_name	phone_number	branch_id
--------------------	------------	-----------	--------------	-----------



<u>equipment_id</u>	cost	name	supplier_id
---------------------	------	------	-------------



<u>customer_id</u>	phone_number	age	email	First_name	last_name	branch_id
--------------------	--------------	-----	-------	------------	-----------	-----------



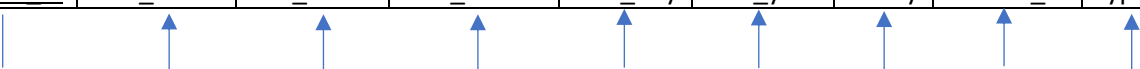
<u>customer_id</u>	<u>classtype</u>
--------------------	------------------

<u>classtype</u>	duration	employee_id
------------------	----------	-------------



<u>classtype</u>	<u>daysinaweek</u>
------------------	--------------------

<u>employee_id</u>	first_name	last_name	DOJ_month	DOJ_day	DOJ_year	salary	branch_id	type
--------------------	------------	-----------	-----------	---------	----------	--------	-----------	------



## Semantic constraints:

There are a number of semantic constraints used in this database. Each table has been assigned a **PRIMARY KEY** and a **FOREIGN KEY** as well as some columns have been assigned the constraint of **NOT NULL**. These commands can be found in the appendix section.

```
ALTER TABLE `gym`.`supplier`  
ADD CONSTRAINT `branch_id`  
FOREIGN KEY (`branch_id`)  
REFERENCES `gym`.`branch` (`branch_id`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

The other semantic constraints that have been applied are in the alter table commands as follows:

Here, in the customers table a check constraint has been added in the alter table statement where in the age of the customer is checked to be greater than 18.

```
ALTER TABLE gym.customer  
ADD CHECK (Age>=18);
```

Another semantic constraint has been added in the customers table within the alter table command where the values of the column doctor\_approved are checked to belong only to 'y' or 'n'.

```
ALTER TABLE gym.customer  
ADD CONSTRAINT check_doctor_approved CHECK (doctor_approved IN ('y', 'n'));
```

## Normalization:

The tables used were mostly already normalized. It's just that two new tables were added in the database named customer-class and days-in-a-week. As the numbers of days a class can be conducted is a multivalued attribute according to my assumption. There another table was created named days-in-a-week where in the primary key of the table class and the attribute days-in-a-week were considered to the entities, and both these entities were taken to be the primary keys of the new table. Another table named customer-class was created because according to my consideration the relation between tables class and customer is an N:M relation.



## Database Security:

Database Security is one of the most important and difficult task to be carried out while creating a database. As database security is basically concerned with DELIBERATE corruption of the database.

Database security can be implement using either security policies or access control. In this database the various access levels have been described as well as a brief description of the Security policy used is provided. In this database the DBMS can restrict access to the database by setting Privileges and User Accounts this is very important in the creation of the database in-order to ensure that no unauthorised person can access the database to obtain information or to make malicious changes within the database.

Privileges can be classified into two main categories:

- Account Level:  
A Database Administrator (DBA) holds responsibility for creating user accounts and for granting and revoking privileges. As owner of all relations in the database the DBA automatically has all relation privileges granted to him and, using the GRANT command, can specify user privileges for each relation. The DBA will create user accounts for all Trainers, receptionists and managers. Privileges and other permissions will be granted based on organisational roles. Account level permissions such as the ability to create, alter or drop schema, tables and views will not be granted to employees. Only the DBA has the authority to perform these operations and should any other employee wish for such an operation to be performed they must contact the DBA and request it.
- Relation Level:  
DBA can control the privilege to access each individual relation or view in the database. These privileges can be specified on entire relations or on specific attributes. There is a Role-Based access provided to access the database.  
Relation Level Privileges can be classified as:
  - Read Privilege- gives an account the ability to use SELECT to retrieve rows from this relation
  - Modification Privileges- gives an account the ability to use INSERT, UPDATE and DELETE to modify rows in this relation
  - Reference Privilege- gives an account the ability to refer to this relation when specifying integrity constraints.

In order, to grant privileges firstly the specific roles must be created . This is done using the **CREATE ROLE** command:

```
CREATE ROLE trainer;
```

```
CREATE ROLE receptionist;
```

```
CREATE ROLE manager;
```

### **Privileges for trainer:**

- The following privilege allows the trainer to select customers in the gym.  
  
GRANT select on gym.customer TO trainer;
- The following privilege allows the trainer to select, insert, delete, update the classes in the schema gym.  
  
GRANT INSERT, SELECT, DELETE, UPDATE ON gym.class TO trainer;
- The following privilege allows the trainer to select, insert, delete, update the attributes of the table customerclasses in the schema gym.  
  
GRANT INSERT, SELECT, DELETE, UPDATE ON gym.customerclass TO trainer;
- The following privilege allows the trainer to select, insert, delete, update the attributes of the table daysinaweak in the schema gym.  
  
GRANT INSERT, SELECT, DELETE, UPDATE ON gym.daysinaweak TO trainer;

### **Privileges for manager:**

- A privilege has been granted to the manager where in the manager can update the salary of the employees using a view named employee\_viewed, wherein only those employees are selected who are trainers.  
  
GRANT UPDATE(salary) ON employee\_viewed TO manager;
- The following privilege allows the manager to update the branch id of the employees in the gym.  
  
GRANT UPDATE(branch\_id) ON gym.employee TO manager;
- The following privilege allows the manager to select the attributes in the table daysinaweak and he also has the power to pass this privilege further to other people.  
  
GRANT SELECT ON gym.daysinaweak TO manager with GRANT OPTION;
- The following privilege allows the manager to perform the operations insert, update, delete, select on the table supplier.  
  
GRANT INSERT, SELECT, DELETE, UPDATE ON gym.supplier TO manager;

### **Privileges for receptionist:**

- The following privilege allows the receptionist to update the phone number and email of the customers enrolled in the gym.  
  
GRANT UPDATE(phone\_number),UPDATE(email) ON gym.customer TO receptionist;

### **Misuse or inappropriate use of privileges:**

If there is a case where a person assigned some role misuses their power then the permissions granted to them for the changes in the database can be revoked using the REVOKE command.

If, for example, a person with the role manager, was found to have updated salary fields without having been instructed to do so, his privileges relating to the employee table will be removed:

```
REVOKE UPDATE(salary) ON gym.employee FROM manager;
```

## **VIEWS**

Views are an important discretionary authorisation mechanism. With the help of views, owners of a relation can grant partial access to the information contained in that relation. A relation basically acts as a new relation in the database. Many views were created in the database in order to allow access to restricted sets of rows and attributes as follows:

This view has been created to restrict access only to the attributes employee\_id, first\_name, last\_name, salary which belong to the table employee and whose type is trainer.

```
CREATE VIEW employee_viewed  
AS  
SELECT employee_id, first_name, last_name, salary  
FROM gym.employee WHERE type = 'trainer';
```

Views can also be created by using multiple tables as in the following example here a view named customer\_trainer is created which allows to restrict access only to the attributes first\_name, last\_name of the table customer and employee\_id of the table employee where the branch\_id of the customer table matches the branch\_id of the employee table.

```
CREATE VIEW customer_trainer  
AS  
SELECT customer.first_name, customer.last_name, employee.employee_id  
FROM customer, employee  
WHERE customer.branch_id = employee.branch_id;
```

## Relational Select and Table Join Operations

- `SELECT * FROM branch;`

branch_id	street	city	country	phone_number
101	dall	dublin	ireland	12345678
102	abbey	delhi	india	23456789
103	kiney	bombay	india	34567890
104	napey	london	england	45678901
105	karant	chelster	italy	56789012

- `SELECT customer_id, class_type FROM customerclass;`

customer_id	class_type
401	abs
402	yoga
403	yoga
404	spinning
405	spinning

- `SELECT customerclass.class_type, customer.first_name`  
`FROM customerclass, customer`  
`WHERE customerclass.customer_id=customer.customer_id;`

class_type	first_name
abs	kina
yoga	robin
yoga	salman
spinning	kriti
spinning	kavya

- `SELECT equipment.cost, supplier.first_name, branch.branch_id, branch.street`  
`FROM equipment, supplier, branch`  
`WHERE equipment.supplier_id=supplier.supplier_id and`  
`supplier.branch_id=branch.branch_id;`

cost	first_name	branch_id	street
6000	ken	101	dall
4000	henry	102	abbey
5000	ruby	103	kiney
7000	paul	104	napey
8000	karina	105	karant

## Update Operations:

Many update operations were performed in the database a few of them are listed below:

- UPDATE employee  
SET salary = 9000  
WHERE employee\_id = '501';
- UPDATE `gym`.`class` SET `employee\_id` = '501' WHERE (`class\_type` = 'abs');
- UPDATE `gym`.`class` SET `employee\_id` = '502' WHERE (`class\_type` = 'cross-fit');
- UPDATE `gym`.`class` SET `employee\_id` = '501' WHERE (`class\_type` = 'full-body');
- UPDATE `gym`.`class` SET `employee\_id` = '503' WHERE (`class\_type` = 'spinning');
- UPDATE `gym`.`class` SET `employee\_id` = '505' WHERE (`class\_type` = 'yoga');
- UPDATE `gym`.`branch`  
SET `phone\_number` = '12323467'  
WHERE (`branch\_id` = '101');
- UPDATE `gym`.`supplier` SET `last\_name` = 'maan' WHERE (`supplier\_id` = '201');

## TRIGGERS IN SQL

Several triggers were created in this database. These triggers came into action when a particular event occurred. The following are the triggers used:

This trigger implementation shows that a customer back up is created such that when we delete a customer we can still view the deleted customer in the back up table named customerbackup.

For this customerbackup was a new table that was created. The implementation is shown as follows.

```
DELIMITER $$
```

```
CREATE TRIGGER cust_back_up
```

```
AFTER DELETE ON customer
```

```
FOR EACH ROW
```

```
BEGIN
```

```
    Insert INTO customerbackup(customer_id,phone_number)  
    values(OLD.customer_id,OLD.phone_number);
```

```
END$$;
```

DELIMITER ;

DELETE FROM `gym`.`customer` WHERE (`customer\_id` = '405');

```
mysql> select * from customerbackup;
+-----+-----+
| customer_id | phone_number |
+-----+-----+
| 405         | 98076543     |
+-----+-----+
1 row in set (0.00 sec)

mysql> select * from customer;
+-----+-----+-----+-----+-----+-----+-----+-----+
| customer_id | phone_number | email          | age | first_name | last_name | branch_id | doctor_approved |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 401         | 12343215     | kina@gmail.com | 20  | kina       | gart      | 101       | y               |
| 402         | 66998877     | robin@hotmail.com | 25  | robin      | penny     | 102       | n               |
| 403         | 45678902     | salman@yahoo.com | 45  | salman     | NULL      | 103       | y               |
| 404         | 56432178     | kriti@gmail.com | 18  | kriti      | len       | 104       | y               |
+-----+-----+-----+-----+-----+-----+-----+-----+
4 rows in set (0.00 sec)
```

The following trigger implementation basically pops up a message when the age of the customer is less than 18.

DELIMITER \$\$

CREATE PROCEDURE chk\_age(IN age INT)

BEGIN

IF age <= 18 THEN

SET MESSAGE\_TEXT = 'Less Than 18!!'

END IF;

END;&&

DELIMITER ;

The following trigger implementation shows that when a new insert statement is created in the table class then the value of the column corresponding to that class type in the table daysinaweak is set to one.

Delimiter \$\$

CREATE TRIGGER update\_date

AFTER INSERT ON class

for each row

begin

Insert into daysinaweak values(new.class\_type, 1);

END;&&

DELIMITER ;

## ADDITIONAL FEATURES OF SQL

PL/SQL variables are used to store specific information in the database. The implementation is shown below. In the following code two new variables are declared and the new id variable is initialised with a particular employee id. Further a select operation is performed on the table employee which assigns the salary of the employee to var\_newsalary with the given employee id.

DECLARE

var\_newsalary number(40);

var\_employeeid number(40) := 503;

BEGIN

SELECT salary

INTO var\_newsalary

FROM gym.employee

WHERE employee\_id = var\_employeeid;

dbms\_output.put\_line(var\_newsalary);

dbms\_output.put\_line('Employee with employee\_ID' || var\_employeeid || ' has a salary of '  
|| var\_newsalary);

END;

.

run;

## Appendix

```
CREATE SCHEMA `gym` ;
```

### **Gym\_branch**

```
CREATE TABLE `gym`.`branch` (  
  `branch_id` INT NOT NULL,  
  `street` VARCHAR(45) NOT NULL,  
  `city` VARCHAR(45) NOT NULL,  
  `country` VARCHAR(45) NOT NULL,  
  `phone_number` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`branch_id`));
```

```
INSERT INTO `gym`.`branch` (`branch_id`, `street`, `city`, `country`, `phone_number`) VALUES ('101',  
'dall', 'dublin', 'ireland', '12345678');
```

```
INSERT INTO `gym`.`branch` (`branch_id`, `street`, `city`, `country`, `phone_number`) VALUES ('102',  
'abbey', 'delhi', 'india', '23456789');
```

```
INSERT INTO `gym`.`branch` (`branch_id`, `street`, `city`, `country`, `phone_number`) VALUES ('103',  
'kiney', 'bombay', 'india', '34567890');
```

```
INSERT INTO `gym`.`branch` (`branch_id`, `street`, `city`, `country`, `phone_number`) VALUES ('104',  
'napey', 'london', 'england', '45678901');
```

```
INSERT INTO `gym`.`branch` (`branch_id`, `street`, `city`, `country`, `phone_number`) VALUES ('105',  
'karant', 'chelster', 'italy', '56789012');
```

### **Supplier**

```
CREATE TABLE `gym`.`supplier` (  
  `supplier_id` INT NOT NULL,  
  `first_name` VARCHAR(45) NOT NULL,  
  `last_name` VARCHAR(45) NULL,  
  `phone_number` INT NOT NULL,  
  `branch_id` INT NOT NULL,  
  PRIMARY KEY (`supplier_id`));
```



```
INSERT INTO `gym`.`supplier` (`supplier_id`, `first_name`, `phone_number`, `branch_id`) VALUES ('201', 'ken', '22331155', '101');
```

```
INSERT INTO `gym`.`supplier` (`supplier_id`, `first_name`, `phone_number`, `branch_id`) VALUES ('202', 'henry', '55446677', '102');
```

```
INSERT INTO `gym`.`supplier` (`supplier_id`, `first_name`, `last_name`, `phone_number`, `branch_id`) VALUES ('203', 'ruby', 'gulzar', '88779900', '103');
```

```
INSERT INTO `gym`.`supplier` (`supplier_id`, `first_name`, `last_name`, `phone_number`, `branch_id`) VALUES ('204', 'paul', 'jennings', '09890980', '104');
```

```
INSERT INTO `gym`.`supplier` (`supplier_id`, `first_name`, `last_name`, `phone_number`, `branch_id`) VALUES ('205', 'karina', 'kaur', '66774411', '105');
```

```
ALTER TABLE `gym`.`supplier`  
ADD CONSTRAINT `branch_id`  
FOREIGN KEY (`branch_id`)  
REFERENCES `gym`.`branch` (`branch_id`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

## Equipment

```
CREATE TABLE `gym`.`equipment` (  
  `equipment_id` INT NOT NULL,  
  `cost` INT NOT NULL,  
  `name` VARCHAR(45) NOT NULL,  
  `supplier_id` INT NOT NULL,  
  PRIMARY KEY (`equipment_id`));
```

```
INSERT INTO `gym`.`equipment` (`equipment_id`, `cost`, `name`, `supplier_id`) VALUES ('301', '6000', 'treadmill', '201');
```

```
INSERT INTO `gym`.`equipment` (`equipment_id`, `cost`, `name`, `supplier_id`) VALUES ('302', '4000', 'cycle', '202');
```

```
INSERT INTO `gym`.`equipment` (`equipment_id`, `cost`, `name`, `supplier_id`) VALUES ('303', '5000', 'cross-trainer', '203');
```

```
INSERT INTO `gym`.`equipment` (`equipment_id`, `cost`, `name`, `supplier_id`) VALUES ('304', '7000', 'stairs', '204');
```

```
INSERT INTO `gym`.`equipment` (`equipment_id`, `cost`, `name`, `supplier_id`) VALUES ('305', '8000', 'cross-cable', '205');
```

```
ALTER TABLE `gym`.`equipment`  
ADD CONSTRAINT `supplier_id`  
FOREIGN KEY (`supplier_id`)  
REFERENCES `gym`.`supplier` (`supplier_id`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

### **customer**

```
CREATE TABLE `gym`.`customer` (  
  `customer_id` INT NOT NULL,  
  `phone_number` INT NOT NULL,  
  `email` VARCHAR(45) NULL,  
  `age` INT NOT NULL,  
  `first_name` VARCHAR(45) NOT NULL,  
  `last_name` VARCHAR(45) NULL,  
  `branch_id` INT NOT NULL,  
  `doctor_approved` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`customer_id`));
```

```
INSERT INTO `gym`.`customer` (`customer_id`, `phone_number`, `email`, `age`, `first_name`,  
  `last_name`, `branch_id`, `doctor_approved`) VALUES ('401', '12343215', 'kina@gmail.com', '20',  
  'kina', 'gart', '101', 'y');
```

```
INSERT INTO `gym`.`customer` (`customer_id`, `phone_number`, `email`, `age`, `first_name`,  
  `last_name`, `branch_id`, `doctor_approved`) VALUES ('402', '66998877', 'robin@hotmail.com', '25',  
  'robin', 'penny', '102', 'n');
```

```
INSERT INTO `gym`.`customer` (`customer_id`, `phone_number`, `email`, `age`, `first_name`,  
  `branch_id`, `doctor_approved`) VALUES ('403', '45678902', 'salman@yahoo.com', '45', 'salman',  
  '103', 'y');
```

```
INSERT INTO `gym`.`customer` (`customer_id`, `phone_number`, `email`, `age`, `first_name`,  
`last_name`, `branch_id`, `doctor_approved`) VALUES ('404', '56432178', 'kriti@gmail.com', '18',  
'kriti', 'len', '104','y');
```

```
INSERT INTO `gym`.`customer` (`customer_id`, `phone_number`, `email`, `age`, `first_name`,  
`branch_id`, `doctor_approved`) VALUES ('405', '98076543', 'kavya@tcd.ie', '22', 'kavya', '105','y');
```

```
ALTER TABLE `gym`.`customer`  
ADD CONSTRAINT `branch_id_customer`  
FOREIGN KEY (`branch_id`)  
REFERENCES `gym`.`branch` (`branch_id`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

```
ALTER TABLE gym.customer  
ADD CHECK (Age>=18);
```

```
ALTER TABLE gym.customer  
ADD CONSTRAINT check_doctor_approved CHECK (doctor_approved IN ('y', 'n'));
```

### **Class**

```
CREATE TABLE `gym`.`class` (  
`class_type` VARCHAR(45) NOT NULL,  
`duration` VARCHAR(45) NULL,  
`employee_id` INT NOT NULL,  
PRIMARY KEY (`class_type`));
```

```
INSERT INTO `gym`.`class` (`class_type`, `duration`, `employee_id`) VALUES ('abs', '45', '301');  
INSERT INTO `gym`.`class` (`class_type`, `duration`, `employee_id`) VALUES ('full-body', '60', '301');  
INSERT INTO `gym`.`class` (`class_type`, `duration`, `employee_id`) VALUES ('cross-fit', '30', '302');  
INSERT INTO `gym`.`class` (`class_type`, `duration`, `employee_id`) VALUES ('spinning', '40', '303');  
INSERT INTO `gym`.`class` (`class_type`, `duration`, `employee_id`) VALUES ('yoga', '50', '305');
```

## Employee

```
CREATE TABLE `gym`.`employee` (  
  `employee_id` INT NOT NULL,  
  `type` VARCHAR(45) NOT NULL,  
  `first_name` VARCHAR(45) NOT NULL,  
  `last_name` VARCHAR(45) NULL,  
  `DOJ-day` INT NOT NULL,  
  `DOJ-month` VARCHAR(45) NOT NULL,  
  `DOJ-year` INT NOT NULL,  
  `salary` INT NOT NULL,  
  `branch_id` INT NOT NULL,  
  PRIMARY KEY (`employee_id`));
```

```
INSERT INTO `gym`.`employee` (`employee_id`, `type`, `first_name`, `last_name`, `DOJ-day`, `DOJ-month`, `DOJ-year`, `salary`, `branch_id`) VALUES ('501', 'trainer', 'kim', 'berry', '20', 'jan', '2015', '5000', '101');
```

```
INSERT INTO `gym`.`employee` (`employee_id`, `type`, `first_name`, `last_name`, `DOJ-day`, `DOJ-month`, `DOJ-year`, `salary`, `branch_id`) VALUES ('502', 'receptionist', 'larry', 'crom', '12', 'march', '2012', '3000', '101');
```

```
INSERT INTO `gym`.`employee` (`employee_id`, `type`, `first_name`, `last_name`, `DOJ-day`, `DOJ-month`, `DOJ-year`, `salary`, `branch_id`) VALUES ('503', 'trainer', 'henry', 'kenny', '14', 'june', '2010', '6000', '102');
```

```
INSERT INTO `gym`.`employee` (`employee_id`, `type`, `first_name`, `DOJ-day`, `DOJ-month`, `DOJ-year`, `salary`, `branch_id`) VALUES ('504', 'manager', 'john', '21', 'dec', '2017', '2000', '102');
```

```
INSERT INTO `gym`.`employee` (`employee_id`, `type`, `first_name`, `last_name`, `DOJ-day`, `DOJ-month`, `DOJ-year`, `salary`, `branch_id`) VALUES ('505', 'trainer', 'kitty', 'sin', '29', 'nov', '2016', '4000', '104');
```

```
ALTER TABLE `gym`.`employee`  
ADD CONSTRAINT `branch_id_employee`  
FOREIGN KEY (`branch_id`)  
REFERENCES `gym`.`branch` (`branch_id`)  
ON DELETE CASCADE  
ON UPDATE CASCADE;
```

### **Customer-class**

```
CREATE TABLE `gym`.`customerclass` (  
  `customer_id` INT NOT NULL,  
  `class_type` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`customer_id`, `class_type`));
```

```
INSERT INTO `gym`.`customerclass` (`customer_id`, `class_type`) VALUES ('401', 'abs');  
INSERT INTO `gym`.`customerclass` (`customer_id`, `class_type`) VALUES ('402', 'yoga');  
INSERT INTO `gym`.`customerclass` (`customer_id`, `class_type`) VALUES ('403', 'yoga');  
INSERT INTO `gym`.`customerclass` (`customer_id`, `class_type`) VALUES ('404', 'spinning');  
INSERT INTO `gym`.`customerclass` (`customer_id`, `class_type`) VALUES ('405', 'spinning');
```

### **Days-in-a-week**

```
CREATE TABLE `gym`.`daysinaweek` (  
  `daysinaweek` INT NOT NULL,  
  `class_type` VARCHAR(45) NOT NULL,  
  PRIMARY KEY (`daysinaweek`, `class_type`));
```

```
INSERT INTO `gym`.`daysinaweek` (`daysinaweek`, `class_type`) VALUES ('5', 'abs');  
INSERT INTO `gym`.`daysinaweek` (`daysinaweek`, `class_type`) VALUES ('2', 'full-body');  
INSERT INTO `gym`.`daysinaweek` (`daysinaweek`, `class_type`) VALUES ('2', 'cross-fit');  
INSERT INTO `gym`.`daysinaweek` (`daysinaweek`, `class_type`) VALUES ('4', 'spinning');  
INSERT INTO `gym`.`daysinaweek` (`daysinaweek`, `class_type`) VALUES ('7', 'yoga');
```