

COMP 3005 - Winter 2024- Project: Health and Fitness Club Management System Project Report

Shaun Sim, Precious Kolawole, Evan Moore

Github: [shaunsim15/comp-3005 \(github.com\)](https://github.com/shaunsim15/comp-3005)

Video: <https://youtu.be/J2ayfwEC7Hw?si=NnizFj9t93AFBHyX>

ERD: https://github.com/shaunsim15/comp-3005/blob/main/Team_2_ERD.pdf

Schema: https://github.com/shaunsim15/comp-3005/blob/main/Team_2_Schema.pdf

DDL/DML: https://github.com/shaunsim15/comp-3005/tree/main/fitness_club/SQL



1. Conceptual Design

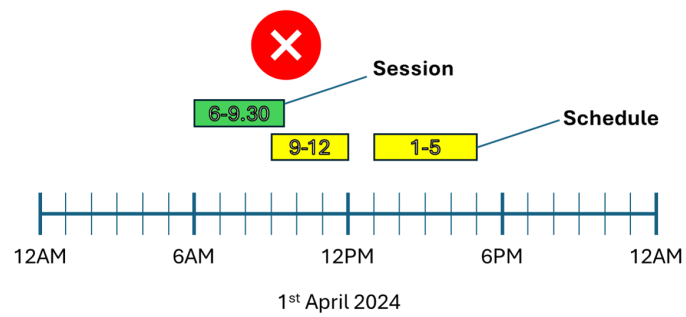
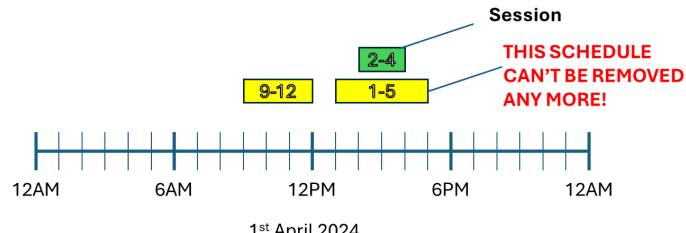
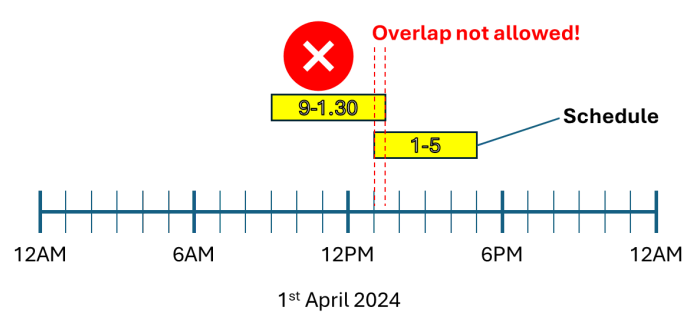
1.1 High-level Explanation

The below table shows how we mapped requirements presented in the form of unstructured text to an ERD. This is similar to the process described in Slide 8 of Lecture 9.

Project Requirement	How this was accounted for in our design at the ERD level
"...a comprehensive platform catering to the diverse needs of club members, trainers, and administrative staff."	<ul style="list-style-type: none">• We decided to have three ERD entities called 'Member', 'Trainer' and 'Admin'.
"Members should be able to register and manage their profiles... they should have access to a personalized dashboard" "Trainers should have the ability to... view member profiles."	<ul style="list-style-type: none">• To support registration, we added email and password attributes for the Member entity (and to Trainer and Admin too, though this wasn't mentioned in the requirements)• While members need to have profiles (viewable by Trainers) and personalized dashboards, we did not think this merited the inclusion of separate ERD entities like 'Profiles' or 'Dashboard'.• We decided that any info needed for a Profile or Dashboard page in the UI would already be present in other entities/tables, and so the UI should be populated using content from these other tables (similar to a non-materialized View).
"Members should... establish personal fitness goals (you can determine suitable fitness goals such as weight and time, and members will set the values"	<ul style="list-style-type: none">• In order to support the establishment of fitness goals, we added 'goal_weight' and 'goal_date' attributes to the Member entity.• These represent a goal weight that a member wishes to reach, and a date by which they aspire to achieve that weight. This is the only type of Goal in our project (Prof Abdelghny said having just one type of Goal was fine).• While having a separate 'Goal' entity might be a more flexible approach if we want to make it easy to define new types of Goal in the future (e.g. a 'calories burnt' Goal), we decided to keep things simple for this project, so

	<p>we did not include one.</p>
<p>"Members should... input health metrics."</p>	<ul style="list-style-type: none"> • The two health metrics that can be inputted for our project are weight and height. • Height is simply a single attribute in the Member entity, as we wouldn't expect this to change much, if at all, over time. • However, Members are very likely interested in tracking how their weight varies over time, regardless of whether they're looking to bulk up or lose some weight. So we created a WeightLog entity, that represents a member's recording/logging of their weight on a given date. We chose to record based on date (date is the partial key of the weak entity) because human weights don't vary significantly on shorter timescales than that (meaning it wouldn't be useful to record multiple WeightLogs on the same day, so we didn't allow for this). • This ties in nicely with the weight goal we defined: members can set a goal in advance, and also keep track of how close they are to achieving it!
<p>"Members can schedule, reschedule, or cancel personal training sessions with certified trainers."</p> <p>"Additionally, they should be able to register for group fitness classes."</p>	<ul style="list-style-type: none"> • To support both 'Personal Training Sessions' and 'Group Fitness Classes', we decided to represent both using a single entity called 'Session'. The 'is_group_booking' boolean attribute is used to indicate whether a given Session is a 'Personal Training Session' or a 'Group Fitness Class'. • We chose to represent these with a single Session entity as it results in fewer tables, and therefore fewer joins- this was one of the desired features of a good design that we saw in Lecture 10, Slide 4. This was important, given that one of the later requirements in the brief involved checking trainers' available hours before creating a Session, and checking that there are no other Sessions booked during a Trainer's hours. This logic would've been significantly complicated by the presence of two different Session-type entities. <p style="text-align: center;"> Good Design = Less NULL + Less Duplicates + Less Tables </p> <ul style="list-style-type: none"> • Each Session (be it a personal session, or a group class) is taught by a single Trainer in the ERD (we discount the possibility of multiple Trainers for a single Session, to keep things simple). • Members can participate in a Session. Multiple Members can participate in a single Group Session, but only one Member can participate in a single Personal Session (note, this latter restriction isn't implemented at the ERD level, so in the diagram, the Member-Session relationship is still many-to-many). • A Member registering for/scheduling a Session involves creating an instance of the Session and the corresponding MemberSession. • A Member canceling a Session involves the reverse- deleting the Session and corresponding MemberSession. Note this is only possible if has_paid_for=False for the MemberSession; all our Sessions are nonrefundable, and we wouldn't want a Member to accidentally cancel a Session they already paid for. • A Member rescheduling a Session involves changing the start_time and

	<p>end_time attributes of the Session. Note this is only possible if the Trainer is free during the new time.</p>
<p>"....tracks exercise routines, fitness achievements, and health statistics (of Members)"</p>	<ul style="list-style-type: none">• We defined a Routine entity that represents an exercise routine (e.g. pushups). Since this is a <u>Fitness Club</u> app, and not a <u>Home Workout</u> app, we assume Members will only ever perform Routines as part of a Session they are enrolled in, and never on their own, at home. Hence, it is OK to link the Routine entity to Sessions in our ERD, rather than directly to Members.• For a given Session and Routine pairing, the routine_count relationship attribute represents the number of times (e.g. 10) that routine (e.g. pushups) is to be performed in that Session (e.g. "Big Mike's Military Workout Session").• We defined an Achievement entity that represents the Achievements which can be completed by a Member.• For a given Member and Achievement pairing, The 'date' relationship attribute represents the date on which the Achievement was completed by that Member.• We decided it wasn't necessary to represent health statistics as their own entity in the ERD, as these are simply aggregate data that can be calculated from other entities (e.g. a Member's average weight can be calculated from all their WeightLogs).
<p>"Trainers should have the ability to manage their schedules"</p> <p>"The system must ensure that the trainer is available"</p>	<ul style="list-style-type: none">• We created a Schedule entity. A given Trainer has many Schedules (a Schedule is a time interval during which a Trainer is available.).• Suppose a Trainer is working a full day (9-5) on 1st April 2024, with a 1 hour lunch break in between. Then they would add two records to the Schedule table to reflect their availability, which on a could be visually represented on a timeline like this:<div><div>9-12</div><div>1-5</div><p>12AM 6AM 12PM 6PM 12AM</p><p>1st April 2024</p></div>• Suppose someone wanted to book a Session during this time. They would need to book a Session whose start and end times lay fully within the start and end time of a Schedule record. So this is allowed:<div><div><div>9-12</div><div>1-5</div></div><div>2-4</div><p>12AM 6AM 12PM 6PM 12AM</p><p>1st April 2024</p></div> <p>But we can't book Sessions with a Trainer that are either wholly or partially outside of their availabilities:</p>

	<div><p>1st April 2024</p><ul style="list-style-type: none">• If a Session has already been booked during a Schedule, we assumed the Trainer should no longer be able to remove that Schedule (otherwise the Members will be left hanging). The logic check to ensure this is not at the ERD level:<div><p>1st April 2024</p><p>THIS SCHEDULE CAN'T BE REMOVED ANY MORE!</p></div><ul style="list-style-type: none">• We further assume that no two Schedules can overlap for a given Trainer. It is for this reason that we can get away with making the start_time the only partial key, and avoid making end_time a partial key as well. This makes our design less wasteful since we don't need to define multiple indices, as Abdelghny mentioned on slide 23 of Lecture 9. So basically, this diagram is disallowed:<div><p>1st April 2024</p><p>Overlap not allowed!</p></div></div>
"Administrative Staff should be equipped with features to manage room bookings... update class schedules"	<ul style="list-style-type: none">• We added a Room entity, which defines a room in which a Session takes place.• Updating a class schedule simply involves changing the start_time and end_time attributes of the Session entity.
"Administrative Staff should be equipped with features to.. monitor fitness equipment maintenance"	<ul style="list-style-type: none">• We added an Equipment entity with attributes to track maintenance of equipment. last_maintained_date is the date on which the equipment last underwent maintenance. days_in_maintenance_interval is how long the equipment can go for without being maintained.
"Administrative Staff should be equipped with features to.. oversee billing, and process payments for membership fees, personal training sessions, and other services"	<ul style="list-style-type: none">• We added a has_paid_for attribute to the 'participates' relationship between Member and Session to indicate if the Member has paid for that Session.• Admins are given the ability to view (i.e. oversee), and also set has_paid_for to 'True', in effect allowing them to 'write off' a Member's

	<p>outstanding payments. This fulfills the payment processing requirement for Sessions.</p> <ul style="list-style-type: none"> • We assumed no membership fees, because our gym follows a pay-per-use philosophy, where Members don't pay regular monthly fees, but only for the Sessions they actually sign up for. Abdelghny confirmed this was fine on Brightspace.
--	---

1.2 Entities in the ER Diagram

1. Member- one of the three kinds of 'user' entities. Represents a member who is registered with the gym.
2. WeightLog- Records in the WeightLog table are used by a member to keep track of their weight over time. This will be used to calculate members' health statistics.
3. Achievement- a predefined fitness-related achievement a member can complete.
4. Session- a class that a member can book, participate in, and pay for. It may be either a personal training session or a group training session. Whether it's a group or personal session is determined by the is_group_booking attribute. Sessions are taught by a Trainer and can be updated by an Admin.
5. Room- where sessions take place. Each Room has a certain capacity which will determine how many members can participate in a Session held in that Room.
6. Equipment- placed in a room. Each piece of equipment needs to be managed to make sure it is being regularly maintained.
7. Routine- performed during Sessions and have corresponding calories burnt (per unit routine).
8. Trainer- one of the three kinds of 'user' entities. Represents a coach who teaches Sessions at the gym.
9. Schedule- a time interval during which a Trainer is available
10. Admin- one of the three kinds of 'user' entities. Represents an employee who works behind the scenes at the gym (they help out with room bookings, etc).

1.3 Relationships in the ER Diagram

The below table explains the relationships in our ERD. Explanations are colour-coded based on whether they relate to **cardinality**, **E1 participation**, **E2 participation**, or none of the three.

Entity 1 (E1)	Entity 2 (E2)	Cardinality	E1 Participation	E2 Participation	Further Explanation
Member	WeightLog	1:N	Partial	Total	<ul style="list-style-type: none">• Each member can have multiple entries in the weight log table since it's used to track the member's weight over time• WeightLog was chosen to be a weak entity since each record is identified by using a combination of the date and the id of the Member that the WeightLog belongs to.• Members have partial participation in this relationship since not all members need to have a record in the table.• Since WeightLog is a weak entity it has total participation in this relationship.
Member	Achievement	M:N	Partial	Partial	<ul style="list-style-type: none">• Many members can have completed a given achievement and many achievements can be completed by a given member• Some members won't have completed any Achievements, and some achievements won't have been completed by any Members.• Each achievement which a member has completed has a corresponding date (the completion date)
Member	Session	M:N	Partial	Partial	<ul style="list-style-type: none">• Many members can participate in one session and many sessions can be participated in by one member.• Some Members won't have been registered for any Sessions, and some Sessions may have no Members (say if an Admin just created the Session, but has not added any Members to it yet).• Every time a member participates in a session there is a corresponding paid_for attribute which indicates if the member has paid for the session.
Session	Room	N:1	Partial	Partial	<ul style="list-style-type: none">• Many Sessions can be held in the same Room as long as they are not happening at the same time• Some Sessions won't have a Room associated with them (this is the default setting when a Member creates a Personal Session- Members are not allowed to reserve a Room, and an Admin would have to do it for them). Some Rooms may not have any Sessions being held in them (say if no one has ever booked that Room).
Session	Routine	M:N	Partial	Partial	<ul style="list-style-type: none">• A Session is composed of multiple Routines, and a Routine may be used in many different Sessions.• Some Sessions won't have any Routines associated

					<p>with them (say if a Trainer has just created a Session, but hasn't decided on any of the Routines they'd like to add to it), and some Routines may not be a part of any Sessions (say if it was an unpopular Routine that no one wants to incorporate in their Session).</p> <ul style="list-style-type: none"> • There is a routine_count relationship attribute which indicates how many times the routine is performed in a specific session.
Session	Trainer	N:1	Total	Partial	<ul style="list-style-type: none"> • A Trainer can lead many Sessions. But we assume each Session can only have 1 Trainer (no co-teaching arrangements) • A Session has total participation in this relationship as each Session needs to be taught by a Trainer (our app doesn't allow Session creation without first specifying a Trainer). Some Trainers may not be teaching any Sessions (e.g. if they were just hired).
Room	Equipment	1:N	Partial	Total	<ul style="list-style-type: none"> • A given Room can hold many pieces of Equipment. But a given piece of Equipment can only be housed in a specific Room. • Some Rooms may contain no Equipment (e.g. if it's used exclusively for sessions that don't require any, like dance sessions) • Equipment has total participation in this relationship as each piece of Equipment needs to be associated with a Room.
Trainer	Schedule	1:N	Partial	Total	<ul style="list-style-type: none"> • A trainer can have many Schedules. But a given Schedule must belong to a specific Trainer. • Schedule is a weak entity as it can't be fully identified by just a start_time and end_time (we also need the trainer_id). • Trainers have partial participation in this relationship since not all Trainers may have a Schedule (For example they are new hires and haven't defined their availability yet). • Schedule has total participation in this relationship as each Schedule needs to have a corresponding trainer (a Schedule without a Trainer makes no sense).

2. Reduction to Relational Schemas

Tables	Explanation
Member	The Members table is a direct translation from the ERD.
WeightLog	Since WeightLog is a weak entity, its

	primary key is formed using the primary key of its owner (member_id) and the date partial key.
MemberAchievement	Since Member and Achievement have a many to many relationship we needed to create a new table to efficiently relate the two separate entities. Its primary keys are formed using the primary keys of Member and Achievement.
Achievement	The Achievement table is a direct translation from the ERD.
MemberSession	Since Member and Session have a many to many relationship we needed to create a new table to efficiently relate the two separate entities. Its primary keys are formed using the primary keys of Member and Achievement.
Session	Session has two foreign keys: room_id and trainer_id. These were needed since Session has a many to one relationship with both room and trainer's.
Room	The Room table is a direct translation from the ERD.
Equipment	Equipment has one foreign key which is room_id. This was needed since Equipment has a many to one relationship with Room.
SessionRoutine	Since Session and Routine have a many to many relationship we needed to create a new table to efficiently relate the two separate entities. Its primary keys are formed using the primary keys of Session and Routine.
Routine	The Routine table is a direct translation from the ERD.
Trainer	The Trainer table is a direct translation from the ERD.

Schedule	Since Schedule is a weak entity, its primary key is formed using the primary key of its owner (trainer_id) and start_date.
Admin	The Admin table is a direct translation from the ERD.

3. DDL File

Tables	Explanation
member	For the member table we made it so that a member's email must be unique and the email and password cannot be null values (Since each member needs to have an email and password to login).
weight_log	The weight log table is set to On Delete Cascade since if a member gets deleted then so should all the members records in the weight_log table.
achievement	The Achievement table is a direct translation from the schema.
member_achievement	The member_achievement table is set to On Delete Cascade since if a member or

	achievement got deleted then the corresponding record in the member_achievement table will be deleted (Since neither exists anymore).
room	The Room table is a direct translation from the Schema.
trainer	The Trainer table is the exact same as the member table except it does not have the attributes goal_weight, goal_date, and height. Also, it obviously has a different primary key.
session	The session table has a check to make sure the price of a session is greater than 0. It is also set to on delete set null if either a specific room or trainer gets deleted since if a room got deleted from the room table or a trainer got deleted from the trainer table we made the decision for the session not to be deleted since in a real life scenario a session could be taught by a different trainer or can be held in a different room
member_session	The member_session table is set to on delete cascade if a member or a session gets deleted since if either of those two events occur then the member_session corresponding records get deleted.
equipment	Equipment has one foreign key which is room_id. This was needed since Equipment has a many to one relationship with Room.
routine	The routine table is a direct translation from the schema.
session_routine	The session_routine table is set to on delete cascade if a session or a routine gets deleted since if either of those two events occur then the session_routine

	corresponding records get deleted.
schedule	The schedule table is set to delete on cascade if a trainer gets deleted since it would not make sense to keep a record of a schedule for a trainer who's still not there.
equipment	The equipment table is set to on delete set null if a room gets deleted since if a room gets deleted that doesn't mean all of the equipment in that room should be gone as well. That equipment could be transferred to another room.
Admin	The Admin table is the exact same as the trainer table, except it has a different primary key.

The function in the ddl file `check_paid_sesh_member_achievement()` calculates the number of paid sessions a member has attended and inserts corresponding achievements into the `member_achievement` table based on predefined thresholds.

The trigger `Paid_sesh_member_achievement` is triggered after an update on the `member_session` table. It calls the `check_paid_sesh_member_achievement()` function to handle achievements based on member session updates.

4. DML File

In the DML, we have included practice data to test our system. Initially, we created logins for members, trainers, and admins to test their respective functionalities and different route accesses. We utilized the `bcrypt` function for password hashing, converting all passwords into hashed format, while also documenting the actual

passwords alongside. We have also added a trainer named Kevin Clark who is available 24/7 all year around. This makes it easier to create a session and not have to worry if the trainer is available.

5. Implementation

Our implementation is a Flask web application built on a two-tier architecture (although we run the client and server on the same local machine, the architecture type is based on the production setting post-deployment, so ours is considered a two-tier architecture). We use PostgreSQL as our database management system (DBMS) and interact with it using psycopg2 (via SQLAlchemy) as the database driver.

Requirement	Implementation
User Registration	<ul style="list-style-type: none">• When the website loads it takes you to the login page.• At this point you can use one of the Member, Admin or Trainer logins from the DML file, or you click on “Sign up” on the top left of the screen.• On the sign up page you can choose if you want to be an “Admin”, “Trainer”, or “Member”.• Code is in auth.py, auth_forms.py, login.html, sign_up.html
Profile Management (Updating personal information, fitness goals, health metrics)	<ul style="list-style-type: none">• A member can update their personal information on the “Update Profile” page.• A member can see their fitness goals on the home page, they also

	<p>can see them on the graph on the “Health Metrics” page (The red dot).</p> <ul style="list-style-type: none"> • A member can update their fitness goals on the “Goals” page. • A member can view their health metrics on the “Health Metrics” page. • Code is in member.py, goal_setting.py, goal_setting_forms.py, health_metrics.py, health_metrics_forms.py, home.py, member/edit.html, home.html, goal_setting.html, health_metrics.html
Dashboard Display (Displaying exercise routines, fitness achievements, health statistics)	<ul style="list-style-type: none"> • Members can view the exercise routines performed in the past week, their fitness achievements, and their health statistics all on the “Dashboard page”. • Code is in dashboard.py, dashboard.html
Schedule Management (Scheduling personal training sessions or group fitness classes. The system must ensure that the trainer is available)	<ul style="list-style-type: none"> • Members can book either a personal training session or sign up for an existing group session on the “Session” page. If the member attempts to book a personal training session with a trainer who is unavailable, an error message will pop up and the member will have to pick an available trainer. • Code is in session.py, session_forms.py, templates/session.
Schedule Management (Trainer can set the time for which they are available.)	<ul style="list-style-type: none"> • Trainers can view, create, update, and delete schedules on the “Schedule” page.

	<ul style="list-style-type: none"> • Code is in schedule.py, schedule_forms.py, templates/schedule.
Member Profile Viewing (Search by Member's name)	<ul style="list-style-type: none"> • Trainers can view profiles on the "View Members" page. They can also search for a member based on their first name, last name, or email. If the trainer wants to view more information about a specific member they just click that member's email. • Code is in member.py, templates/member.
Room Booking Management	<ul style="list-style-type: none"> • Admin can manage room bookings by going to the "Sessions" page and editing the room for existing Sessions (or setting the room for new Sessions). • Code is in session.py, session_forms.py, templates/session.
Equipment Maintenance Monitoring	<ul style="list-style-type: none"> • Admins can view, create, update, and delete equipment on the "Equipment" page (including updating the maintenance dates for equipment). • Code is in equipment.py, equipment_forms.py, templates/equipment.
Class Schedule Updating	<ul style="list-style-type: none"> • Admin can view, create, update, and delete sessions by going to the "Session" page (including updating the time/schedule for the session/class). The updates will be checked to make sure the update is valid. • Code is in session.py, session_forms.py, templates/session.
Billing and Payment Processing (Your system should assume integration with a payment service [Note: Do not actually integrate with a payment service])	<ul style="list-style-type: none"> • Admin can view bills, and write off any unpaid bills on the "Billing" page.

	<ul style="list-style-type: none"> • Code is in <code>billing.py</code>, <code>billing_forms.py</code>, <code>templates/billing</code>.
--	--

6. Bonus Features

Bonus Feature	Explanation
Implementation of a UI	Our implementation includes a fully functional UI that utilizes frameworks such as Bootstrap for a visually appealing design. It also features error messages that pop up when users input does not make sense.
Trigger	The trigger that was implemented was when a member has paid for 1 session, 5 sessions, or 10 sessions, it will add a record in the member achievement table signifying the member has completed these achievements.
Use of ORM	Our implementation uses ORM (SQLAlchemy) which besides its obvious benefits such as abstraction, portability and reusability, it also effectively achieves a similar outcome as a non-materialized view. Similar to how non-materialized views limit the information accessible to non-privileged individuals, we utilize ORM queries to restrict access to information for non-privileged users. For instance, members are unable to view personal sessions booked by other members.
Form validations	In our UI when a user inputs information into a form, it will be checked to make sure it is logically sound. An example of this is when a member logs a weight it makes sure that the inputted date is not in

	the future. It also makes sure that the weight is not less than or equal to zero. If one of those two events occurs then an error message flashes across the screen.
Password encryption in the db	For members, trainers, and admins the bcrypt function is used to encrypt their passwords.
Use of Javascript	Javascript is used to be able to display the billing total dynamically in the edit.html file under the billing folder. It is also used in health_metrics.html to display the members' logged weights over time and goal weight on a line chart.
Ability to choose what routines you want to do as a member.	When a member sets up a personal training session they are able to choose what routines they want to perform, and also choose how many times they want to perform that routine.
Session and schedule logic going above and beyond what's needed.	When creating or updating sessions and schedules there was a lot of extra work done in order to make sure we cover all cases and prevent bad data from being written to the db. See this link to view all cases that were handled: Sessions Logic-ensure it's reflected in code · Issue #7 · shaunsim15/comp-3005 (github.com)
Having a room capacity in Sessions	To make it more realistic we added a limit on the number of participants able to participate in a session, and logic to ensure this limit is not exceeded. We also displayed room occupancy to the user on each Session page.
Uniqueness across all users when it comes to registering an email address	In the code we implement the registration such that no two users have the exact same email. Even if they are of different user types (e.g. Admin and Trainer).