UNIVERSITÀ DEGLI STUDI DI BARI ALDO MORO

Course of Database System

PROJECT: GYM DB

Sgarra Claudia

Mat. 747553

Academic Year: 2020/2021

SUMMARY

GYM DATABASE

Requirement analysis

1) ***Choose the right level of abstraction***
   No changes.

2) ***Linearize phase and break those articulated***
   No changes. There are no excessive complex sentences.

3) ***Standardize sentences***
   We always use the same syntax style: for <**data**> **we represent** <**property**>.
   - For **customer** we represent tax code, name, surname, date of birth and address. For the *customers who train with tools and weights* we represent a fixed monthly price and a personal card assigned.
   - For **course** we represent name, description, calendar (with days of weeks and times in which they take place), the cost. For each day / time in which the course take place, we represent the instructor that hold it. For the *spinning course* we represent the level of difficulty, the customers booked, the number of place available and the instructor who will hold it.
   - For **instructor** we represent tax code, name, surname, years of experience.
   - For **personal card** we represent the name, a series of exercises, series number and number of repetitions.
   - For **exercise** we represent the technical name and a description.

4) ***Identify homonyms and synonyms:***
   We unify terms with same meaning and distinguish terms with different meanings:
   - “customer” has the same meaning of “client” **->** We will use the word “**customer**”.
   - “personal card” has the same meaning of “personal file” **->** We will use the word “**personal card**”.

5) ***Making explicit reference between terms:***
   In the sentence n. 11, it is not clear whether the name is related to personal card or exercises:
   “a personal file consisting of a series of exercises, characterized by name”. **->**
   We explicit the reference to personal card.

6) ***Building a glossary of terms***

Term	**Definition**	**Synonyms**	**Connections**
Customer	Person who train in the gym	Client	Course, Personal card
Course	Training held at a specific date and time with an instructor		Customer, Instructor

Instructor	Person able to train and exercise activities or functions.		Course
Personal card	Series of exercise assigned to a customer who train with tools and weight	Personal file	Customer, Exercise
Exercise	Exercise for training		Personal card

7) Reorganize sentences for keyword sentences:

General phrases
The manager of a gym is interested in storing data relating to their courses and their customers.

Phrases for customer
In particular, for customers it is necessary to store the tax code, name, surname, date of birth and address.

Phrases for course
The courses are characterized by a name, a description, the calendar (days of the week in which they take place and times) and the cost. For each day / time in which the course takes place, it is necessary to store the instructor who will hold it. Among the courses, spinning courses have an organization based on reservations, as there is a limited number of places. For each spinning course, therefore, it is necessary to memorize the level of difficulty, the customers booked, the number of places available and the instructor who will hold it

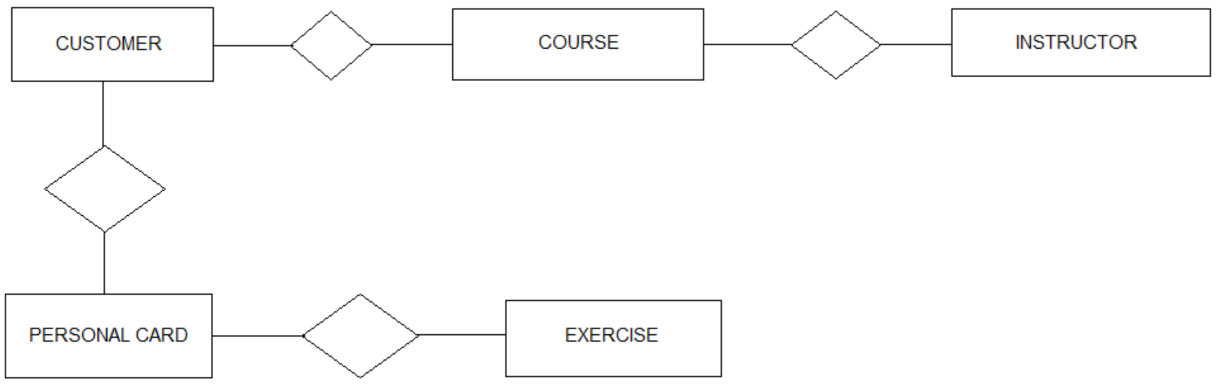
Phrases for instructor
Regarding the instructors, it is necessary to store the tax code, name, surname and years of experience.

Phrases for personal card
Customers who train with tools and weights pay a fixed monthly price and, for them, the gym manager keeps track of the exercises assigned through personal cards. Each interested client will have a personal file consisting of a series of exercises, characterized by name, series number and number of repetitions for each series. The gym manager is interested in maintaining the history of the cards for each customer, with start date and end date.

Phrases for exercise
In particular, each exercise is characterized by a technical name and a description.

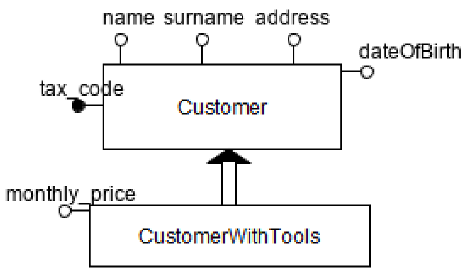
Conceptual Design:

The strategy followed in the phase of conceptual modeling is the hybrid strategy:
starting from the specifications, we will represent all the information in an initial skeleton using a few abstract concepts.
Then, each entity of the scheme will be refined, and the different schemes obtained will be integrated, leading to the final ER scheme, much more detailed than the initial one.



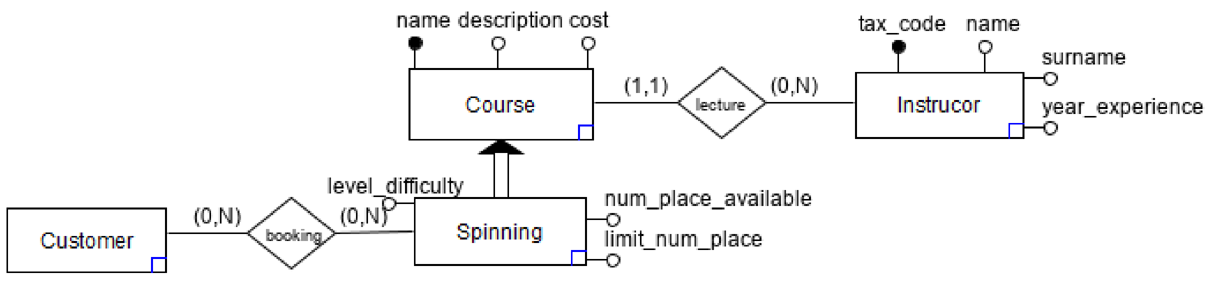
After designing the skeleton, we add all information described in the specifications:

Customer:
The customer is described by tax code, name, surname, date of birth and address.
For the customer who train with tools and weight we need to store the specific attribute of the mouthly cost they pay, so we add a generalization for customer introducing the sub-entity "CustomerWithTools".



Course:
The course is described by name, description, days of week, time and cost. For each day /time, we describe the instructor, so we need to introduce a relationship "Lesson" between "course" and the entity "instructor". A course at a specified day and time is hold by one instructor and an instructor can hold different courses so the cardinality is (1, N).
For the spinning course we need to represent the level of difficulty, limit number of places, number of places available. So, we add a generalization for "course" introducing the sub-entity "spinning". So, spinning course inherits the attribute of course and has the specific attributes.
For spinning course, we also need to represent the booking of customers, because the organization of this course is based on reservation, so we add a relationship between "Customer" and "Spinning" called

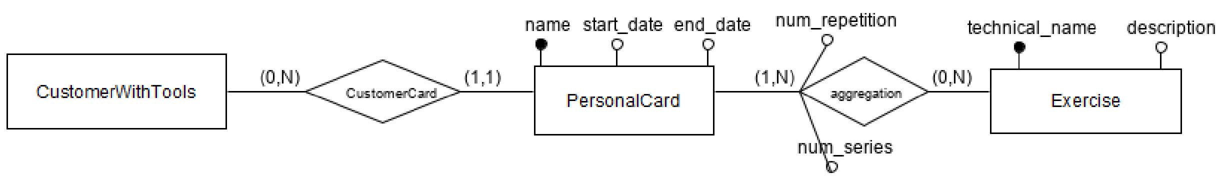
"booking". The cardinality is (N, N) because the customer can book 0 or more spinning courses and the spinning course can have 0 or more bookings (in the limit of number of place).



Personal card:
The personal card is described by the name, start date and end date.
It is assigned to a customer who train with tools so we add a relationship between "PersonalCard" and "CustomerWithTools". A personal card is assigned to only one customer and a customer can have one or more personal card (one for each different type of training for example, so the cardinality of the relationship "CustomerCard" is (1,N).

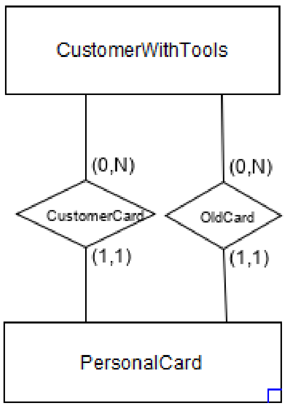
The personal card is composed by different exercises and for each exercise is specified the number of series and the number of repetitions. So we add a relationship between "PersonalCard" and "Exercise" of cardinality (N, N) and add the attributes of this relationship num_series and num_repetition

Exercise:
The exercise is described by a technical name and a description.



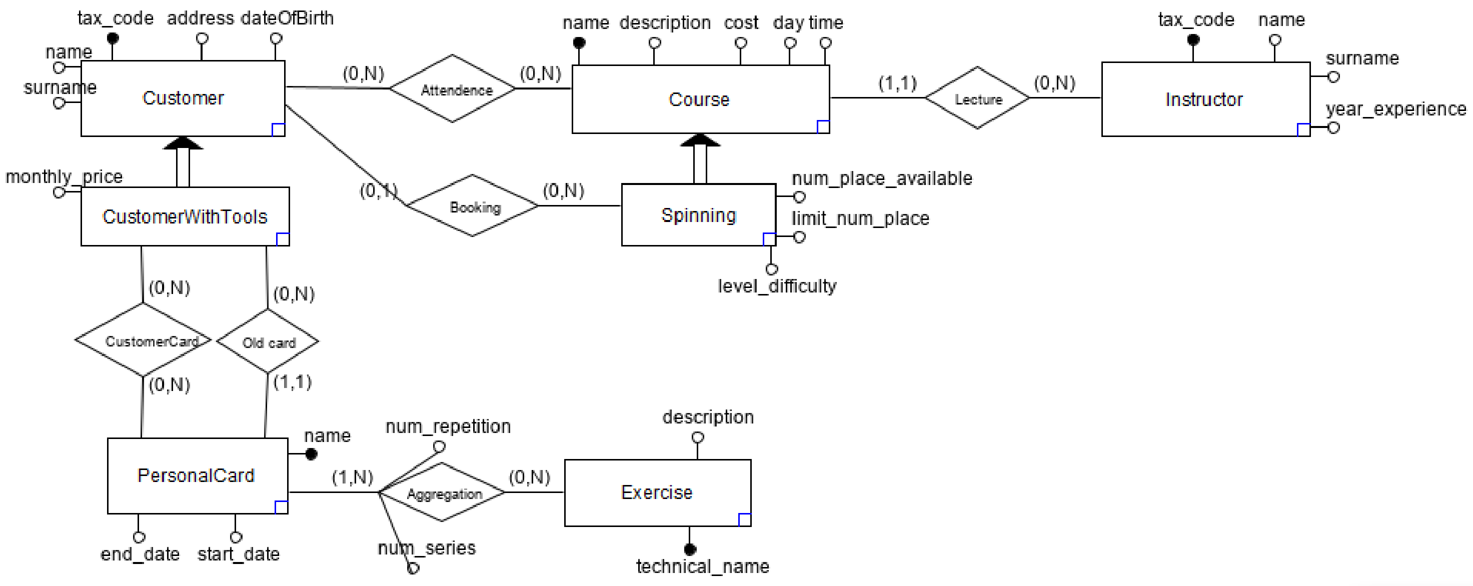
History of the personal card:

To represent the history of the cards for each customer, we add the design pattern concern the archiving of a concept.



Final schema

The **final schema** is obtained by **integration** of the **schemas** obtained up to this point. We connect the entities with the relations provided in the skeleton schema.



Redundancies:

There is a **redundancy**:

- the **number of available places** of the course spinning is obtainable by counting the bookings for the course and subtracting this number from the limit number of places.

Constraints **(that cannot be represented in the ER):**

- a customer could attend the course of spinning only if it booked it.
- the start_date of personalCard must precede the end_data. The old cards are that card for which the end_date is higher than the current date.

Logical design

Before implementing tables and types, we need to check if it is necessary to **restructure** the **conceptual schema** created before designing the logical schema of the object-relational database. To do this, we build **volumes table** and **operation table** and then the **access tables**.

Tabella dei volumi

Concept	Construct	Volume	Explanation
Customer	E	300	
CustomerWithTools	E	100	100 of 300 customers trains with tools and weights
Course	E	20	
Spinning	E	3	
Attendance	R	600	If every customer attendance 2 course a week, every course has 30=(300*2/20) attendence 30*20 courses=600
Instructor	E	50	
Holding	R	100	Each instructor holds 1 course 2 days a week
Booking	R	30	10 for each spinning course
Personal Card	E	1000	
Customer Card	R	200	Customers who train with tools are 200 and each of then as 1 personal card in average
Old card	R	8000	Each customer who train with tools has 4 old cards in average
Exercise	E	500	
Composition	R	10.000	Each Personal Card is composed by 10 exercise

OPERATIONS TABLE

Operation	Type	Frequency
Op1: Data entry for a new customer	I	20 times a week
Op2: Printing the calendar of a given instructor	I	10 times a day
Op3: Booking of a customer for a spinning course	I	30 times a day
Op4: Display of the number of places available for a spinning course	I	100 times a day
Op5: Creating a new personal card for a customer	I	10 times a week

ACCESS TABLE

Op1: Data entry for a new customer

Concept	**Construct**	**Access**	**Type**
Customer	E	1	w

Op2: Printing the calendar of a given instructor

Concept	**Construct**	**Access**	**Type**
Instructor	E	1	r
Lesson	R	2	r
Course	E	2	r

Op3: Booking of a customer for a spinning course

With redundancy:

Concept	**Construct**	**Access**	**Type**
Spinning	E	1	r
Customer	E	1	r
Booking	R	1	w
Spinning	E	1	w

Accesses = 2 * 30 + (2*2) * 30 = 180 (accesses in written value two)

Without redundancy:

Concept	**Construct**	**Access**	**Type**
Spinning	E	1	r
Booking	R	30	r
Customer	E	1	r
Booking	R	1	w

Accesses = (1*30) + (30*30) + (1*30) + (1*2) * 30 = 1020 (accesses in written value two)

Op4: Display of the number of places available for a spinning course

With redundancy:

Concept	**Construct**	**Access**	**Type**
Spinning	E	1	r

Memory occupied: 4*3 = 12 bytes

Accesses = 1*100 times a day = 100 accesses

Without redundancy:

Concept	**Construct**	**Access**	**Type**
Spinning	E	1	r
Booking	R	30	r

Time= 31*100 times a day = 3100 accesses

Op5: Creating a new personal card for a customer

Concept	**Construct**	**Access**	**Type**
CustomerWithTools	E	1	r
CustomerCard	R	1	w
PersonalCard	E	1	w
Exercise	E	10	r
Aggregation	R	10	w

Redundancy analysis:
We decide to maintain the redundancy because is more convenient in terms of times.

Translation into the Object relational model

Implementation (Oracle)

TYPE	TABLE

Create type instructor_ty AS OBJECT(
Tax_code VARCHAR(21),
Name VARCHAR(20),
Surname VARCHAR(20),
Year_experience INTEGER
);
	CREATE TABLE instructors of instructor_ty(
	Tax_code PRIMARY KEY
	);

Create type customer_ty AS OBJECT (Tax_code VARCHAR(21), Name VARCHAR(20), Surname VARCHAR(20), Address VARCHAR(40), dateOfBirth DATE) not final;	Create table customers of customer_ty(Tax_code PRIMARY KEY);
Create type customerWithTools_ty UNDER customer_ty (Month_price FLOAT);	
Create type customers_NT as table of REF customer_ty;	
Create type course_ty AS OBJECT(Id INTEGER, Name VARCHAR(20), Description VARCHAR(50), Cost FLOAT, day VARCHAR(10), Time VARCHAR(5), Instructor REF instructor_ty, Attending_customers customers_NT) not final;	Create table courses of course_ty(Id PRIMARY KEY, day CHECK (day in (‘lunedì’, ‘martedì’, ‘mercoledì’, ‘giovedì’, ‘venerdì’, ‘sabato’, ‘domenica’)), instructor NOT NULL) nested table attending_customers store as attendingCustomersTab ;
Create type spinning_ty UNDER course_ty(Level_difficulty VARCHAR(20), Limit_num_places INTEGER, Num_places_available INTEGER);	
Create type booking_ty AS OBJECT(Id INTEGER, Customer REF customer_ty, Course REF course_ty);	Create table bookings of booking_ty(Id PRIMARY KEY, Customer NOT NULL, Course NOT NULL);
Create type exercise_ty AS OBJECT(Technical_name VARCHAR(20), Description VARCHAR(50));	Create table exercises of exercise_ty(Technical_name PRIMARY KEY);
Create type exerciseForCard_ty AS OBJECT(Exercise REF exercise_ty,	

Num_series INTEGER, Num_repetition INTEGER);	
Create type exercises_nt as table of exerciseForCard_ty;	
Create type personal_card_ty AS OBJECT(Id INTEGER, Name VARCHAR(20), Start_date DATE, End_date DATE, Instructor REF instructor_TY, Customer REF customer_ty, Exercises exercises_nt);	Create table personalCards of personal_card_ty(Id PRIMARY KEY, Instructor NOT NULL, Customer NOT NULL) nested table exercises store as exercisesCardTab;
	Create table oldPersonalCards of personal_card_ty(Id PRIMARY KEY, Instructor NOT NULL, Customer NOT NULL) nested table exercises store as exercisesOldCardTab;

Population of the database

We have implemented procedures to **populate** the tables of the database. The procedures are the following:

POPULATE_CUSTOMERS:
Procedure that inserts **300** tuples in the table Customers. In particular 100 of these 200 customers are customers who train with tools and weight, so they have the attribute price generated randomly. Attributes such as tax_code, name, surname, dateOfBirth are randomized.

POPULATE_INSTRUCTORS:
Procedure that inserts **50** tuples in the table Instructors. Attributes such as tax_code, name, surname, and years_experince are randomized.

POPULATE_COURSES:
Procedure that inserts **20** tuples in the table Courses. Attributes such as name and cost are randomized. IDs are incremental. The instructor is taken randomly. 3 of these 20 courses are spinning courses so they have the name "spinning", the level of difficulty and the limit of number of places. For each course, 30 random participants are inserted in the nested table attending_customers, except for spinning courses whose participants will be the customers that booked it.

POPULATE_BOOKINGS:
Procedure that inserts **30** tuples in the table Bookings. IDs are incremental. The reference to the customer and the reference to the spinning course is taken randomly.

POPULATE_EXERCISES:
Procedure that inserts **500** tuples in the table Exercises. Technical names are randomized.

POPULATE_PERSONALCARDS:
Procedure that inserts **200** tuples in the table PersonalCard. IDs are incremental. Attributes such as name, start_date, end_date. The customer is taken randomly. For each personal card, 10 random exercises are inserted in the nested table exercises who require a list of technical names of exercises and the number of series and the number of repetitions for each of them.

POPULATE_OLDPERSONALCARDS:
Procedure that inserts **800** tuples in the table oldPersonalCard. IDs are incremental. Attributes such as name, start_date, end_date. The customer is taken randomly. For each personal card, 10 random exercises are inserted in the nested table exercises who require a list of technical names of exercises and the number of series and the number of repetitions for each of them.

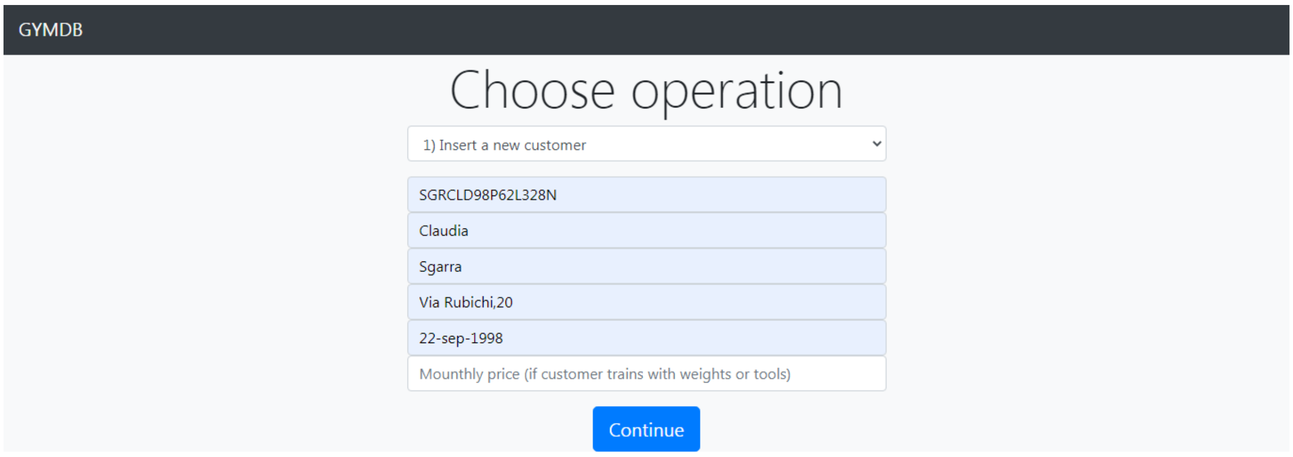
Procedures:

Op1: Data entry for a new customer

INSERT_CUSTOMER(taxCode, name, surname, birth, address, price)

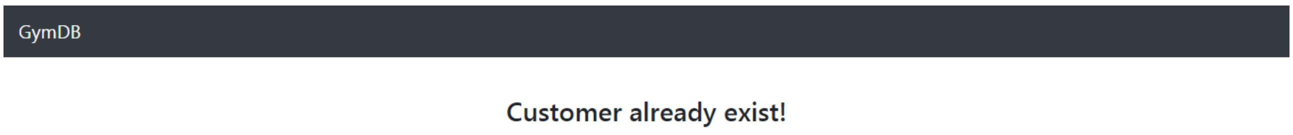
Procedure that, given the tax code, name, surname, date of birth, address and monthly price, insert a new tupel in the table Customers. It begins with the control of the existence of the customers that will be inserted. If it not already exist in the database insert a customer_ty id the attribute price is null, or a customer of type customerWithTools_ty at the contrary.

```
create or replace procedure insert_customer(taxCode varchar, name varchar,
surname varchar, birth date, address varchar, price float) as
exist number;
begin
select count(*) into exist from customers c where c.tax_code= taxcode;
if (exist=0) then
      if (price is null) then
            INSERT into customers values(customer_ty(taxCode, name, surname,
            address, birth));
      else
            INSERT into customers values(customerWithTools_ty(taxCode, name,
            surname, address, birth, price));
      end if;
else
      raise_application_error(-2000, 'The customer already exist');
end if;
end;
```



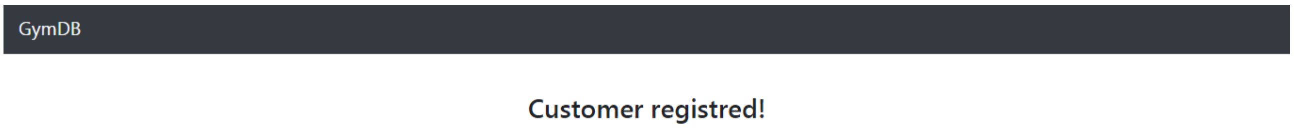
SGRCLD98P62L328N
Claudia
Sgarra
Via Rubichi,20
22-sep-1998
Mounthly price (if customer trains with weights or tools)

If the customer already exists:



Customer already exist!

If the customer do not already exists:



GymDB

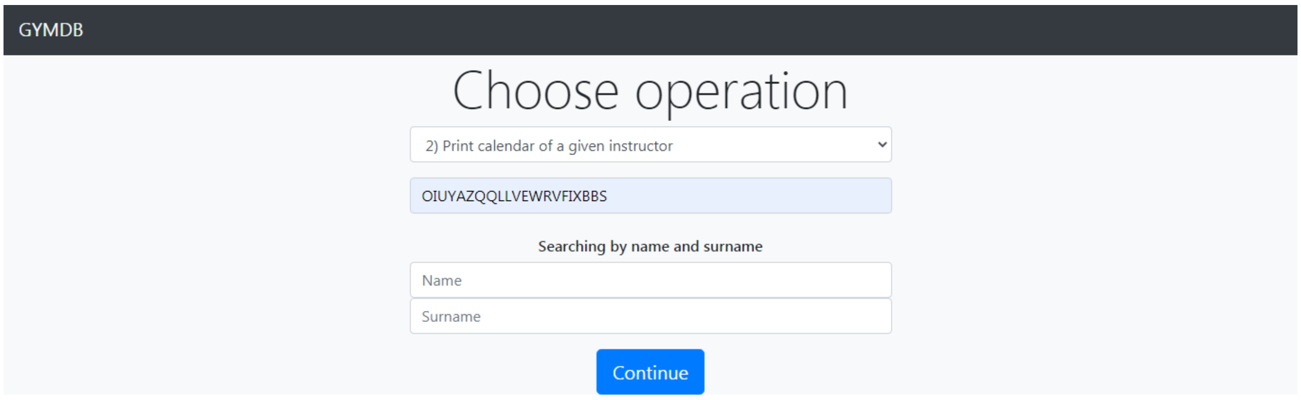
Customer registred!

Op2: Print the calendar of a given instructor

Procedure that prints days and times of the courses holding by a given instructor identified by tax code or name and surname.

```
create or replace procedure print_calendar_instructor (taxCode varchar, nome
varchar, cognome varchar) as
      cursor c is (select * from courses where deref(instructor).tax_code =
      taxCode or (deref(instructor).name=nome and
      deref(instructor).surname=cognome));
      course c%rowtype;
      exist number;
      begin
            select count(*) into exist from instructors where tax_code= taxCode
            or (name=nome and surname=cognome);
         if (exist=0) then
            raise_application_error(-2000, 'The instructor does not exist');
         else
            open c;
            loop
            fetch c into course;
            exit when c%notfound;
            dbms_output.put_line('Course: ' || course.name || ' Day: ' ||
            course.day || ' Time: ' || course.time);
            end loop;
         end if;
      end;
```

You can specify the instructor with only the tax code or with name and surname.



GYMDB

Choose operation

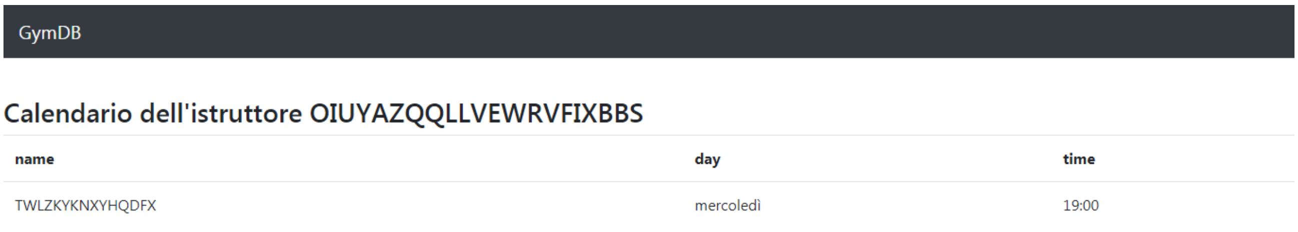
2) Print calendar of a given instructor

OIUYAZQQLLVEWRVFIXBBS

Searching by name and surname

Name

Surname


GymDB

Calendario dell'istruttore OIUYAZQQLLVEWRVFIXBBS

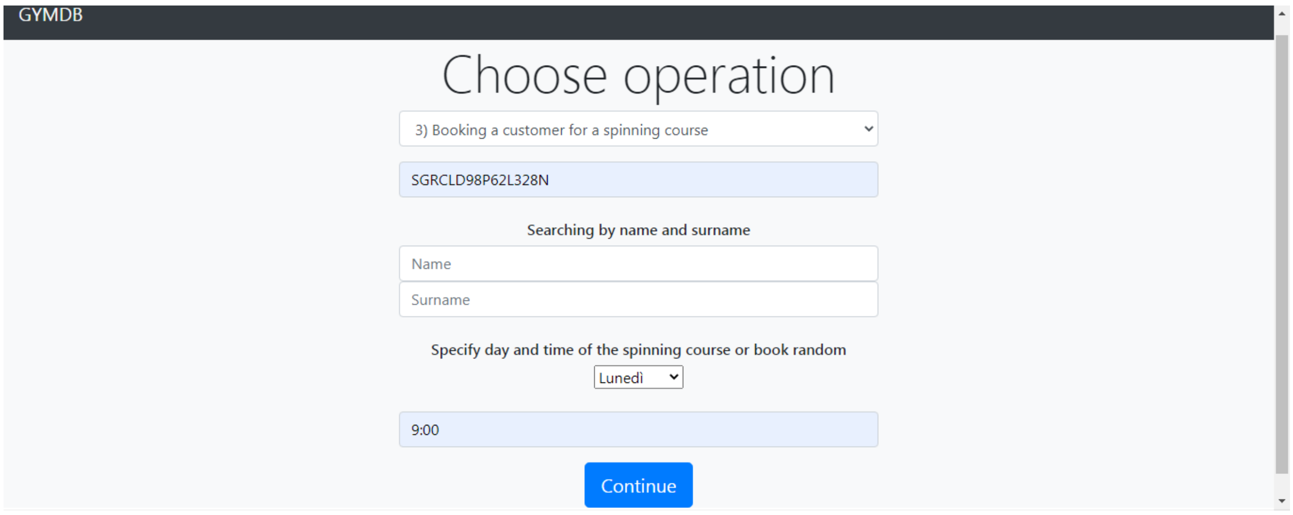
name	day	time
TWLZKYKNXYHQDFX	mercoledì	19:00

Op3: Booking of a customer for a spinning course

BOOK_SPINNING(taxCode, nome, cognome, giorno, ora)

Procedure that insert a booking in the table bookings given the tax code or the name and surname of the customer, and day, time of the spinning course. It controls if the customer exists in the database and proceeds. Then it inserts a booking to a determined course if the day and time are given, otherwise the customer is booked to a spinning course where there are places available.

```
create or replace procedure book_spinning(taxCode varchar, nome varchar,
cognome varchar, giorno varchar, ora varchar) as
existCustomer number;
existCourse number;
cursor c is (select * from courses c where value(c) is of (spinning_ty) and
TREAT(value(c) as spinning_ty).num_places_available>1);
course c%rowtype;
id_booking number;
num_places number;
begin
      select max(id)into id_booking from bookings;
      select count(*) into existCustomer from customers where tax_code=TaxCode
      or (name=nome and surname=cognome);
      if (existCustomer=0) then
            raise_application_error(-2000,'This customer does not exists');
      else if (existCustomer>1) then
            raise_application_error(-2000,'Specify the tax code and try again');
      else
            if((giorno is null) and (ora is null)) then
                  open c;
                  fetch c into course;
                  insert into bookings values(id_booking+1, (select ref(c) from
                  customers c where tax_code=TaxCode or (name=nome and
                  surname=cognome)),
                   (select ref(co) from courses co where id=course.id));
            else
                  select count(*) into existCourse from courses c where value(c)
                  is of (spinning_ty) and day=giorno and time=ora;
                  if (existCourse=0) then
                        raise_application_error(-2000,'The course does not
                        exist');
                  else
                        select TREAT(value(c) as
                        spinning_ty).num_places_available into num_places from
                        courses c where value(c) is of (spinning_ty) and
                        day=giorno and time=ora;
                        if (num_places>0) then
                                    insert into bookings values(id_booking+1,
                                    (select ref(c) from customers c where
                                    tax_code=TaxCode or (name=nome and
                                    surname=cognome)),

                                    (select ref(co) from courses co where
                                    value(co) is of (spinning_ty) and day=giorno
                                    and time=ora ));
                        else
                                    raise_application_error(-2000,'The course is
                                    full');
                        end if;
                  end if;
            end if;
      end if;
end if;
end;
```


GYMDB

Choose operation

3) Booking a customer for a spinning course

SGRCLD98P62L328N

Searching by name and surname

Name

Surname

Specify day and time of the spinning course or book random

Lunedì

9:00

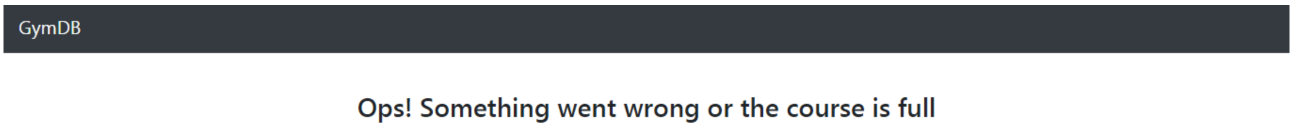
Continue

If the customer exists and there are places available for the course specify:

GymDB

Customer booked!

Otherwise:

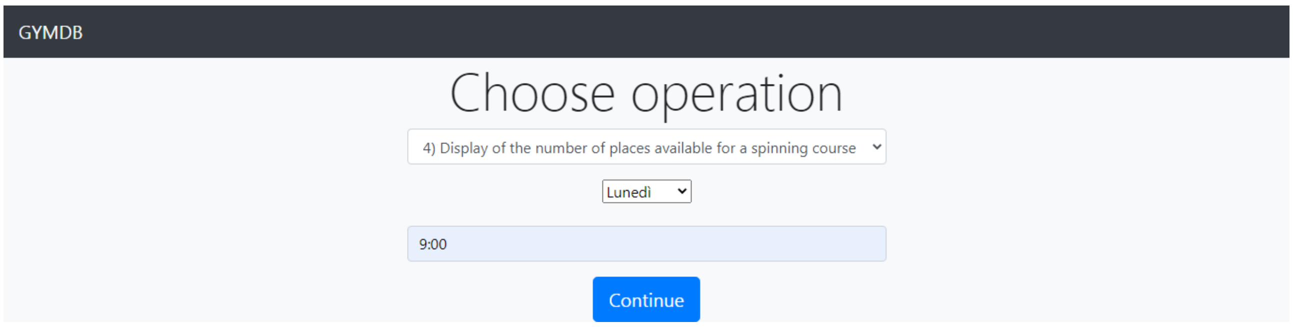

GymDB

Ops! Something went wrong or the course is full

Op4: Display of the number of places available for a spinning course

Procedure that prints the number of places available of a given spinning course characterized by day and time. If there are not courses at that day and time the procedure raises an exception, otherwise insert the booking.

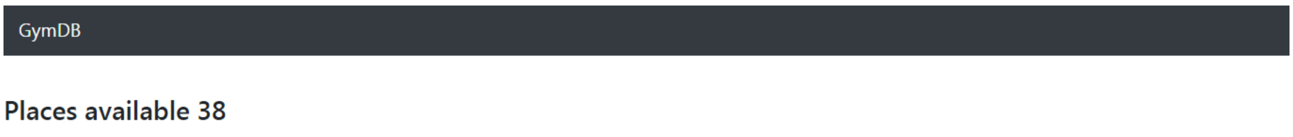
```
create or replace procedure print_places_available(giorno varchar, ora varchar)
as
exist number;
num_places number;
begin
select count(*) into exist from courses c where value(c) is of type
(spinning_ty) and day=giorno and time=ora;
      if (exist=1) then
            select TREAT(value(c) as spinning_ty).num_places_available into
            num_places from courses c where value(c) is of (spinning_ty) and
            day=giorno and time=ora;
      else
            raise_application_error(-2000,'Insert correct information about the
            course');
      end if;
end;
```



4) Display of the number of places available for a spinning course

Lunedì

9:00



Places available 38

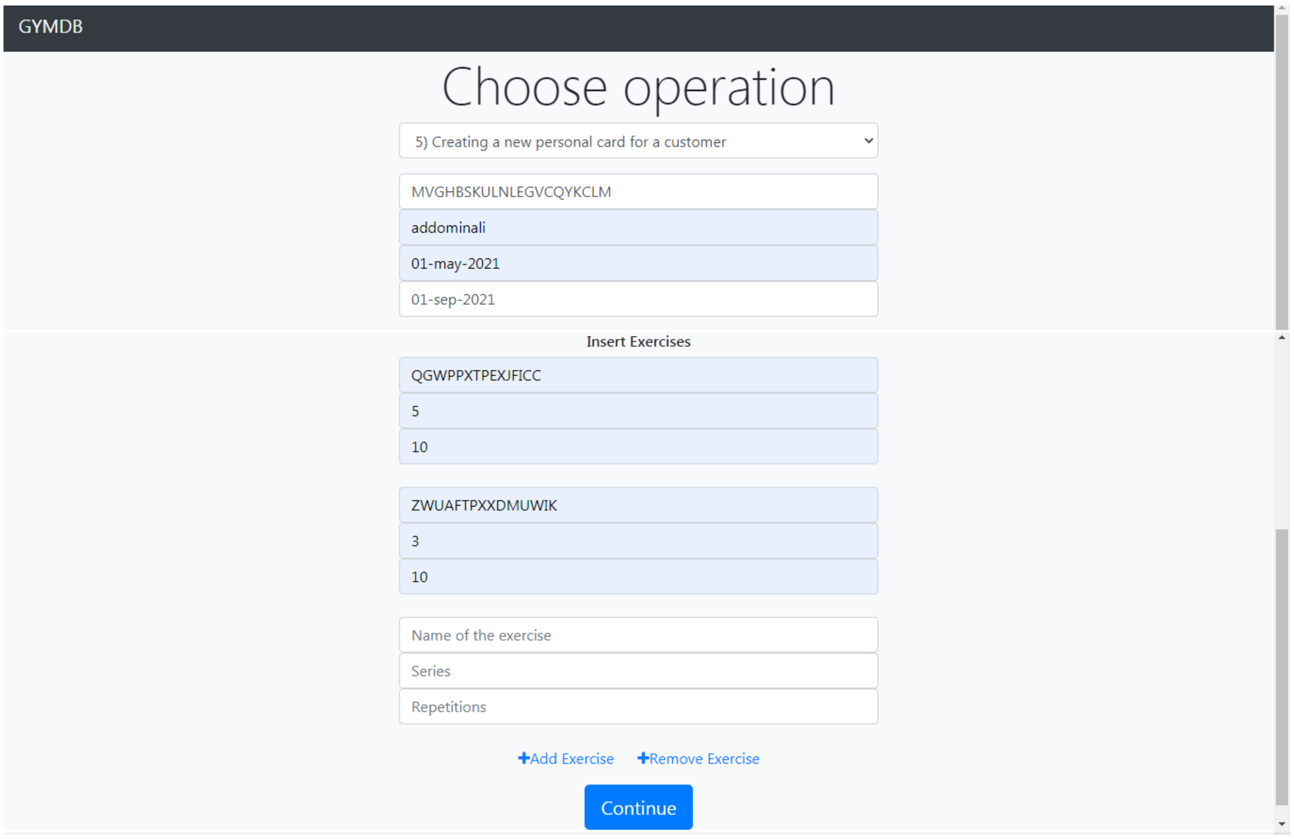
Op5: Creating a new personal card for a customer

CREATE_PERSONAL_CARD(taxCod, nameCard, startDate, endDate, NameExercises, NumSer, NumReps)

Procedure that creates a personal card for a given customer with a list of exercises with the number of series and the number of repetitions for each of them.

If the customer specified exists and, in the database, he is a customer who trains with tools, and the exercises specified are present in the database, the procedure proceeds with the insertion. After insert a new personal card, the exercises will be evaluated.
To each exercise is specified the number of series and repetitions, so in the nested table of exercises of the personal card it is inserted an object with the reference to the exercise, the number of series and number of repetitions.

```
create or replace
PROCEDURE CREATE_PERSONAL_CARD(taxCode VARCHAR2, nameCard VARCHAR2, startDate
DATE, endDate DATE, NameExercises ListExercises, NumSer ListSerie, NumReps
ListRepetitions) as
      Cursor exercisesCursor IS (SELECT * from table(NameExercises));
      Cursor numSerCursor IS (SELECT * from table(NumSer));
      Cursor numRepsCursor IS (SELECT * from table(NumReps));
      newExercise varchar(20);
      numSerExercise number;
      numRepsExercise number;
      maxIDPersCard number;
      ExistsCustWithTools number;
      ExistsExercise number;
      ExistsExercises number;
      NumExercisesInserted number;
      BEGIN
        NumExercisesInserted :=0;
        ExistsExercise :=0;
        ExistsExercises :=0;
        SELECT max(id) into maxIDPersCard from personalcards;
        maxIDPersCard:=maxIDPersCard+1;
        select count(*) into ExistsCustWithTools from customers c where
      value(c) is of (CustomerWithTools_TY) and c.tax_code=TaxCode;
        OPEN exercisesCursor;
        LOOP
        FETCH exercisesCursor INTO newExercise;
        NumExercisesInserted := NumExercisesInserted +1;
        SELECT count(*) into existsExercise from exercises where
      technical_name=newExercise;
        if (existsExercise=1) then
            ExistsExercises := ExistsExercises +1;
        end if;
        existsExercise:=0;
        EXIT WHEN exercisesCursor%NOTFOUND;
        end loop;
        CLOSE exercisesCursor;
        IF(ExistsCustWithTools>0) THEN
            IF (existsExercises=NumExercisesInserted) THEN
            INSERT INTO personalcards VALUES (maxIDPersCard, nameCard,
      startdate, endDate, (select ref(c) from customers c where value(c) is of
      (customerWithTools_ty) and tax_code=taxCode), exercises_NT());
            OPEN exercisesCursor;
            OPEN numSerCursor;
            OPEN numRepsCursor;
            LOOP
            FETCH exercisesCursor INTO newExercise;
            FETCH numSerCursor INTO numSerExercise;
            FETCH numRepsCursor INTO numRepsExercise;
            EXIT WHEN exercisesCursor%NOTFOUND;
            insert into table(
            select exercises
            from personalcards p
            where p.id= maxidperscard
            ) values (exerciseForCard_ty( (select ref(e) from exercises e where
      technical_name=newExercise), numSerExercise, numRepsExercise));
            END LOOP;
            CLOSE exercisesCursor;
            CLOSE numSerCursor;
            CLOSE numRepsCursor;
            ELSE
            raise_application_error(-20002, 'Check if the exercises exist.');
            END IF;
            ELSE
            raise_application_error(-20003, 'Impossible to insert a personal
      card for a customer who does not trains with tools weights. Check also if
      the exercises exist.');
            end if;
            END;
```



GYMDB

Choose operation

5) Creating a new personal card for a customer

MVGHBSKULNLEGVCQYKCLM

addominali

01-may-2021

01-sep-2021

Insert Exercises

QGWPPXTPEXJFICC

5

10

ZWUAFTPXXDMUWIK

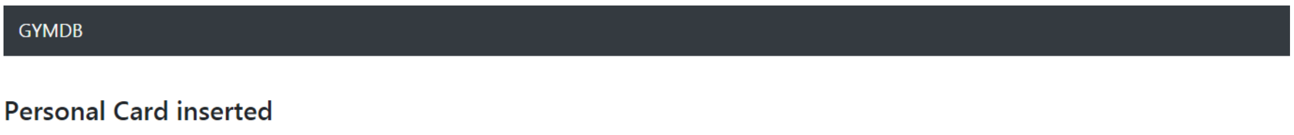
3

10

Name of the exercise

Series

Repetitions



GYMDB

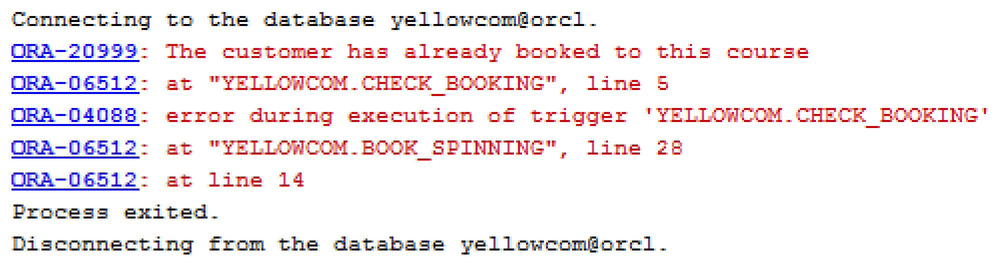
Personal Card inserted

TRIGGER

- **Check booking:** check if the booking for a course already exists before inserting it

```
create or replace trigger check_booking
   before insert on bookings
   for each row
   declare exist number;
   begin
      select count(*) into exist from bookings where customer=:new.customer and
      course=:new.course;
      if (exist>0) then
      raise_application_error(-20999,'The customer has already booked to this
      course');
      end if;
   end;
```

So, if we insert another booking for the same customer and for the same course, the trigger blocks the insert:



```
Connecting to the database yellowcom@orcl.
ORA-20999: The customer has already booked to this course
ORA-06512: at "YELLOWCOM.CHECK_BOOKING", line 5
ORA-04088: error during execution of trigger 'YELLOWCOM.CHECK_BOOKING'
ORA-06512: at "YELLOWCOM.BOOK_SPINNING", line 28
ORA-06512: at line 14
Process exited.
Disconnecting from the database yellowcom@orcl.
```

- **Update number of places available for a spinning course:** for each booking inserted or deleted update the number of places available for the related spinning course.

```
create or replace TRIGGER UPDATE_NUM_PLACES_AVAILABLE
  AFTER INSERT OR DELETE ON bookings
  FOR EACH ROW
  DECLARE
  spinningCourse spinning_ty;
  BEGIN
     IF INSERTING THEN
      select treat(value(c) as spinning_ty) into spinningCourse from courses c
      where c.id=deref(:new.course).id;
      spinningCourse.Num_places_available := spinningCourse.Num_places_available
      - 1;
      update courses c set value(c) = spinningCourse where
      c.id=deref(:new.course).id;
     END IF;
     IF DELETING THEN
      select treat(value(c) as spinning_ty) into spinningCourse from courses c
      where c.id=deref(:new.course).id;
      spinningCourse.Num_places_available := spinningCourse.Num_places_available
      + 1;
      update courses c set value(c) = spinningCourse where
      c.id=deref(:new.course).id;
     END IF;
  END;
```

- **UPDATE ATTENDING CUSTOMERS:** insert the customer to the list of attending customers of a course when insert a booking for that course.

```
create or replace trigger update_attending_course
  AFTER insert ON bookings
  for each row
  declare
  cou course_TY;
  cust customer_ty;
  begin
     insert into table(
     select attending_customers
     from courses
     where id=deref(:new.course).id
     )values((select ref(c) from customers c where
tax_code=deref(:new.customer).tax_code));
  end;
```

JOB

- **INSERT_OLD_PERSONAL_CARD:** executes the procedure insert_oldPersonalCard every 30 seconds. This procedure check if there are some personal cards with past end date, and in this case delete it from the table personalCard and insert it in the table oldPersonalCard.

BEGIN
*DBMS_SCHEDULER.**create_job**(*
job_name => 'INSERT_OLD_PERSONAL_CARD',
job_type => 'STORED_PROCEDURE',
job_action => 'INSERT_OLDPERSONALCARD',
start_date => '30-JUN-2021 10:30:00 AM Europe/Rome',
repeat_interval => 'FREQ=SECONDLY;INTERVAL=30');
dbms_scheduler.enable(name => 'INSERT_OLD_PERSONAL_CARD');
END;

```
create or replace procedure insert_oldPersonalCard as
      cursor c is (select * from personalcards where end_date<current_date);
      cursor e (card number) is (select DEREF(value(ex).exercise).technical_name
      as techName, num_series, num_repetition
      from personalCards p, table(p.exercises) ex
      where p.id= card);
      oldPersonalCard c%rowtype;
      oldExercise e%rowtype;
      newid number;
      begin
      open c;
      loop
         fetch c into oldpersonalcard;
         exit when c%notfound;
```

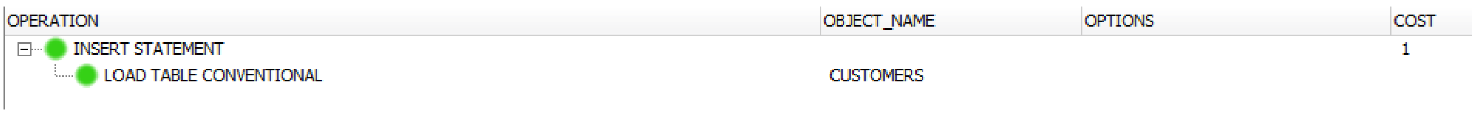
```
   select max(id) into newid from oldpersonalcards;
   newid:=newid+1;
   insert into oldpersonalcards values(newid, oldpersonalcard.name,
   oldpersonalcard.start_date, oldpersonalcard.end_date,
   oldpersonalcard.customer, exercises_NT());

   open e(oldpersonalcard.id);
   loop
      fetch e into oldExercise;
      exit when e%notfound;
      insert into table(
        select exercises
      from oldpersonalcards
      where id=newid
      ) values(exerciseForCard_ty( (select ref(e) from exercises e where
      technical_name=oldExercise.techName), oldExercise.num_series,
      oldExercise.num_repetition));
   end loop;
close e;
end loop;
close c;
delete from personalcards where end_date<current_date;
end;
```

PHYSICAL DESIGN

We analyse the operations given from the specifications to see if they can be optimized:

Op1: Data entry for a new customer

```
INSERT into customers values(customerWithTools_ty('SGRCLD98P62L328N',
'Claudia', 'Sgarra', 'Via Rubichi 20', '22-nov-2021', null));
```

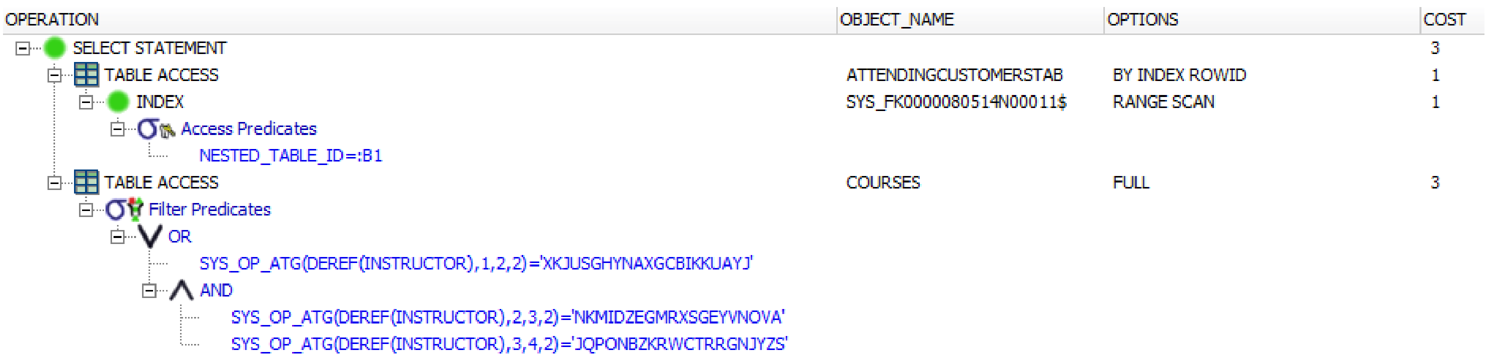


OPERATION	OBJECT_NAME	OPTIONS	COST
● INSERT STATEMENT			1
● LOAD TABLE CONVENTIONAL	CUSTOMERS		

We cannot optimize this operation because is a simple insert statement.

Op2: Printing the calendar of a given instructor

```
select *
from courses
where deref(instructor).tax_code = 'XKJUSGHYNAXGCBIKKUAYJ'
or (deref(instructor).name='NKMIDZEGMRXSGEYVNOVA' and
deref(instructor).surname='JQPONBZKRWCTRRGNJYZS');
```



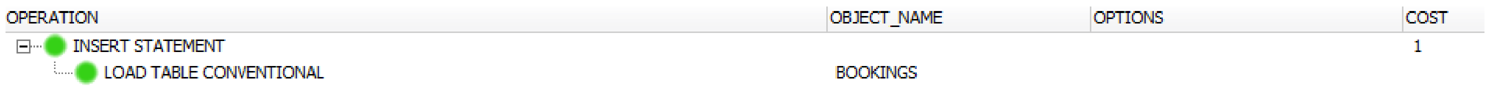
OPERATION	OBJECT_NAME	OPTIONS	COST
● SELECT STATEMENT			3
TABLE ACCESS	ATTENDINGCUSTOMERSTAB	BY INDEX ROWID	1
● INDEX	SYS_FK0000080514N00011$	RANGE SCAN	1
Access Predicates			
NESTED_TABLE_ID=:B1			
TABLE ACCESS	COURSES	FULL	3
Filter Predicates			
OR			
SYS_OP_ATG(DEREF(INSTRUCTOR),1,2,2)='XKJUSGHYNAXGCBIKKUAYJ'			
AND			
SYS_OP_ATG(DEREF(INSTRUCTOR),2,3,2)='NKMIDZEGMRXSGEYVNOVA'			
SYS_OP_ATG(DEREF(INSTRUCTOR),3,4,2)='JQPONBZKRWCTRRGNJYZS'			

Cannot optimize because the selects are on deref.

Op3: Booking of a customer for a spinning course

```
insert into bookings values((select max(id) from bookings)+1,
      (select ref(c) from customers c where tax_code='ZVXEAILPTBYEDTHRODZDV' or
      (name='XEGNIABJMWVLCRZ' and surname='CAWMTOALLOAORKE')),
      (select ref(co) from courses co where value(co) is of (spinning_ty) and
      day='lunedì' and time='9:00' ));
```

The cost for the whole insertion is 1.

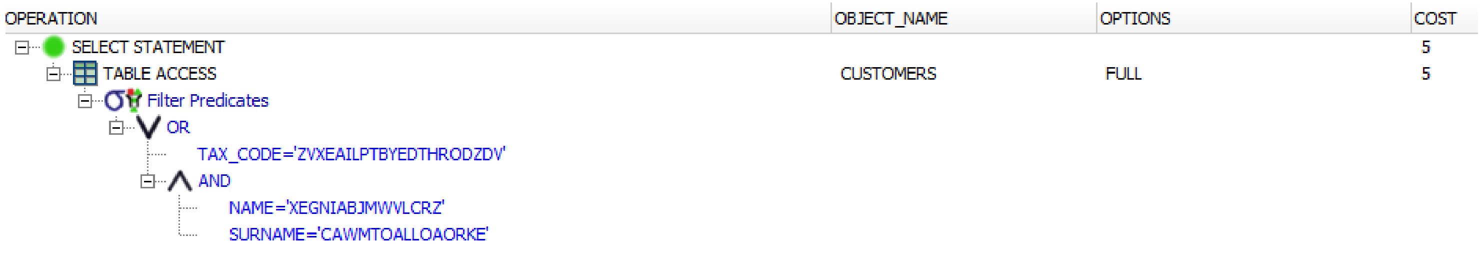


OPERATION	OBJECT_NAME	OPTIONS	COST
INSERT STATEMENT			1
LOAD TABLE CONVENTIONAL	BOOKINGS		

(select ref(c) from customers c where tax_code='ZVXEAILPTBYEDTHRODZDV' or (name='XEGNIABJMWVLCRZ' and surname='CAWMTOALLOAORKE'))

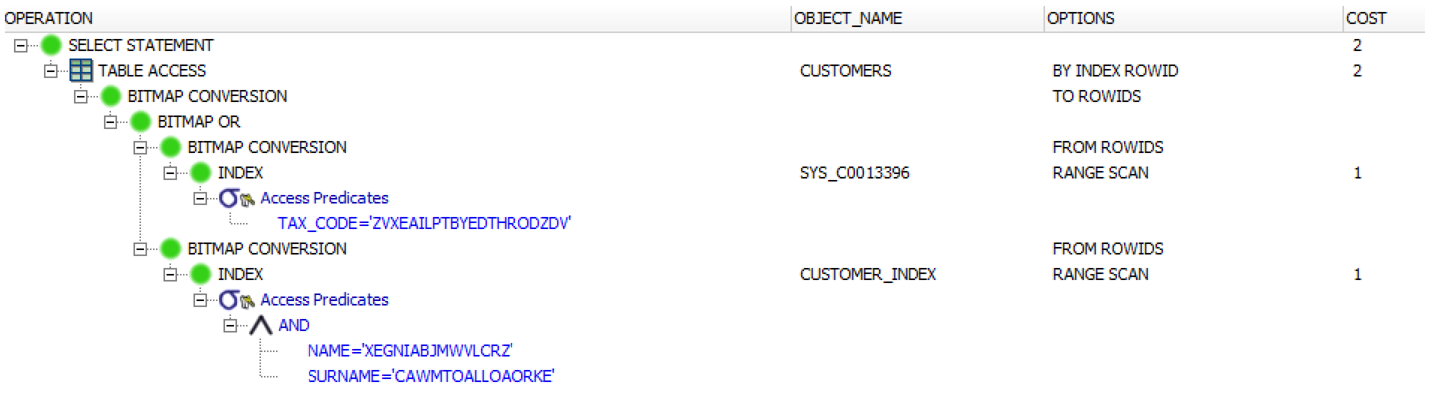
The cost for the first select is 5, while introducing an **index on name and surname** of the customer the cost reduces to 2. This can be conveniente if the number of customers will grown a lot during the next year. So we decide to not remove the index.

INITIAL EXECUTE PLAN:



OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			5
TABLE ACCESS	CUSTOMERS	FULL	5
Filter Predicates			
OR			
TAX_CODE='ZVXEAILPTBYEDTHRODZDV'			
AND			
NAME='XEGNIABJMWVLCRZ'			
SURNAME='CAWMTOALLOAORKE'			

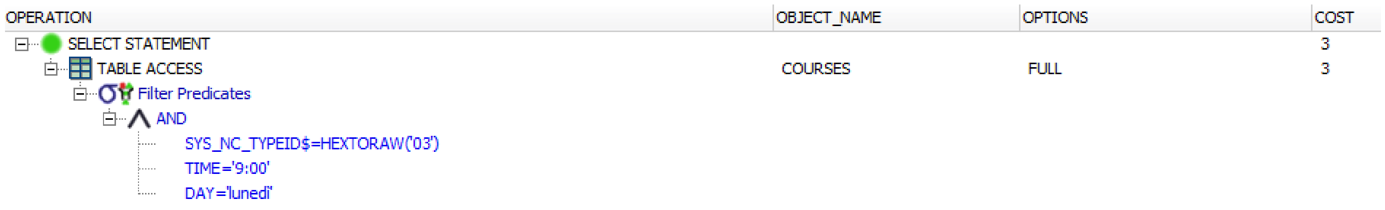
EXECUTE PLAN WITH INDEX:



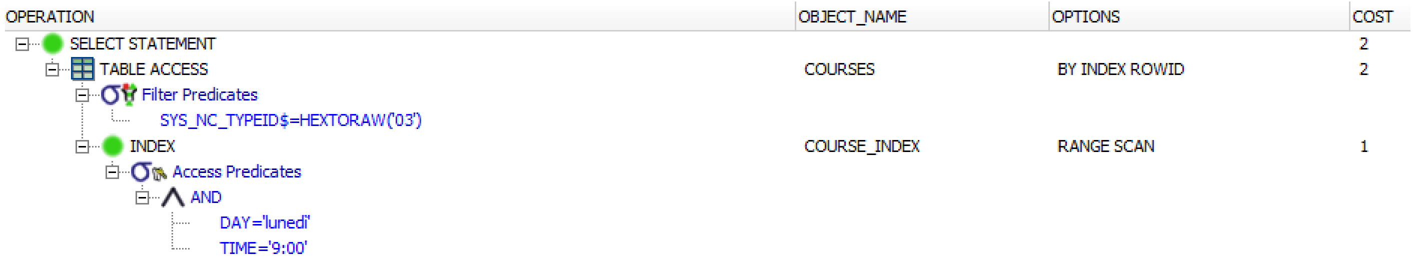
OPERATION	OBJECT_NAME	OPTIONS	COST
SELECT STATEMENT			2
TABLE ACCESS	CUSTOMERS	BY INDEX ROWID	2
BITMAP CONVERSION		TO ROWIDS	
BITMAP OR			
BITMAP CONVERSION		FROM ROWIDS	
INDEX	SYS_C0013396	RANGE SCAN	1
Access Predicates			
TAX_CODE='ZVXEAILPTBYEDTHRODZDV'			
BITMAP CONVERSION		FROM ROWIDS	
INDEX	CUSTOMER_INDEX	RANGE SCAN	1
Access Predicates			
AND			
NAME='XEGNIABJMWVLCRZ'			
SURNAME='CAWMTOALLOAORKE'			

```
(select ref(co) from courses co where value(co) is of (spinning_ty) and
day='lunedì' and time='9:00' ));
```

For the second select the cost is 3 because the number of courses is low, and adding an index on day and time the cost do not reduce a lot, so we do not optimize it.



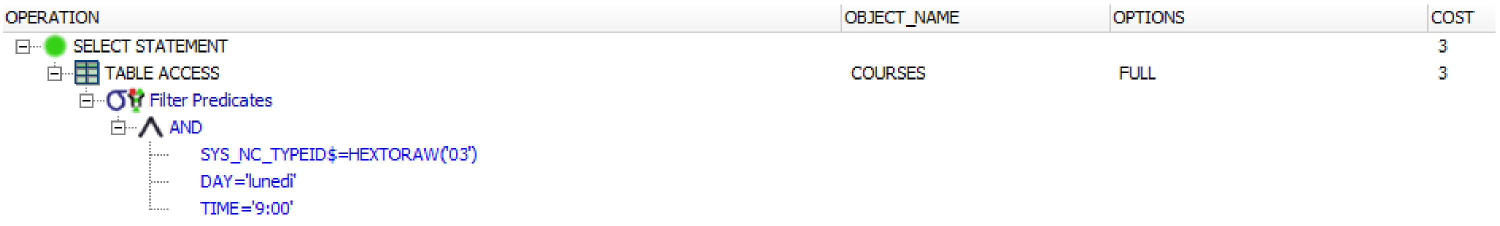
```
OPERATION                                   OBJECT_NAME     OPTIONS          COST
SELECT STATEMENT                                                              3
  TABLE ACCESS                              COURSES         FULL              3
    Filter Predicates
      AND
        SYS_NC_TYPEID$=HEXTORAW('03')
        TIME='9:00'
        DAY='lunedi'
```



```
OPERATION                                   OBJECT_NAME     OPTIONS          COST
SELECT STATEMENT                                                              2
  TABLE ACCESS                              COURSES         BY INDEX ROWID    2
    Filter Predicates
      SYS_NC_TYPEID$=HEXTORAW('03')
    INDEX                                   COURSE_INDEX    RANGE SCAN        1
      Access Predicates
        AND
          DAY='lunedi'
          TIME='9:00'
```

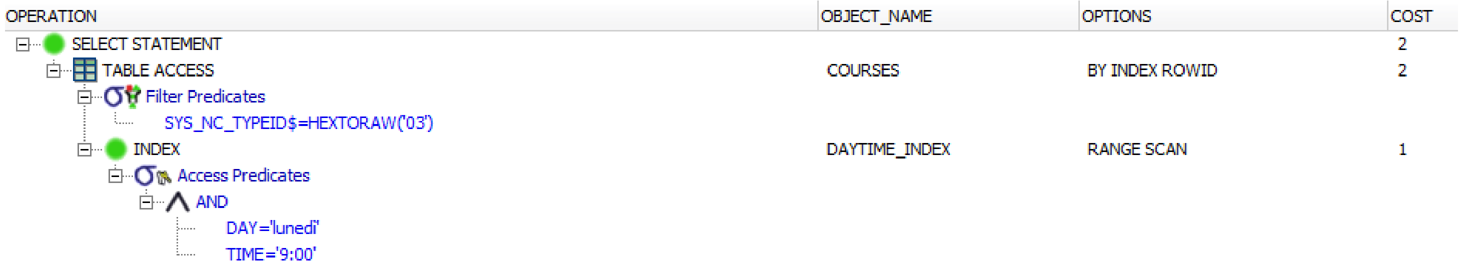
Op4: Display of the number of places available for a spinning course

```
select TREAT(value(c) as spinning_ty).num_places_available
from courses c
where value(c) is of (spinning_ty) and day='lunedì' and time='9:00';
```



```
OPERATION                                   OBJECT_NAME     OPTIONS          COST
SELECT STATEMENT                                                              3
  TABLE ACCESS                              COURSES         FULL              3
    Filter Predicates
      AND
        SYS_NC_TYPEID$=HEXTORAW('03')
        DAY='lunedi'
        TIME='9:00'
```

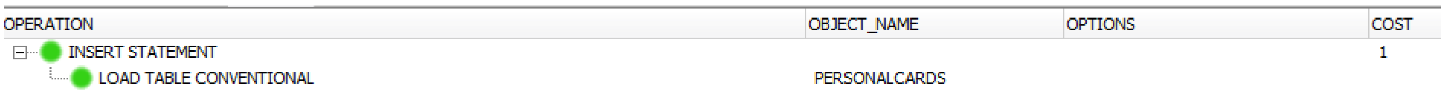
In this case, we should index the **day and time attribute** to reduce the cost of the query, even if it is already low because the courses are 30. If we add the index on those attributes, perhaps, we obtain the **cost 2.** So we decide to not optimize because the cost of using the index to run the query will be greater than running the query without the index.



```
OPERATION                                   OBJECT_NAME     OPTIONS          COST
SELECT STATEMENT                                                              2
  TABLE ACCESS                              COURSES         BY INDEX ROWID    2
    Filter Predicates
      SYS_NC_TYPEID$=HEXTORAW('03')
    INDEX                                   DAYTIME_INDEX   RANGE SCAN        1
      Access Predicates
        AND
          DAY='lunedi'
          TIME='9:00'
```

Op5: Creating a new personal card for a customer

```
INSERT INTO personalcards VALUES ((select max(id) from personalcards)+1,
'cardio', '01-jun-2021', '30-sep-2021',
      (select ref(c) from customers c where value(c) is of
      (customerWithTools_ty) and tax_code='ZVXEAILPTBYEDTHRODZDV'),
      exercises_NT());

insert into table(
select exercises
from personalcards p
where p.id= (select max(id) from bookings)+1,
) values (exerciseForCard_ty( (select ref(e) from exercises e where
technical_name='QGWPPXTPEXJFICC'), 10, 10));
```



OPERATION	OBJECT_NAME	OPTIONS	COST
● INSERT STATEMENT			1
● LOAD TABLE CONVENTIONAL	PERSONALCARDS		

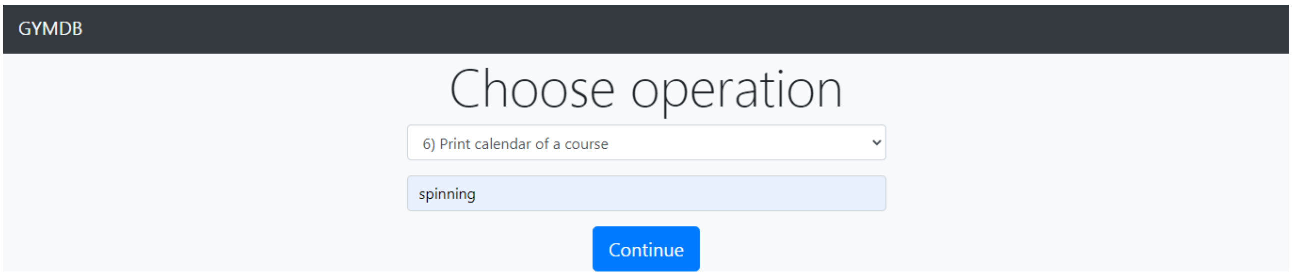
We cannot optimize because the selects on customers and exercises are on primary keys that are already indexed.

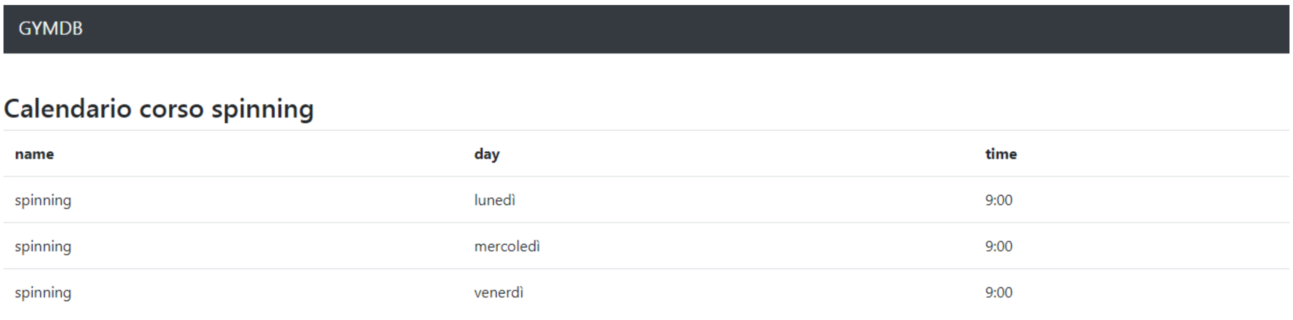
EXTENSIONS

Op6: PRINT_CALENDAR_COURSE (nomeCourse)

Procedure that prints the days and times of a specified course.

```
create or replace procedure print_calendar_course (nome varchar) as
  cursor c is (select * from courses where name=nome);
  course c%rowtype;
  exist number;
  begin
      select count(*) into exist from courses where name=nome;
      if (exist=0) then
            raise_application_error(-2000,'The course does not exist');
      else
            open c;
            loop
            fetch c into course;
            exit when c%notfound;
            dbms_output.put_line('Course: ' || course.name || 'Day: ' ||
            course.day || 'Time: ' || course.time);
            end loop;
      end if;
end;
```

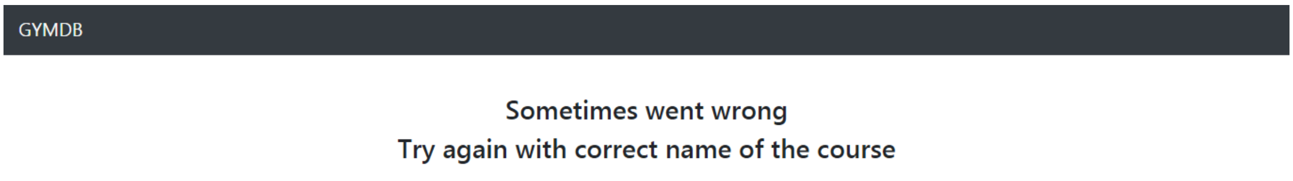




Calendario corso spinning

name	day	time
spinning	lunedì	9:00
spinning	mercoledì	9:00
spinning	venerdì	9:00

If the name of the course is not correct:

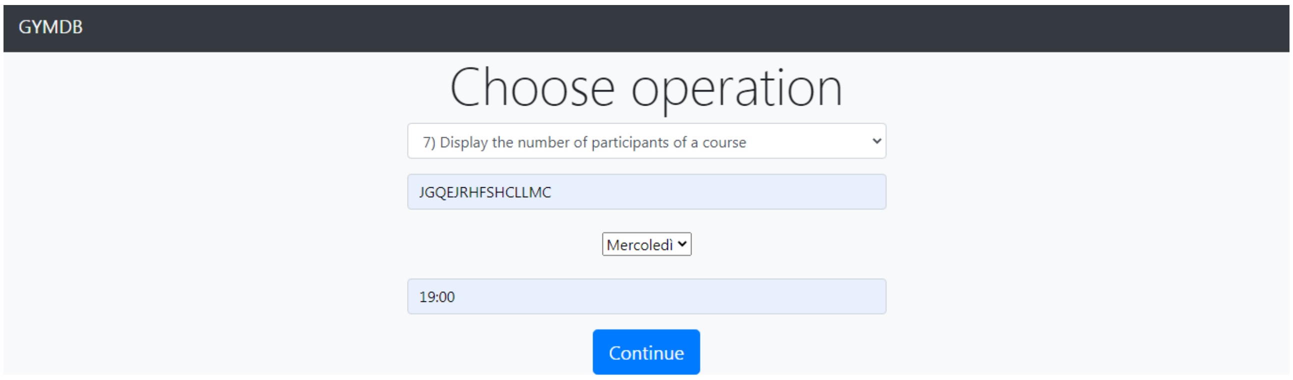


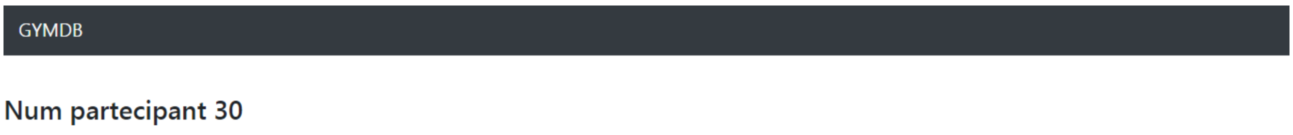
Sometimes went wrong
Try again with correct name of the course

Op7: PRINT_NUM_PARTECIPANT (nameCustomer, dayCourse, timeCourse)

Procedure that shows the number of participants of a course counting the customers in the nested table “attending_customers” of each course.

```
create or replace procedure print_num_partecipant(nome varchar, giorno varchar,
ora varchar) as
   num INTEGER;
   exist integer;
   begin
      select count(*) into exist from courses where name=nome and day=giorno and
      time=ora;
      if (exist=0) then
            raise_application_error(-2000,'The course does not exist');
      else
            select count(*) into num from (table(select attending_customers from
            courses where name=nome and day=giorno and time=ora));
      dbms_output.put_line('The number of partecipant is ' || num);
      end if;
   end;
```



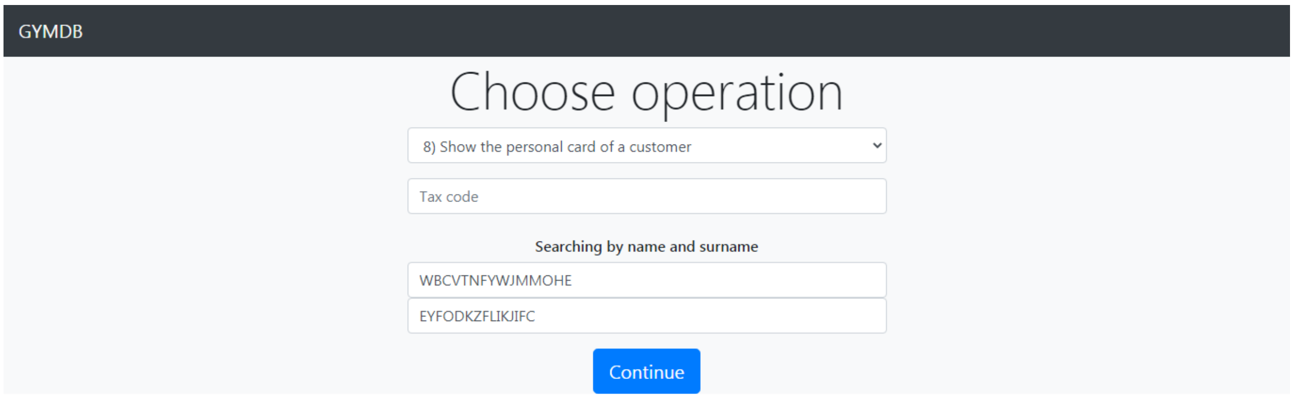


Num partecipant 30

Op8: PRINT_PERSONAL_CARD (taxCodeCustomer, nameCustomer, surnameCustomer)

Procedure that prints the personal card of a given customer with the correspondent exercises, and for each of them, the number of series and the number of repetitions.

```
create or replace procedure print_personal_card(taxCode VARCHAR2, nome
VARCHAR2, cognome VARCHAR2) as
   cursor c is (select p.name, DEREF(value(e).exercise).technical_name as
   techName, DEREF(value(e).exercise).description as descr, num_series,
   num_repetition
   from personalCards p, table(p.exercises) e
   where deref(customer).tax_code=taxCode or (deref(customer).name=nome and
deref(customer).surname=cognome));
   card c%rowtype;
   begin
      open c;
      loop
      fetch c into card;
      exit when c%notfound;
      dbms_output.put_line('Card name ' || card.name || ' Esercizio: ' ||
card.techName || ' Descrizione: ' || card.descr || ' Series: ' ||
card.num_series || ' Repetition: ' || card.num_repetition);
      end loop;
   end;
```


GYMDB

Choose operation

8) Show the personal card of a customer

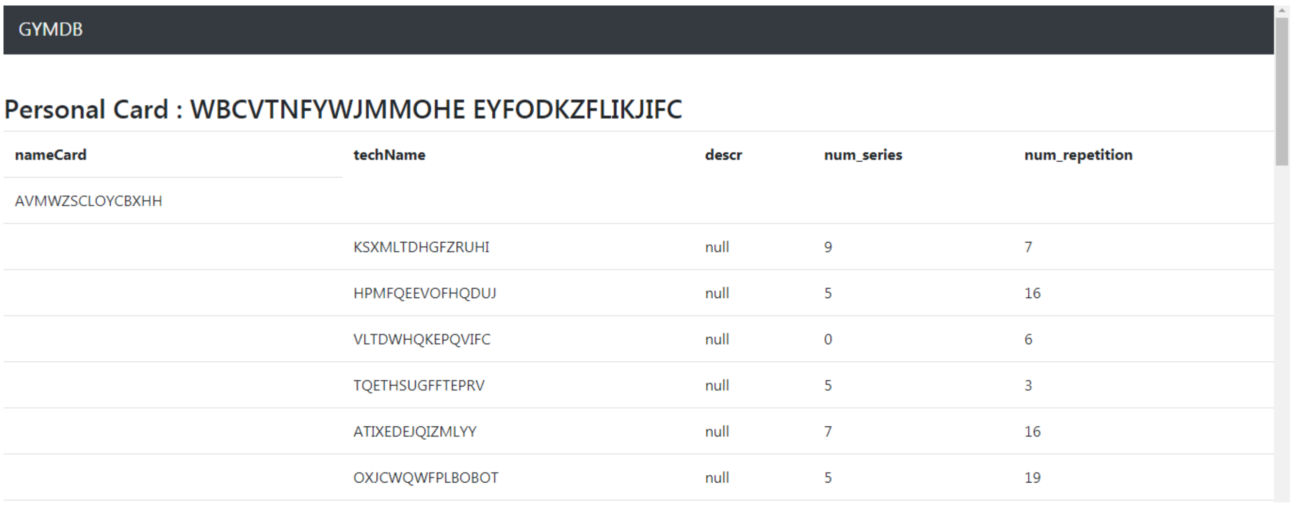
Tax code

Searching by name and surname

WBCVTNFYWJMMOHE

EYFODKZFLIKJIFC

Continue


GYMDB

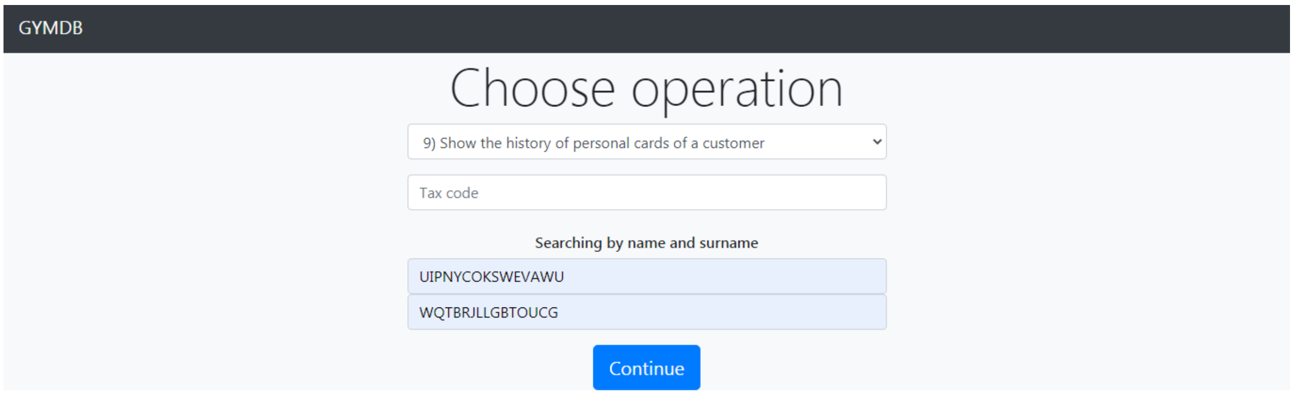
Personal Card : WBCVTNFYWJMMOHE EYFODKZFLIKJIFC

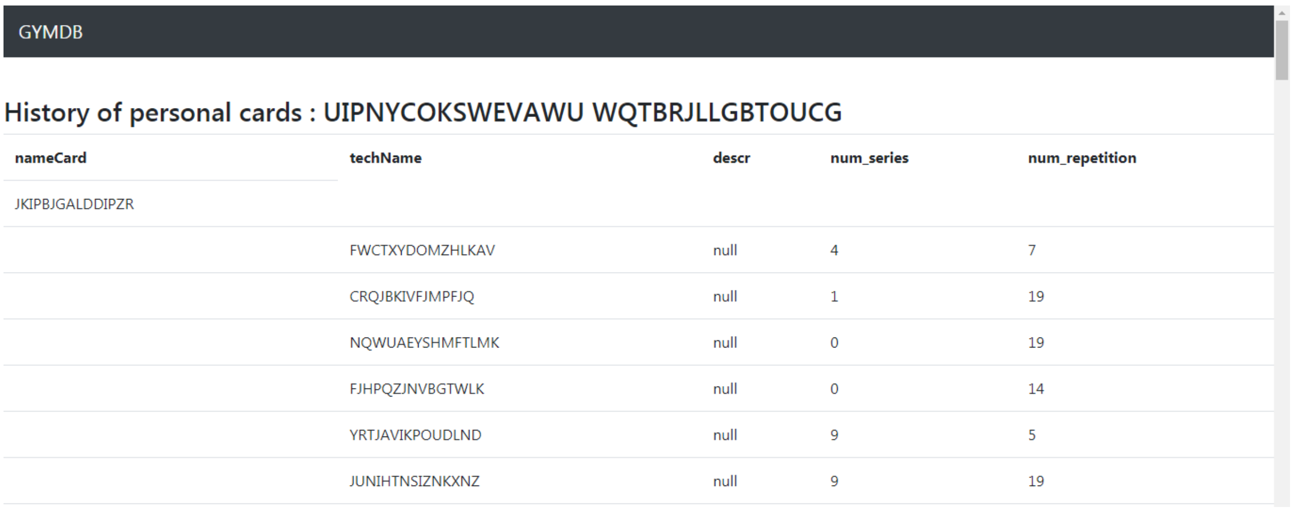
nameCard	techName	descr	num_series	num_repetition
AVMWZSCLOYCBXHH				
	KSXMLTDHGFZRUHI	null	9	7
	HPMFQEEVOFHQDUJ	null	5	16
	VLTDWHQKEPQVIFC	null	0	6
	TQETHSUGFFTEPRV	null	5	3
	ATIXEDEJQIZMLYY	null	7	16
	OXJCWQWFPLBOBOT	null	5	19

Op9: PRINT_OLD_PERSONAL_CARD (taxCodeCustomer, nameCustomer, surnameCustomer)

Procedure as the previous but for old personal cards of a given customer.

```
create or replace procedure print_old_personal_card(taxCode VARCHAR2, nome
VARCHAR2, cognome VARCHAR2) as
   cursor c is (select p.name, DEREF(value(e).exercise).technical_name as
   techName, DEREF(value(e).exercise).description as descr, num_series,
   num_repetition
   from oldPersonalCards p, table(p.exercises) e
   where deref(customer).tax_code=taxCode or (deref(customer).name=nome and
deref(customer).surname=cognome));
   card c%rowtype;
   begin
      open c;
      loop
      fetch c into card;
      exit when c%notfound;
      dbms_output.put_line('Card name ' || card.name || ' Esercizio: ' ||
card.techName || ' Descrizione: ' || card.descr || ' Series: ' ||
card.num_series || ' Repetition: ' || card.num_repetition);
      end loop;
   end;
```


GYMDB
Choose operation
9) Show the history of personal cards of a customer
Tax code
Searching by name and surname
UIPNYCOKSWEVAWU
WQTBRJLLGBTOUCG
Continue



History of personal cards : UIPNYCOKSWEVAWU WQTBRJLLGBTOUCG

nameCard	techName	descr	num_series	num_repetition
JKIPBJGALDDIPZR				
	FWCTXYDOMZHLKAV	null	4	7
	CRQJBKIVFJMPFJQ	null	1	19
	NQWUAEYSHMFTLMK	null	0	19
	FJHPQZJNVBGTWLK	null	0	14
	YRTJAVIKPOUDLND	null	9	5
	JUNIHTNSIZNKXNZ	null	9	19

Setup:

To use the webapp, import the WAR file and go into config.java and change the user and password of the user of Oracle Database, according to the user available in the machine.
Then run the sql script export.sql (to create type, tables and procedures and populating the DB) on that user and start the webapp from index.jsp.