COMP40725

Introduction to Relational Databases and SQL Programming

Name:	**Mike Finn**
Student No:	**13204114**
Project Name:	**pro**

Original Design Document

Client Brief

A golf club wishes to create a database to keep a record of member activity. They want to record any action that the member has relating to the club. The club offers a number of different services that can be availed of by members. Lessons are available for members if they wish to improve their game. There are specialized coaches to help with different areas of the game. The club wishes to keep a record of all of the payments to the club by members. A car parking space can be applied for by a member. Each member needs to apply for a handicap once becoming a member. A member can play in tournaments organized by the club.

Business Rules

The club employs coaches who each have only one specialty such as driving, putting etc.
The club employs more than one coach per specialty.
There is a number of different payment types accepted eg. visa, cheque, cash etc.
The club wants to keep a record of all payments for membership, lessons, and tournament entry.
A member can apply for a designated car parking space.
Each member can have only one handicap that is calculated by the club. Upon joining a handicap must be applied for in the first year.
A player can play in any of the organized tournament once their handicap has been calculated.
Prizes are paid for the best three scores in a tournament.

Assumptions

One member can be enrolled in many lessons.
Lessons have only one coach but there can be many different members in one lesson.
Tournaments are all played over one day.
Only one tournament can occur at any one time.
A member cannot enter a tournament if they have not yet received their handicap.

The club needs to be alerted if a member is approaching 1 year's membership without a handicap.

The club only accepts Visa, Mastercard or cash as payment types.

This database only records information for the tournaments that they organize.

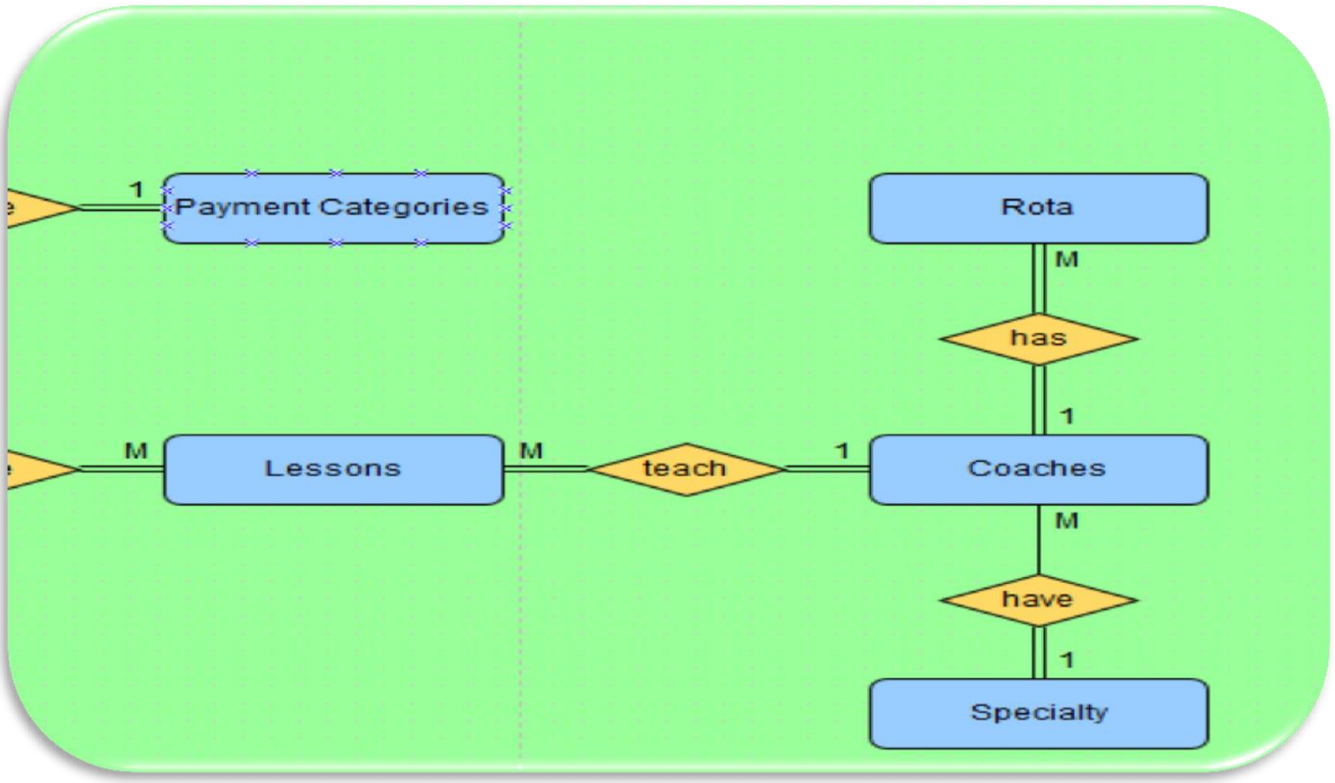
The results of the tournaments need to be stored.

There are only four specialties that the club offers lessons in; Driving, irons, putting, psychology.

There can be more than 1 coach per specialty. There can be a specialty without a coach (transition period between one coach leaving and another being hired).

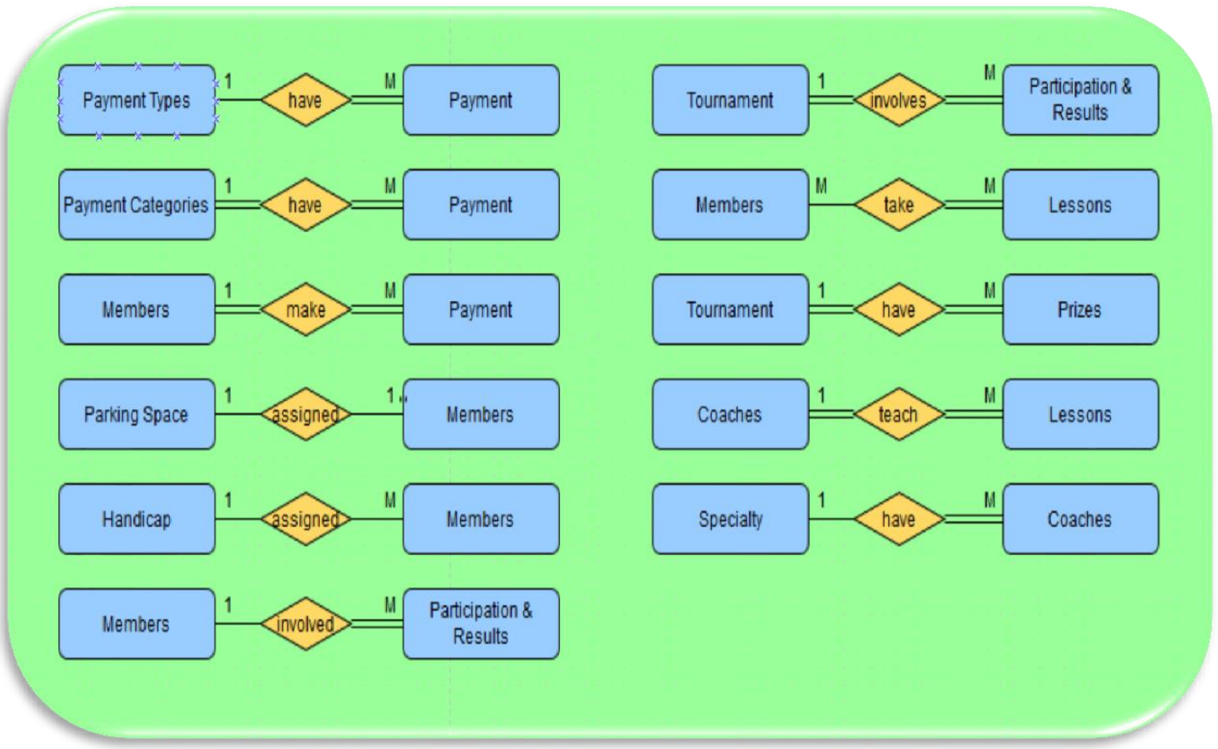
Iterative Process

Originally I was tracking a Rota which had a relationship with Coaches. As this database is only concerned with member activity this was removed. I have included a screenshot of what it had originally looked like:



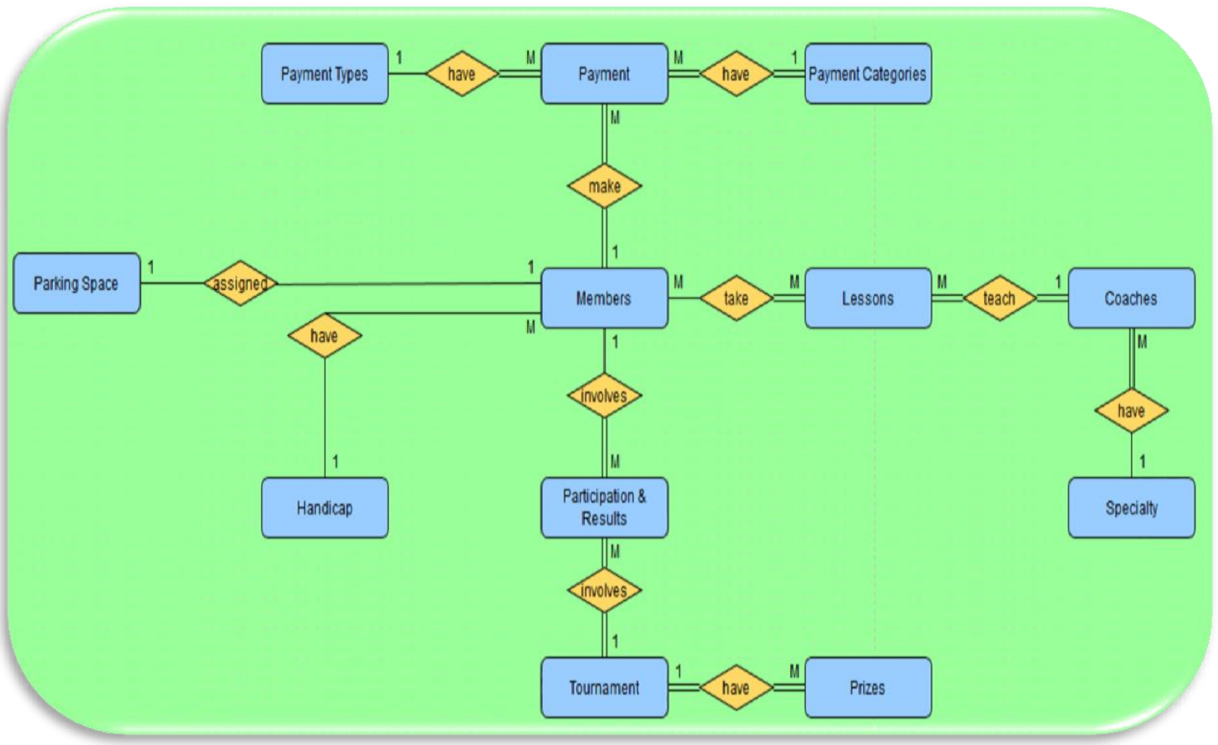
Originally I believed *Participation* would be a junction table between *members* and *tournament* but I realized that I could store results in it so I renamed it to be *Participation & Results* and included it in the ERD

Relationships



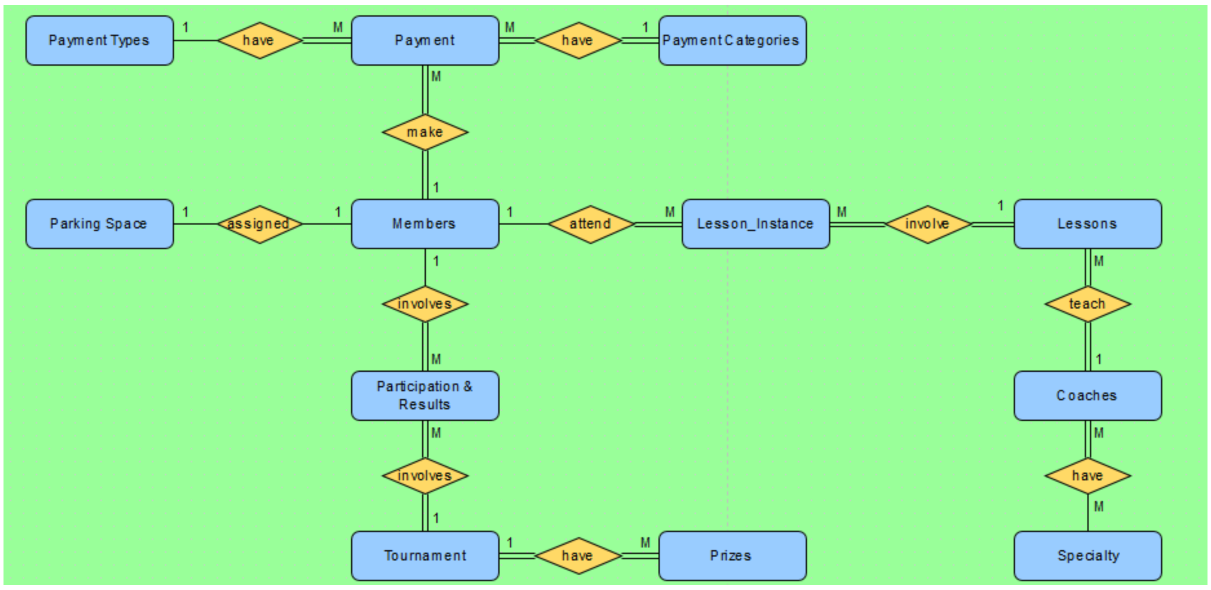
Payment Types 1 have M Payment
Payment Categories 1 have M Payment
Members 1 make M Payment
Parking Space 1 assigned 1 Members
Handicap 1 assigned M Members
Members 1 involved M Participation & Results
Tournament 1 involves M Participation & Results
Members M take M Lessons
Tournament 1 have M Prizes
Coaches 1 teach M Lessons
Specialty 1 have M Coaches

ERD



PROJECT

Updated ERD



Navigating this document

1. I have provided screenshots of the actual queries being processed as well as the results. I have also provided the query text along with any commenting.
2. I have shown screenshots of any tests (mainly for the procedures) and explained what it shows in commenting.
3. Procedure 5 is the most complicated and where I am aiming to get ambition marks.
4. The use of Rollback and SavePoint is illustrated in Procedure 5 processPayment.
5. The use of cursors is shown in Procedures 3, 6 and 7.
6. The first and last trigger are ‘after’ and the second is a ‘before’.
7. For efficiency a lot of my tables hold only numeric values. For easier visualization of the information stored I have created a number of different VIEWS. This will be explained in comments with the code and a screenshot of the typical output.
8. I have shown the code for the complete procedures package (pages 30-43) and then gone back to each procedure in order to show testing. Again, this is further documented in the comments

Points to note

1. All commenting from the database creation is in the databaseCreation.sql file.
2. In the original design document the M:M relationship between members and lessons should have been broken down to include memberID, lessonID and the date the lesson occurred. Originally I believed that it would not break down to reveal another entity but it makes more sense to include the date the lesson occurred there. As a result I have to create a new M:M relationship. This meant that I had to change the business rule regarding one specialty per coach. I have altered this to be:

   A coach can have many specialties and a specialty can have many coaches. This results in the updated ERD which I have attached with this document.

3. I have left out *tee times* (a potential entity) as this is dealt with through another company. This other company is a centralized booking system for a large number of golf courses and as such this entity is not to be included in this database. http://www.brsgolf.com/

4. I have removed the Handicap entity and subsumed the information into the member's entity. It was originally storing only *memberIDs* under each handicap but as many members will have the same handicap any redundancy would be the same as if it was merely an attribute in the *members* entity.

5. There may a few extra records created than is illustrated in the Database creation file as when I was working with the database I needed to create additional tournaments and members, etc.

6. I understand that I could subsume Payment_Types and Payment_Categories into Payment but for efficiency I have decided to leave them as separate entities. Better to store '11' than 'Class 3 Tournament Entry' and '40' for each payment.

7. *"Lessons have only one coach but there can be many different members in one lesson."*

   This assumption has been changed so that all tuition is one-to-one as is actually the case in the golf club.

Additional Business Rules

1. This is a member's only golf course.
2. Handicap is based on scores from rounds played but are not derived in the system. The captain of the golf club calculates this.
3. After consideration the Club has decided to include a number of different payment types: Visa, Mastercard, Cheque, cash, American express, debit card, paypal, western union, etc.
4. Due to higher demand the driving lessons are more expensive.
5. Members need to have an address in the Republic of Ireland.
6. A player cannot play in a tournament unless they have paid in full beforehand.
7. All tournament names are the names of famous golfers followed by 'Open'

Rules to facilitate queries

1. The score in Tournament participation is adjusted by the player for handicap before being entered. From the rules of golf:

   In any round of a handicap competition, the competitor must ensure that his handicap is recorded on his score card before it is returned to the Committee. If no handicap is recorded on his score card before it is returned (Rule 6-6b), or if the recorded handicap is higher than that to which he is entitled and this affects the number of strokes received, he is disqualified from the handicap competition; otherwise, the score stands.

   This means that the actual number of strokes taken can be calculated from the score recorded and the player's handicap.

2. Annual membership was due on the 4th of February 2014
3. Car parking places must be paid for on 1st March 2014
4. The Club is creating a new specialty called fitness but have not hired a coach for this new area as of the submission of this project.
5. Prizes: If there is a draw the value of the 2 or 3 prizes split over the members who drew and the prizes are held back for another tournament.
6. 1 hour lessons start at 6 and 2 hour lessons start at 7pm every weekday at the well-lit driving range. There are no lessons at the weekends. Each coach is available from 6-9 each day during the week.
7. I have created a couple of future tournaments.
8. The car park has 150 spaces available for parking. I have created the 142 extra unused car parking spaces so that they can be assigned rather than created when needed.

4 INNER JOINS

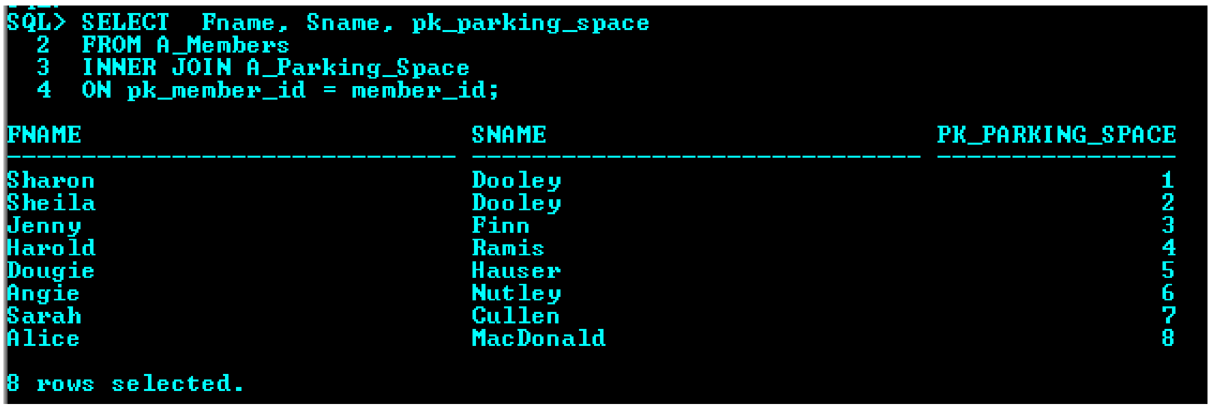
Which members have been assigned Car parking Spaces?

SELECT Fname, Sname, pk_parking_space

FROM A_Members

INNER JOIN A_Parking_Space

ON pk_member_id = member_id;



```
SQL> SELECT  Fname, Sname, pk_parking_space
  2  FROM A_Members
  3  INNER JOIN A_Parking_Space
  4  ON pk_member_id = member_id;

FNAME                          SNAME                          PK_PARKING_SPACE
------------------------------ ------------------------------ ----------------
Sharon                         Dooley                                        1
Sheila                         Dooley                                        2
Jenny                          Finn                                          3
Harold                         Ramis                                         4
Dougie                         Hauser                                        5
Angie                          Nutley                                        6
Sarah                          Cullen                                        7
Alice                          MacDonald                                     8

8 rows selected.
```

Which members were late with paying their membership fees and when did they actually pay them?

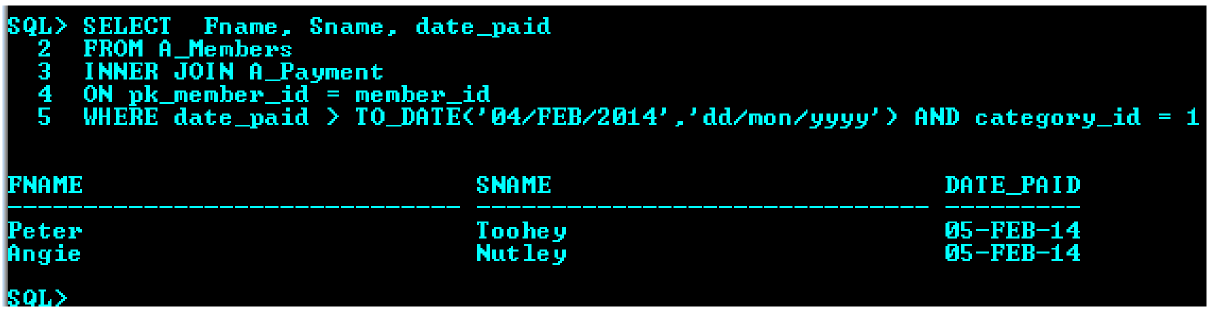
SELECT Fname, Sname, date_paid

FROM A_Members

INNER JOIN A_Payment

ON pk_member_id = member_id

WHERE date_paid > TO_DATE('04/FEB/2014','dd/mon/yyyy') AND category_id = 1;



```
SQL> SELECT  Fname, Sname, date_paid
  2  FROM A_Members
  3  INNER JOIN A_Payment
  4  ON pk_member_id = member_id
  5  WHERE date_paid > TO_DATE('04/FEB/2014','dd/mon/yyyy') AND category_id = 1

FNAME                          SNAME                          DATE_PAID
------------------------------ ------------------------------ ---------
Peter                          Toohey                         05-FEB-14
Angie                          Nutley                         05-FEB-14

SQL>
```

How many payments were made with cash?

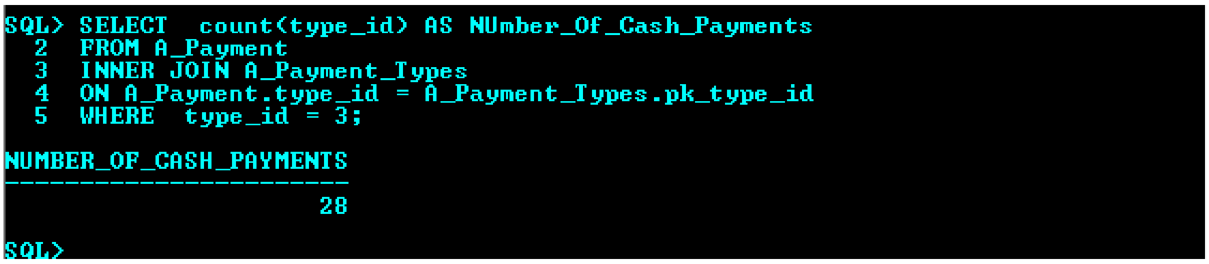
SELECT count(type_id)

FROM A_Payment

INNER JOIN A_Payment_Types

ON A_Payment.type_id = A_Payment_Types.pk_type_id

WHERE type_id = 3;



```
SQL> SELECT  count(type_id) AS NUmber_Of_Cash_Payments
  2  FROM A_Payment
  3  INNER JOIN A_Payment_Types
  4  ON A_Payment.type_id = A_Payment_Types.pk_type_id
  5  WHERE  type_id = 3;

NUMBER_OF_CASH_PAYMENTS
-----------------------
                     28

SQL>
```

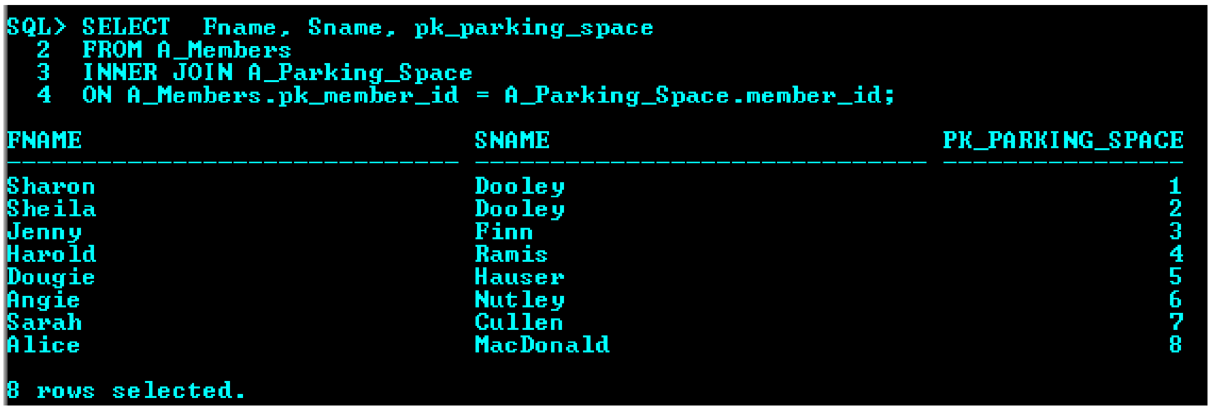
Find a list of all the members with car parking spaces and the space number?

SELECT Fname, Sname, pk_parking_space

FROM A_Members

INNER JOIN A_Parking_Space

ON A_Members.pk_member_id = A_Parking_Space.member_id;



```
SQL> SELECT  Fname, Sname, pk_parking_space
  2  FROM A_Members
  3  INNER JOIN A_Parking_Space
  4  ON A_Members.pk_member_id = A_Parking_Space.member_id;

FNAME                          SNAME                          PK_PARKING_SPACE
------------------------------ ------------------------------ ----------------
Sharon                         Dooley                                        1
Sheila                         Dooley                                        2
Jenny                          Finn                                          3
Harold                         Ramis                                         4
Dougie                         Hauser                                        5
Angie                          Nutley                                        6
Sarah                          Cullen                                        7
Alice                          MacDonald                                     8

8 rows selected.
```

6 OUTER JOINS

2 LEFT OUTER JOINS

Show the parking space status of all of the members including those that do not have assigned parking spaces?

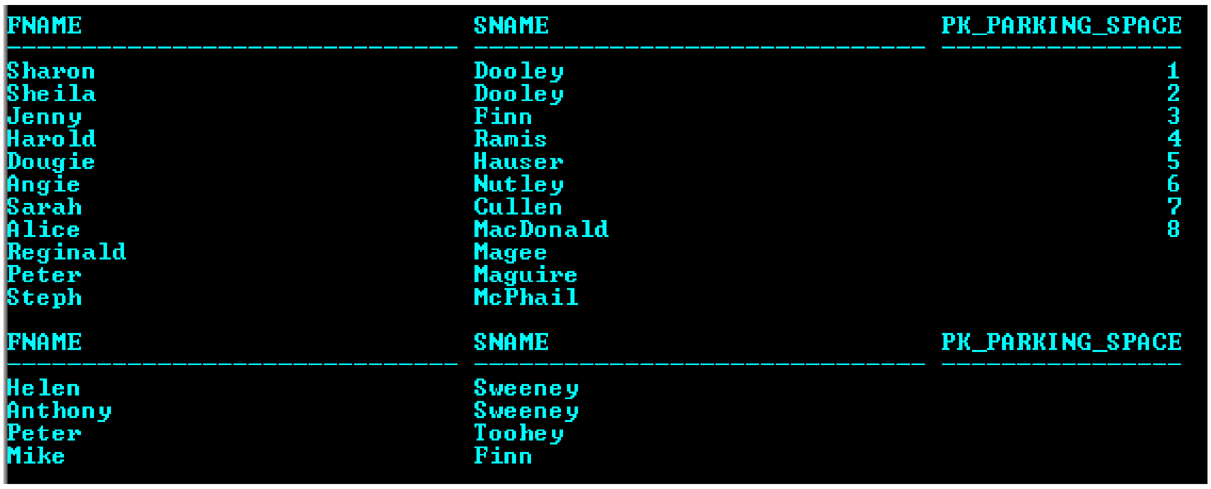
SELECT Fname, Sname, pk_parking_space

FROM A_Members

LEFT OUTER JOIN A_Parking_Space

ON A_Members.pk_member_id = A_Parking_Space.member_id

ORDER BY pk_parking_space;



```
FNAME                          SNAME                          PK_PARKING_SPACE
------------------------------ ------------------------------ ----------------
Sharon                         Dooley                                        1
Sheila                         Dooley                                        2
Jenny                          Finn                                          3
Harold                         Ramis                                         4
Dougie                         Hauser                                        5
Angie                          Nutley                                        6
Sarah                          Cullen                                        7
Alice                          MacDonald                                     8
Reginald                       Magee
Peter                          Maguire
Steph                          McPhail

FNAME                          SNAME                          PK_PARKING_SPACE
------------------------------ ------------------------------ ----------------
Helen                          Sweeney
Anthony                        Sweeney
Peter                          Toohey
Mike                           Finn
```

Which members have taken lessons on what dates and which members have not yet taken lessons?

SELECT Fname, Sname, datetime_lesson

FROM A_Members

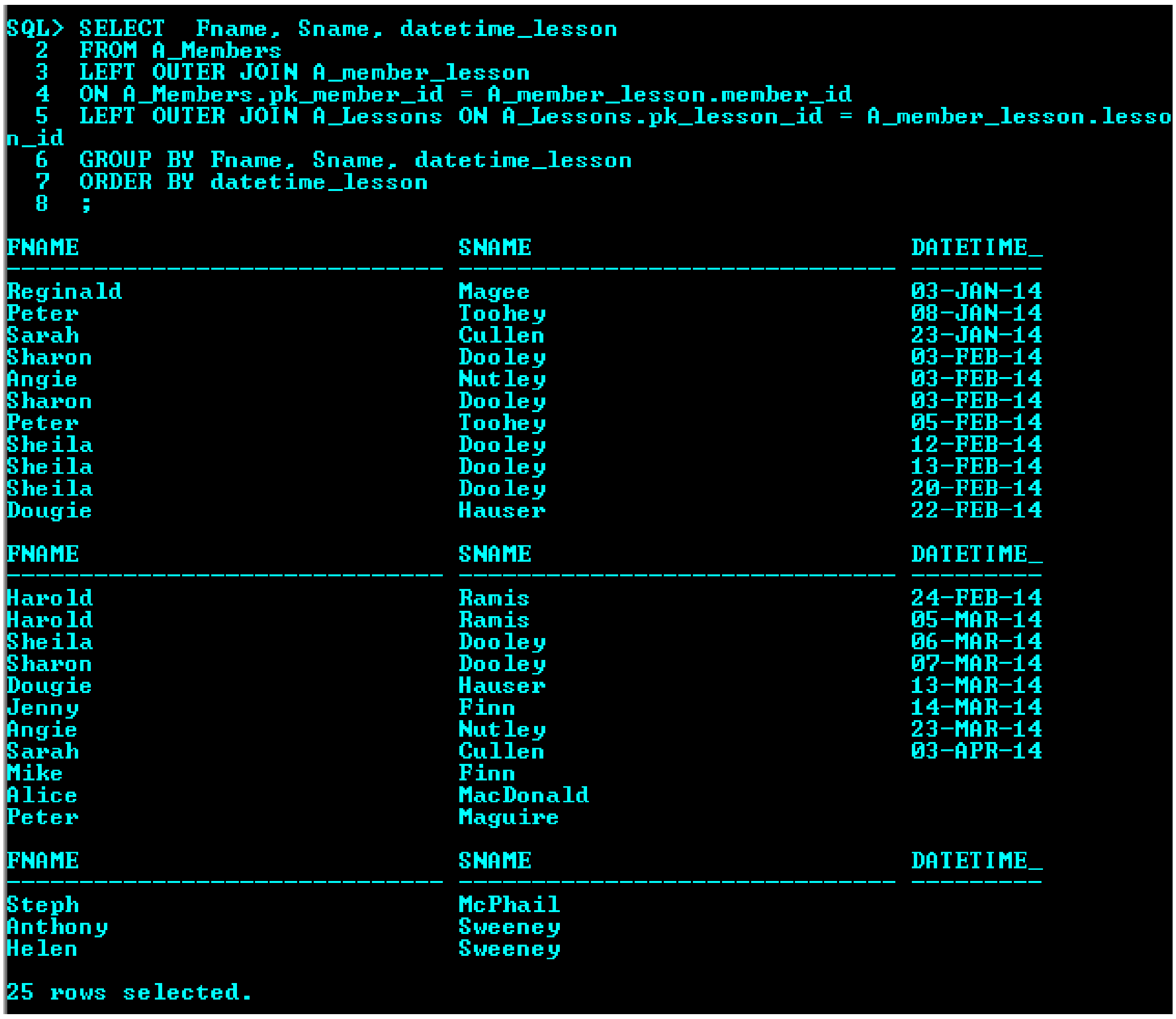
LEFT OUTER JOIN A_member_lesson

ON A_Members.pk_member_id = A_member_lesson.member_id

LEFT OUTER JOIN A_Lessons ON A_Lessons.pk_lesson_id = A_member_lesson.lesson_id

GROUP BY Fname, Sname, datetime_lesson

ORDER BY datetime_lesson;


```
SQL> SELECT  Fname, Sname, datetime_lesson
  2  FROM A_Members
  3  LEFT OUTER JOIN A_member_lesson
  4  ON A_Members.pk_member_id = A_member_lesson.member_id
  5  LEFT OUTER JOIN A_Lessons ON A_Lessons.pk_lesson_id = A_member_lesson.lesso
n_id
  6  GROUP BY Fname, Sname, datetime_lesson
  7  ORDER BY datetime_lesson
  8  ;

FNAME                          SNAME                          DATETIME_
------------------------------ ------------------------------ ---------
Reginald                       Magee                          03-JAN-14
Peter                          Toohey                         08-JAN-14
Sarah                          Cullen                         23-JAN-14
Sharon                         Dooley                         03-FEB-14
Angie                          Nutley                         03-FEB-14
Sharon                         Dooley                         03-FEB-14
Peter                          Toohey                         05-FEB-14
Sheila                         Dooley                         12-FEB-14
Sheila                         Dooley                         13-FEB-14
Sheila                         Dooley                         20-FEB-14
Dougie                         Hauser                         22-FEB-14

FNAME                          SNAME                          DATETIME_
------------------------------ ------------------------------ ---------
Harold                         Ramis                          24-FEB-14
Harold                         Ramis                          05-MAR-14
Sheila                         Dooley                         06-MAR-14
Sharon                         Dooley                         07-MAR-14
Dougie                         Hauser                         13-MAR-14
Jenny                          Finn                           14-MAR-14
Angie                          Nutley                         23-MAR-14
Sarah                          Cullen                         03-APR-14
Mike                           Finn
Alice                          MacDonald
Peter                          Maguire

FNAME                          SNAME                          DATETIME_
------------------------------ ------------------------------ ---------
Steph                          McPhail
Anthony                        Sweeney
Helen                          Sweeney

25 rows selected.
```

2 FULL OUTER JOINS

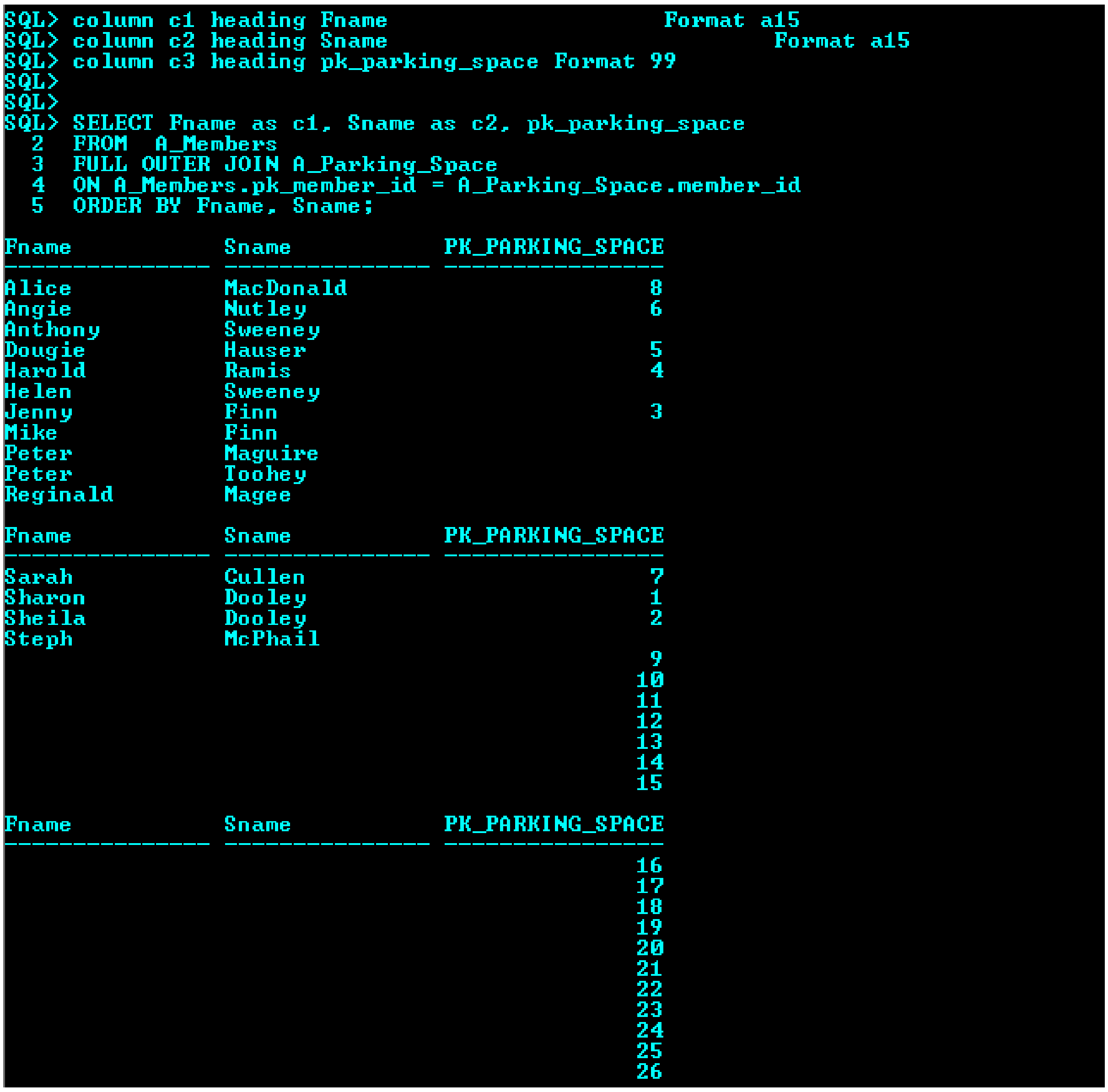
Show all members that have been assigned a car parking space, those that have not and the remaining available spots?

When creating the database I created the 150 car parking spaces so that they will just be assigned as and when they are needed. I will deal with the assignment in a later trigger.

As a result of the number of parking spaces my screenshot cannot show all of the outputted records. Hopefully it will be obvious to you what it does.

*********-------------- For the purpose of some of the queries in this document I have formatted the outputted columns for convenience------------------- *********

```
column c1 heading Fname                 Format a15

column c2 heading Sname                 Format a15

column c3 heading pk_parking_space      Format 99

SELECT Fname as c1, Sname as c2, pk_parking_space

FROM  A_Members

FULL OUTER JOIN A_Parking_Space

ON A_Members.pk_member_id = A_Parking_Space.member_id

ORDER BY Fname, Sname;
```



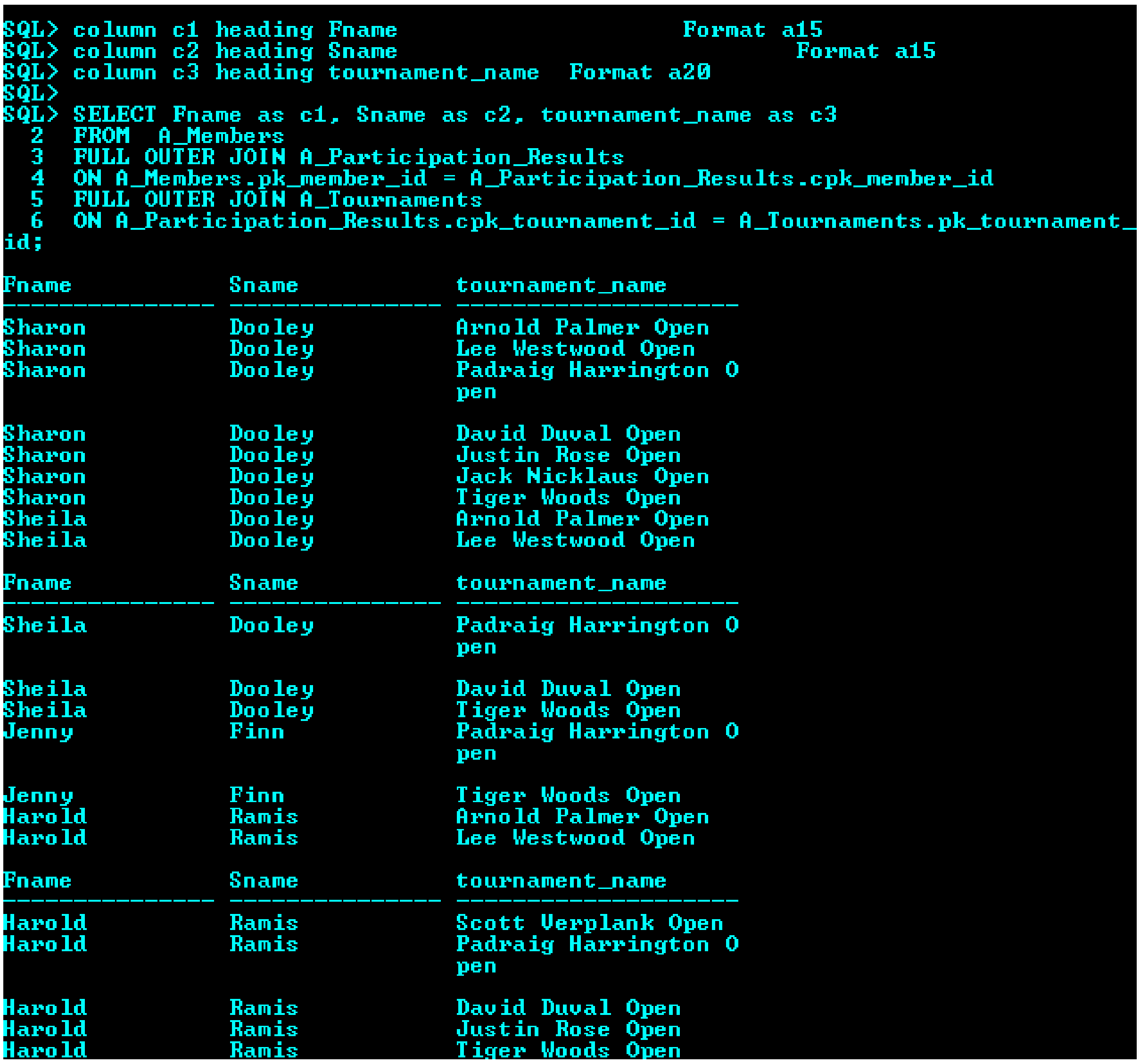
```
SQL> column c1 heading Fname                           Format a15
SQL> column c2 heading Sname                                   Format a15
SQL> column c3 heading pk_parking_space Format 99
SQL>
SQL>
SQL> SELECT Fname as c1, Sname as c2, pk_parking_space
  2  FROM  A_Members
  3  FULL OUTER JOIN A_Parking_Space
  4  ON A_Members.pk_member_id = A_Parking_Space.member_id
  5  ORDER BY Fname, Sname;

Fname           Sname           PK_PARKING_SPACE
--------------- --------------- ----------------
Alice           MacDonald                      8
Angie           Nutley                         6
Anthony         Sweeney
Dougie          Hauser                         5
Harold          Ramis                          4
Helen           Sweeney
Jenny           Finn                           3
Mike            Finn
Peter           Maguire
Peter           Toohey
Reginald        Magee

Fname           Sname           PK_PARKING_SPACE
--------------- --------------- ----------------
Sarah           Cullen                         7
Sharon          Dooley                         1
Sheila          Dooley                         2
Steph           McPhail
                                               9
                                              10
                                              11
                                              12
                                              13
                                              14
                                              15

Fname           Sname           PK_PARKING_SPACE
--------------- --------------- ----------------
                                              16
                                              17
                                              18
                                              19
                                              20
                                              21
                                              22
                                              23
                                              24
                                              25
                                              26
```

Show a list of the tournaments that each member has entered including members that have not entered any?

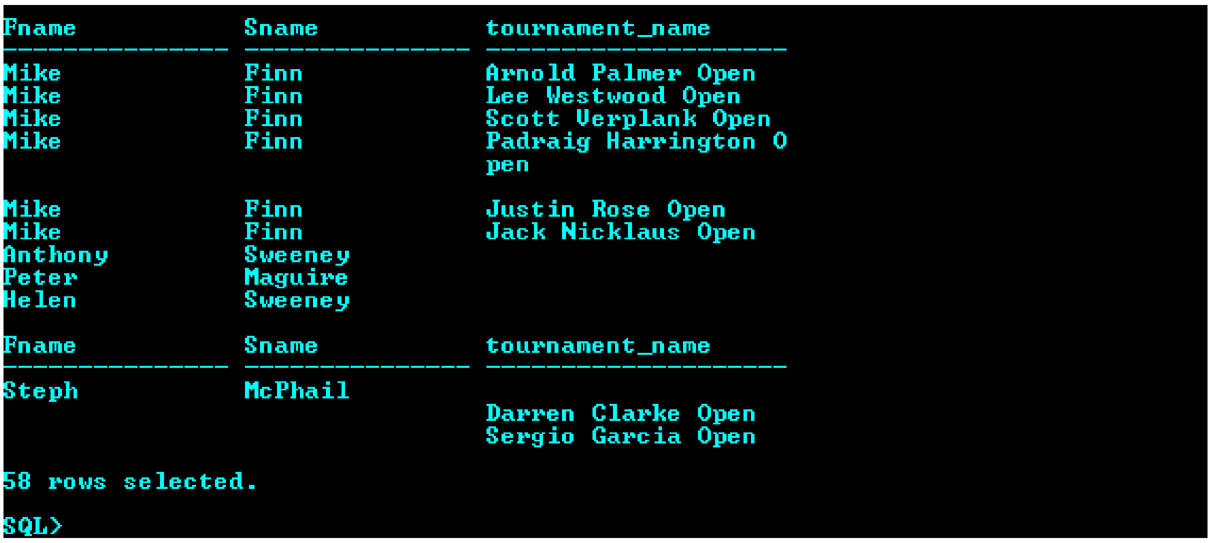
There will be a couple of tournaments with no entrants and a couple of members that have not entered any tournaments

I have given two screenshots as the data outputted is too great for a single one. I have included the start and the end to illustrate the answer

```
column c1 heading Fname                    Format a15

column c2 heading Sname                    Format a15

column c3 heading tournament_name          Format a20

SELECT Fname as c1, Sname as c2, tournament_name as c3

FROM  A_Members

FULL OUTER JOIN A_Participation_Results

ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id

FULL OUTER JOIN A_Tournaments

ON A_Participation_Results.cpk_tournament_id = A_Tournaments.pk_tournament_id;
```



```
SQL> column c1 heading Fname                    Format a15
SQL> column c2 heading Sname                            Format a15
SQL> column c3 heading tournament_name  Format a20
SQL>
SQL> SELECT Fname as c1, Sname as c2, tournament_name as c3
  2  FROM  A_Members
  3  FULL OUTER JOIN A_Participation_Results
  4  ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id
  5  FULL OUTER JOIN A_Tournaments
  6  ON A_Participation_Results.cpk_tournament_id = A_Tournaments.pk_tournament_
id;

Fname           Sname           tournament_name
--------------- --------------- --------------------
Sharon          Dooley          Arnold Palmer Open
Sharon          Dooley          Lee Westwood Open
Sharon          Dooley          Padraig Harrington O
                                pen

Sharon          Dooley          David Duval Open
Sharon          Dooley          Justin Rose Open
Sharon          Dooley          Jack Nicklaus Open
Sharon          Dooley          Tiger Woods Open
Sheila          Dooley          Arnold Palmer Open
Sheila          Dooley          Lee Westwood Open

Fname           Sname           tournament_name
--------------- --------------- --------------------
Sheila          Dooley          Padraig Harrington O
                                pen

Sheila          Dooley          David Duval Open
Sheila          Dooley          Tiger Woods Open
Jenny           Finn            Padraig Harrington O
                                pen

Jenny           Finn            Tiger Woods Open
Harold          Ramis           Arnold Palmer Open
Harold          Ramis           Lee Westwood Open

Fname           Sname           tournament_name
--------------- --------------- --------------------
Harold          Ramis           Scott Verplank Open
Harold          Ramis           Padraig Harrington O
                                pen

Harold          Ramis           David Duval Open
Harold          Ramis           Justin Rose Open
Harold          Ramis           Tiger Woods Open
```



```
Fname           Sname           tournament_name
--------------- --------------- --------------------
Mike            Finn            Arnold Palmer Open
Mike            Finn            Lee Westwood Open
Mike            Finn            Scott Verplank Open
Mike            Finn            Padraig Harrington O
                                pen

Mike            Finn            Justin Rose Open
Mike            Finn            Jack Nicklaus Open
Anthony         Sweeney
Peter           Maguire
Helen           Sweeney

Fname           Sname           tournament_name
--------------- --------------- --------------------
Steph           McPhail
                                Darren Clarke Open
                                Sergio Garcia Open

58 rows selected.

SQL>
```

2 RIGHT OUTER JOINS

How many payments were made with each payment type? Was there any payment types not used?

The record set is sorted by the most popular.

SELECT type_name, count(category_id)

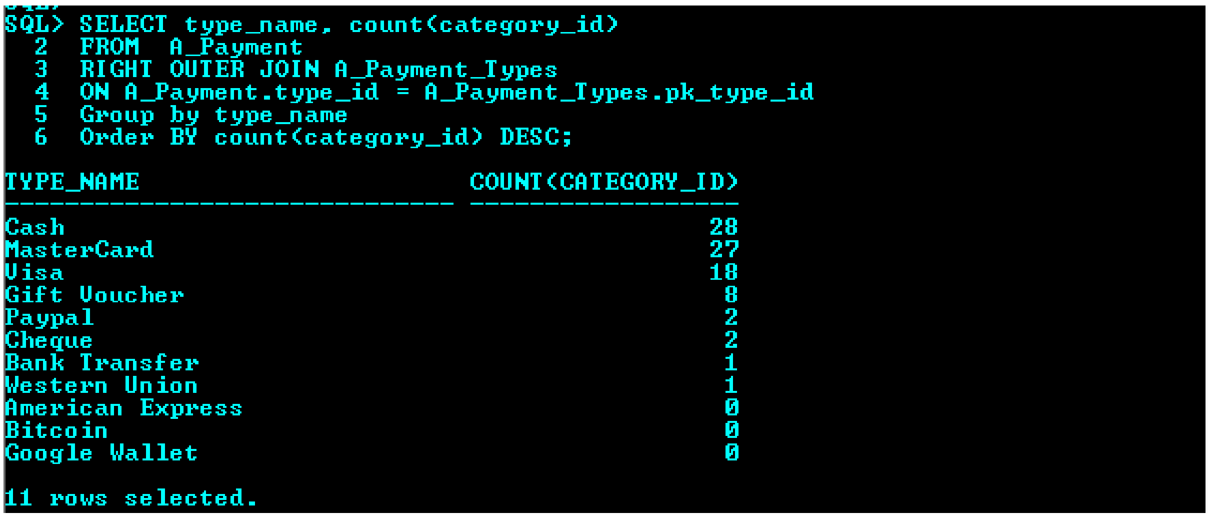
FROM A_Payment

RIGHT OUTER JOIN A_Payment_Types

ON A_Payment.type_id = A_Payment_Types.pk_type_id

Group by type_name

Order BY count(category_id) DESC;



```
SQL> SELECT type_name, count(category_id)
  2  FROM  A_Payment
  3  RIGHT OUTER JOIN A_Payment_Types
  4  ON A_Payment.type_id = A_Payment_Types.pk_type_id
  5  Group by type_name
  6  Order BY count(category_id) DESC;

TYPE_NAME                      COUNT(CATEGORY_ID)
------------------------------ ------------------
Cash                                           28
MasterCard                                     27
Visa                                           18
Gift Voucher                                    8
Paypal                                          2
Cheque                                          2
Bank Transfer                                   1
Western Union                                   1
American Express                                0
Bitcoin                                         0
Google Wallet                                   0

11 rows selected.
```

Give a list of the coaches and their specialties including any specialties that do not have an assigned coach.

In this case Fitness has yet to get an assigned coach.

column c1 heading Fname   Format a15

column c2 heading Sname   Format a15

column c3 heading Specialty   Format a20

SELECT A_Coaches.Fname as c1, A_coaches.Sname as c2, specialty_name as c3

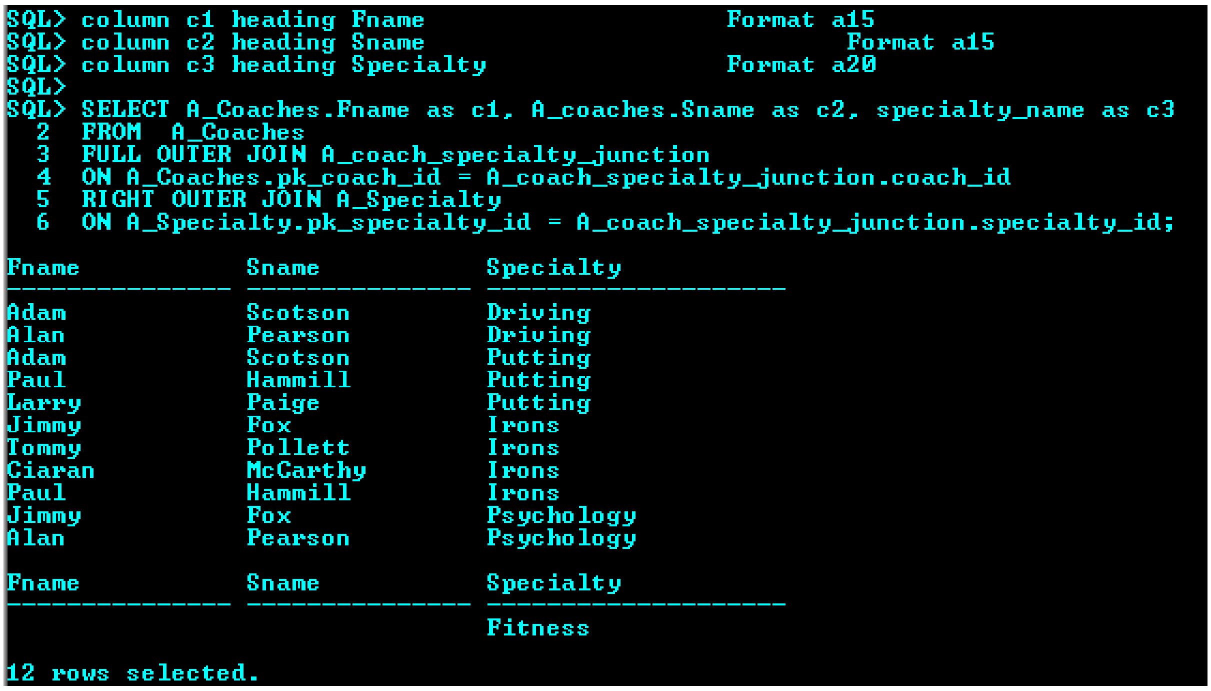
FROM A_Coaches

FULL OUTER JOIN A_coach_specialty_junction

ON A_Coaches.pk_coach_id = A_coach_specialty_junction.coach_id

RIGHT OUTER JOIN A_Specialty

ON A_Specialty.pk_specialty_id = A_coach_specialty_junction.specialty_id;



```
SQL> column c1 heading Fname                        Format a15
SQL> column c2 heading Sname                                Format a15
SQL> column c3 heading Specialty                    Format a20
SQL>
SQL> SELECT A_Coaches.Fname as c1, A_coaches.Sname as c2, specialty_name as c3
  2  FROM  A_Coaches
  3  FULL OUTER JOIN A_coach_specialty_junction
  4  ON A_Coaches.pk_coach_id = A_coach_specialty_junction.coach_id
  5  RIGHT OUTER JOIN A_Specialty
  6  ON A_Specialty.pk_specialty_id = A_coach_specialty_junction.specialty_id;

Fname           Sname           Specialty
--------------- --------------- --------------------
Adam            Scotson         Driving
Alan            Pearson         Driving
Adam            Scotson         Putting
Paul            Hammill         Putting
Larry           Paige           Putting
Jimmy           Fox             Irons
Tommy           Pollett         Irons
Ciaran          McCarthy        Irons
Paul            Hammill         Irons
Jimmy           Fox             Psychology
Alan            Pearson         Psychology

Fname           Sname           Specialty
--------------- --------------- --------------------
                                Fitness

12 rows selected.
```

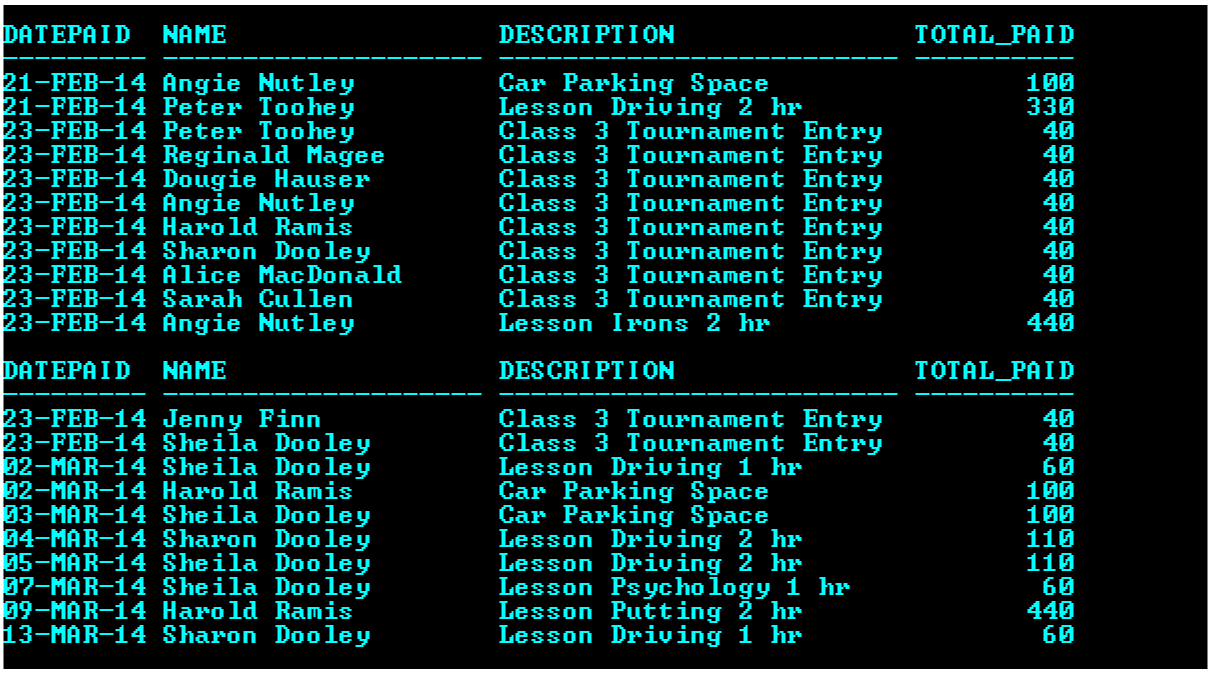
CUBE QUERY WITH AT LEAST TWO COLUMNS

In order to create a meaningful CUBE query I first had to create a VIEW to run it on.

I called this vw_purchase_history and took in data from three tables. After this I performed the cube - which will measure total_paid against each of the attributes requested.

View – the screenshot that follows is a snippet of the view table

```
*/

column Name                Format a20

column Description         Format a25

CREATE OR REPLACE VIEW vw_purchase_history AS

SELECT A_Payment.date_paid as datePaid,

A_Members.Fname||' '||A_Members.Sname AS Name,

A_Payment_Categories.category_name as Description,

(A_Payment_Categories.cost*A_Payment.quantity) as total_paid

FROM A_Payment

INNER JOIN A_Members

ON A_Members.pk_member_id = A_Payment.member_id

INNER JOIN A_Payment_Categories

ON A_Payment.category_id = A_Payment_Categories.pk_category_id;
```


```
DATEPAID  NAME                  DESCRIPTION                TOTAL_PAID
--------- --------------------- -------------------------- ----------
21-FEB-14 Angie Nutley          Car Parking Space                 100
21-FEB-14 Peter Toohey          Lesson Driving 2 hr               330
23-FEB-14 Peter Toohey          Class 3 Tournament Entry           40
23-FEB-14 Reginald Magee        Class 3 Tournament Entry           40
23-FEB-14 Dougie Hauser         Class 3 Tournament Entry           40
23-FEB-14 Angie Nutley          Class 3 Tournament Entry           40
23-FEB-14 Harold Ramis          Class 3 Tournament Entry           40
23-FEB-14 Sharon Dooley         Class 3 Tournament Entry           40
23-FEB-14 Alice MacDonald       Class 3 Tournament Entry           40
23-FEB-14 Sarah Cullen          Class 3 Tournament Entry           40
23-FEB-14 Angie Nutley          Lesson Irons 2 hr                 440

DATEPAID  NAME                  DESCRIPTION                TOTAL_PAID
--------- --------------------- -------------------------- ----------
23-FEB-14 Jenny Finn            Class 3 Tournament Entry           40
23-FEB-14 Sheila Dooley         Class 3 Tournament Entry           40
02-MAR-14 Sheila Dooley         Lesson Driving 1 hr                60
02-MAR-14 Harold Ramis          Car Parking Space                 100
03-MAR-14 Sheila Dooley         Car Parking Space                 100
04-MAR-14 Sharon Dooley         Lesson Driving 2 hr               110
05-MAR-14 Sheila Dooley         Lesson Driving 2 hr               110
07-MAR-14 Sheila Dooley         Lesson Psychology 1 hr             60
09-MAR-14 Harold Ramis          Lesson Putting 2 hr               440
13-MAR-14 Sharon Dooley         Lesson Driving 1 hr                60
```

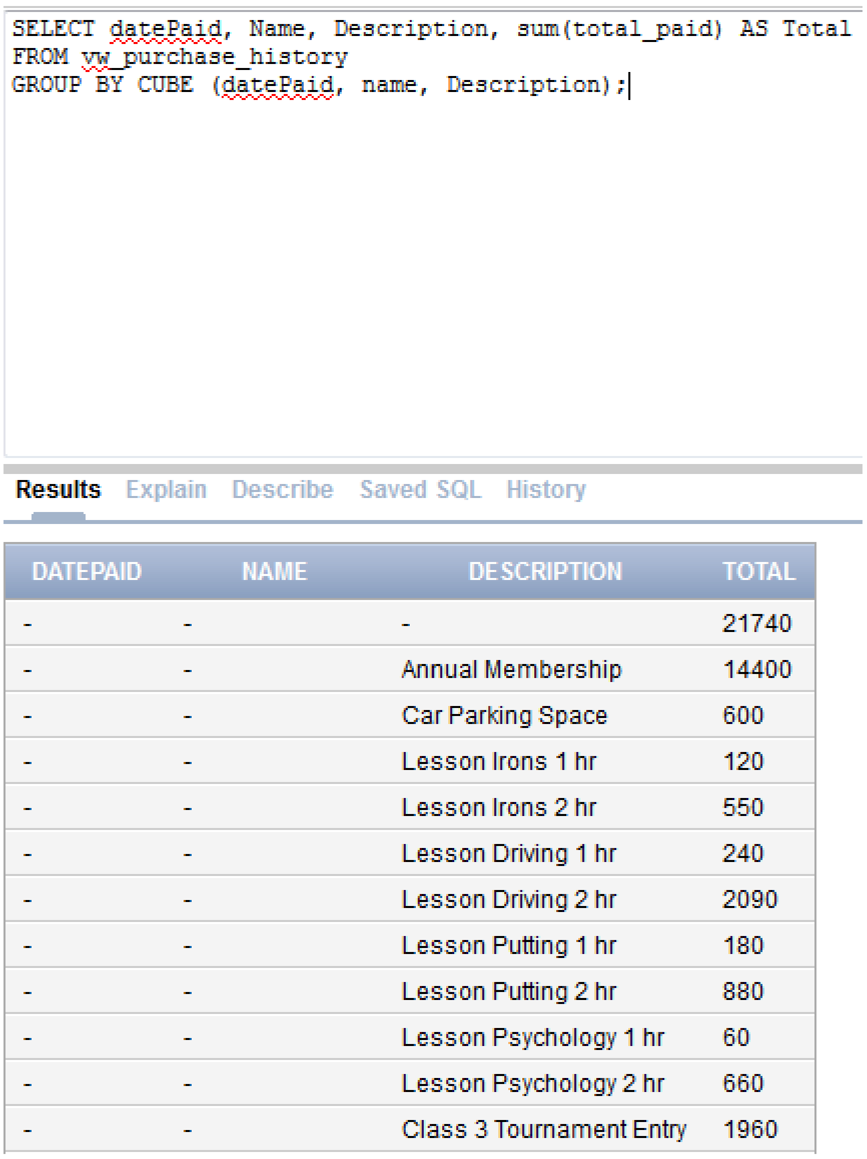
CUBE

The screenshot that follows is a snippet of the cube Query. Again the returned data is far too much to fit into one screenshot so I have provided some screenshots from within the oracle 11g application. I have not provided it all but there should be enough to show that it works properly.

SELECT datePaid, Name, Description, sum(total_paid) AS Total

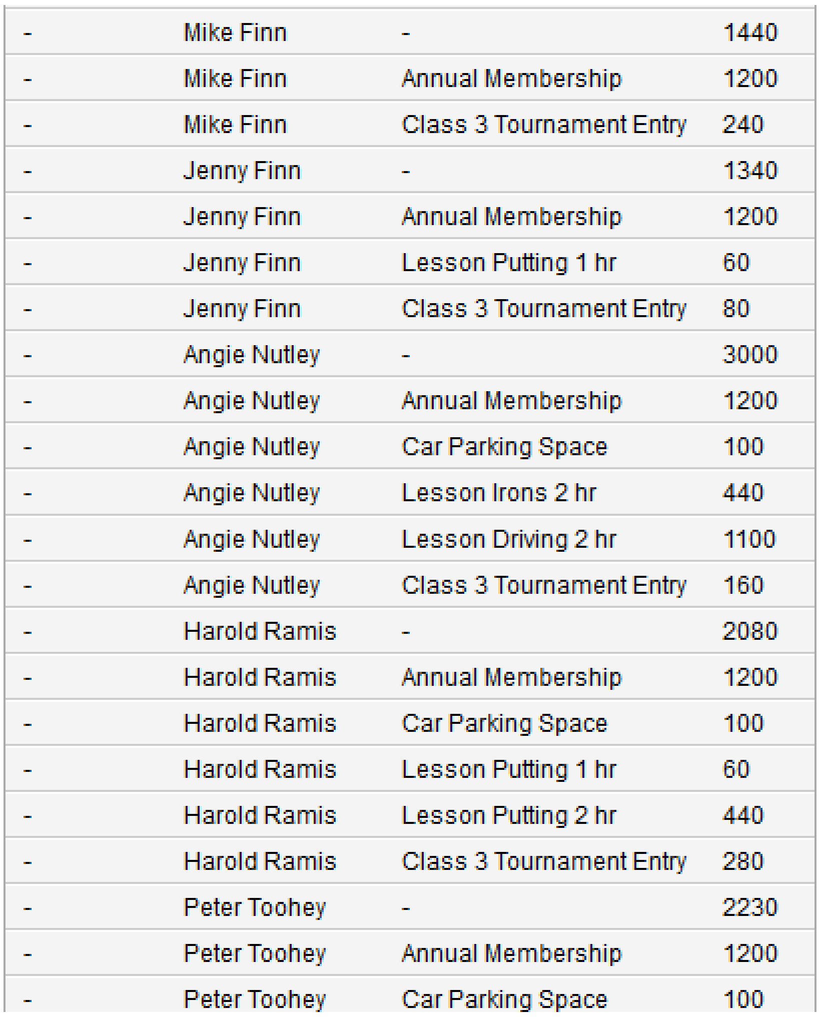
FROM vw_purchase_history

GROUP BY CUBE (datePaid, name, Description);

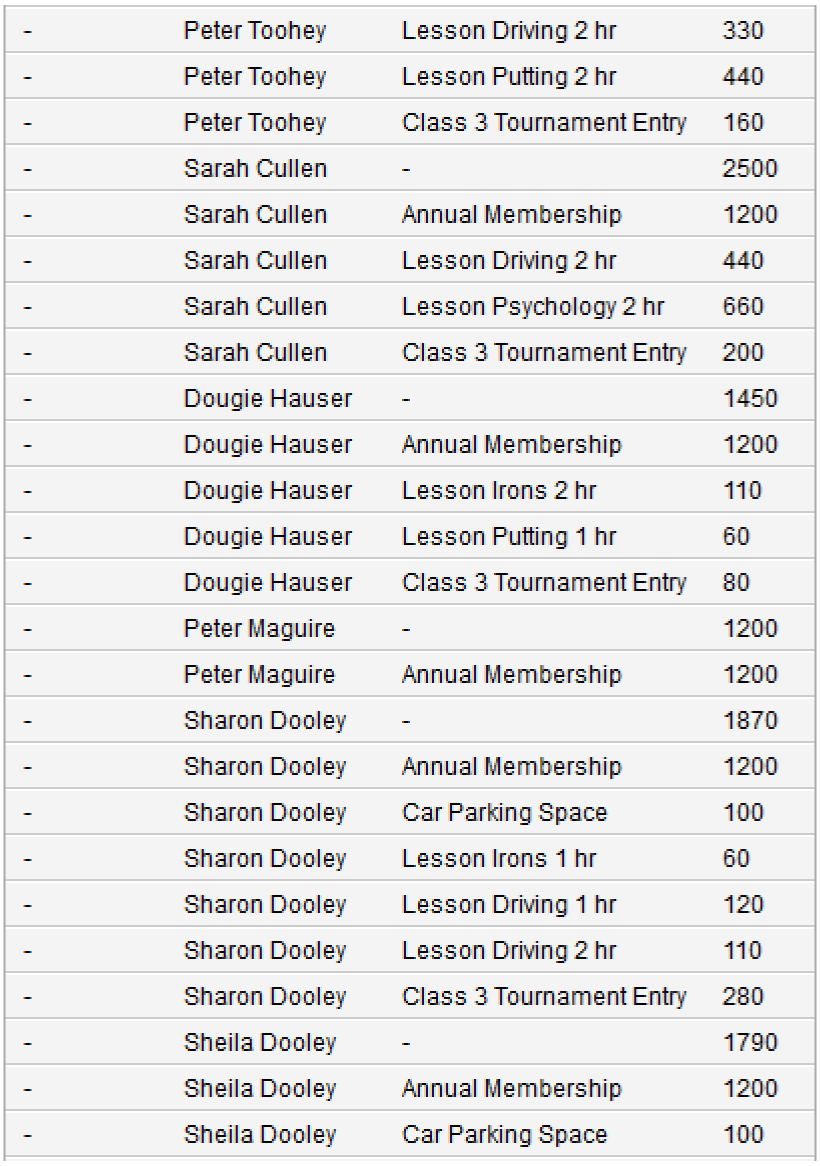


```
SELECT datePaid, Name, Description, sum(total_paid) AS Total
FROM vw_purchase_history
GROUP BY CUBE (datePaid, name, Description);
```

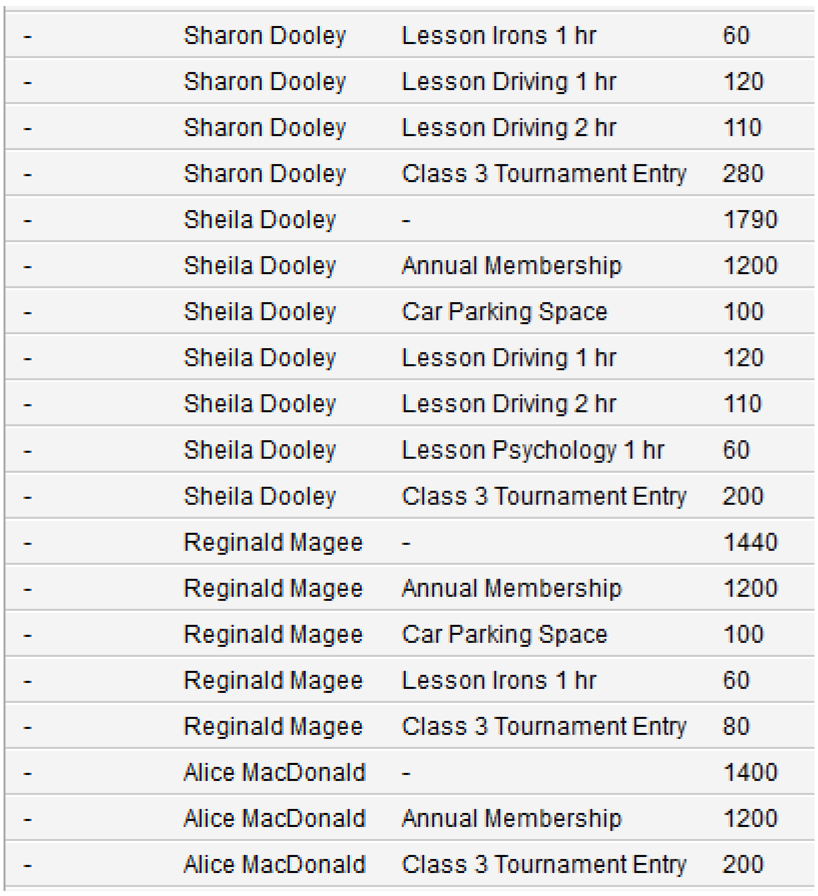
DATEPAID	NAME	DESCRIPTION	TOTAL
-	-	-	21740
-	-	Annual Membership	14400
-	-	Car Parking Space	600
-	-	Lesson Irons 1 hr	120
-	-	Lesson Irons 2 hr	550
-	-	Lesson Driving 1 hr	240
-	-	Lesson Driving 2 hr	2090
-	-	Lesson Putting 1 hr	180
-	-	Lesson Putting 2 hr	880
-	-	Lesson Psychology 1 hr	60
-	-	Lesson Psychology 2 hr	660
-	-	Class 3 Tournament Entry	1960



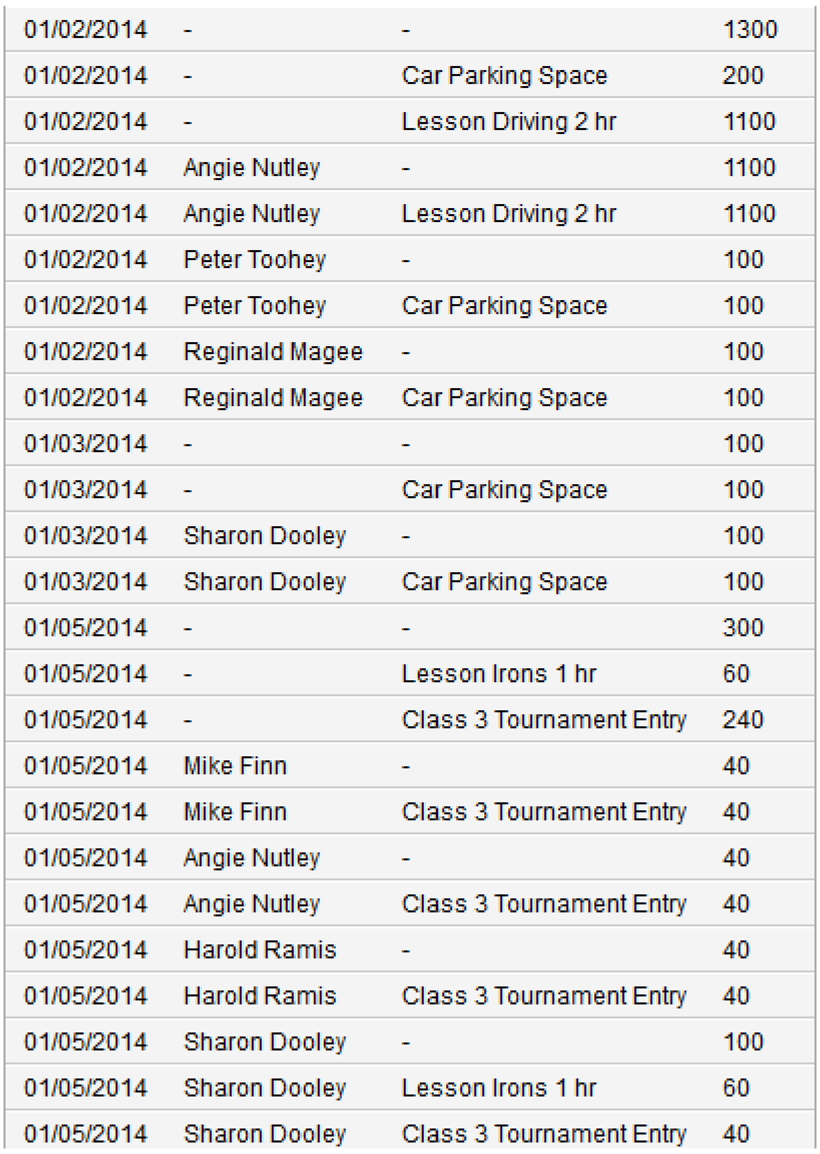
-	Mike Finn	-	1440
-	Mike Finn	Annual Membership	1200
-	Mike Finn	Class 3 Tournament Entry	240
-	Jenny Finn	-	1340
-	Jenny Finn	Annual Membership	1200
-	Jenny Finn	Lesson Putting 1 hr	60
-	Jenny Finn	Class 3 Tournament Entry	80
-	Angie Nutley	-	3000
-	Angie Nutley	Annual Membership	1200
-	Angie Nutley	Car Parking Space	100
-	Angie Nutley	Lesson Irons 2 hr	440
-	Angie Nutley	Lesson Driving 2 hr	1100
-	Angie Nutley	Class 3 Tournament Entry	160
-	Harold Ramis	-	2080
-	Harold Ramis	Annual Membership	1200
-	Harold Ramis	Car Parking Space	100
-	Harold Ramis	Lesson Putting 1 hr	60
-	Harold Ramis	Lesson Putting 2 hr	440
-	Harold Ramis	Class 3 Tournament Entry	280
-	Peter Toohey	-	2230
-	Peter Toohey	Annual Membership	1200
-	Peter Toohey	Car Parking Space	100



-	Peter Toohey	Lesson Driving 2 hr	330
-	Peter Toohey	Lesson Putting 2 hr	440
-	Peter Toohey	Class 3 Tournament Entry	160
-	Sarah Cullen	-	2500
-	Sarah Cullen	Annual Membership	1200
-	Sarah Cullen	Lesson Driving 2 hr	440
-	Sarah Cullen	Lesson Psychology 2 hr	660
-	Sarah Cullen	Class 3 Tournament Entry	200
-	Dougie Hauser	-	1450
-	Dougie Hauser	Annual Membership	1200
-	Dougie Hauser	Lesson Irons 2 hr	110
-	Dougie Hauser	Lesson Putting 1 hr	60
-	Dougie Hauser	Class 3 Tournament Entry	80
-	Peter Maguire	-	1200
-	Peter Maguire	Annual Membership	1200
-	Sharon Dooley	-	1870
-	Sharon Dooley	Annual Membership	1200
-	Sharon Dooley	Car Parking Space	100
-	Sharon Dooley	Lesson Irons 1 hr	60
-	Sharon Dooley	Lesson Driving 1 hr	120
-	Sharon Dooley	Lesson Driving 2 hr	110
-	Sharon Dooley	Class 3 Tournament Entry	280
-	Sheila Dooley	-	1790
-	Sheila Dooley	Annual Membership	1200
-	Sheila Dooley	Car Parking Space	100



-	Sharon Dooley	Lesson Irons 1 hr	60
-	Sharon Dooley	Lesson Driving 1 hr	120
-	Sharon Dooley	Lesson Driving 2 hr	110
-	Sharon Dooley	Class 3 Tournament Entry	280
-	Sheila Dooley	-	1790
-	Sheila Dooley	Annual Membership	1200
-	Sheila Dooley	Car Parking Space	100
-	Sheila Dooley	Lesson Driving 1 hr	120
-	Sheila Dooley	Lesson Driving 2 hr	110
-	Sheila Dooley	Lesson Psychology 1 hr	60
-	Sheila Dooley	Class 3 Tournament Entry	200
-	Reginald Magee	-	1440
-	Reginald Magee	Annual Membership	1200
-	Reginald Magee	Car Parking Space	100
-	Reginald Magee	Lesson Irons 1 hr	60
-	Reginald Magee	Class 3 Tournament Entry	80
-	Alice MacDonald	-	1400
-	Alice MacDonald	Annual Membership	1200
-	Alice MacDonald	Class 3 Tournament Entry	200



01/02/2014	-	-	1300
01/02/2014	-	Car Parking Space	200
01/02/2014	-	Lesson Driving 2 hr	1100
01/02/2014	Angie Nutley	-	1100
01/02/2014	Angie Nutley	Lesson Driving 2 hr	1100
01/02/2014	Peter Toohey	-	100
01/02/2014	Peter Toohey	Car Parking Space	100
01/02/2014	Reginald Magee	-	100
01/02/2014	Reginald Magee	Car Parking Space	100
01/03/2014	-	-	100
01/03/2014	-	Car Parking Space	100
01/03/2014	Sharon Dooley	-	100
01/03/2014	Sharon Dooley	Car Parking Space	100
01/05/2014	-	-	300
01/05/2014	-	Lesson Irons 1 hr	60
01/05/2014	-	Class 3 Tournament Entry	240
01/05/2014	Mike Finn	-	40
01/05/2014	Mike Finn	Class 3 Tournament Entry	40
01/05/2014	Angie Nutley	-	40
01/05/2014	Angie Nutley	Class 3 Tournament Entry	40
01/05/2014	Harold Ramis	-	40
01/05/2014	Harold Ramis	Class 3 Tournament Entry	40
01/05/2014	Sharon Dooley	-	100
01/05/2014	Sharon Dooley	Lesson Irons 1 hr	60
01/05/2014	Sharon Dooley	Class 3 Tournament Entry	40

01/05/2014	Sheila Dooley	-	40
01/05/2014	Sheila Dooley	Class 3 Tournament Entry	40
01/05/2014	Alice MacDonald	-	40
01/05/2014	Alice MacDonald	Class 3 Tournament Entry	40
01/12/2014	-	-	160
01/12/2014	-	Class 3 Tournament Entry	160
01/12/2014	Mike Finn	-	40
01/12/2014	Mike Finn	Class 3 Tournament Entry	40
01/12/2014	Harold Ramis	-	40
01/12/2014	Harold Ramis	Class 3 Tournament Entry	40
01/12/2014	Sharon Dooley	-	40
01/12/2014	Sharon Dooley	Class 3 Tournament Entry	40
01/12/2014	Sheila Dooley	-	40
01/12/2014	Sheila Dooley	Class 3 Tournament Entry	40
01/19/2014	-	-	160
01/19/2014	-	Class 3 Tournament Entry	160
01/19/2014	Mike Finn	-	40
01/19/2014	Mike Finn	Class 3 Tournament Entry	40
01/19/2014	Angie Nutley	-	40
01/19/2014	Angie Nutley	Class 3 Tournament Entry	40
01/19/2014	Harold Ramis	-	40
01/19/2014	Harold Ramis	Class 3 Tournament Entry	40
01/19/2014	Alice MacDonald	-	40

5 SUBQUERIES

I've included six as I had written the 6th before realising I was already done.

Who paid the most for lessons in one purchase and how much did they spend?

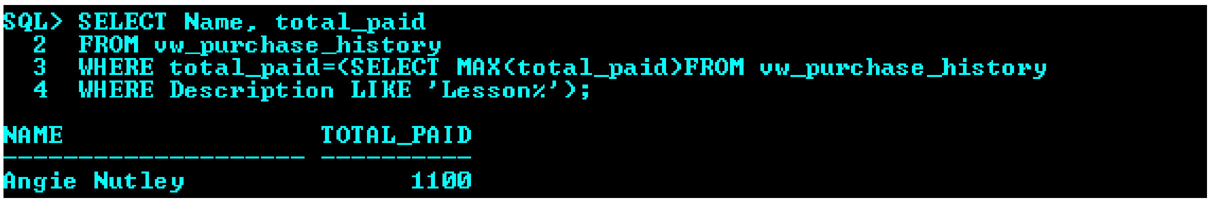
This finds the maximum total paid from the view created earlier where the description had 'Lesson' in it. There are a number of different types of lesson and they all need to be included.

SELECT Name, total_paid

FROM vw_purchase_history

WHERE total_paid = (SELECT MAX(total_paid)FROM vw_purchase_history

WHERE Description LIKE 'Lesson%');



```
SQL> SELECT Name, total_paid
  2  FROM vw_purchase_history
  3  WHERE total_paid=(SELECT MAX(total_paid)FROM vw_purchase_history
  4  WHERE Description LIKE 'Lesson%');

NAME                 TOTAL_PAID
-------------------- ----------
Angie Nutley               1100
```

Which players scored the worst scores in the Tiger Woods Open?

In the sub part of the query I have joined the two tables because tournament name is not stored in A_Participation_Results.

In the first part I am returning the details about the rows select in the sub part

SELECT DISTINCT Fname, Sname, score

FROM A_members

Inner JOIN A_Participation_Results

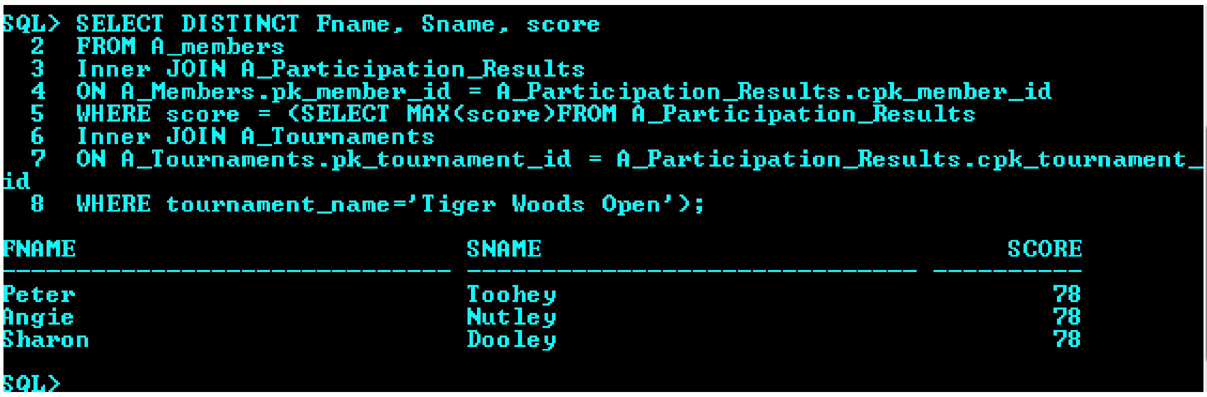
ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id

WHERE score = (SELECT MAX(score)FROM A_Participation_Results

Inner JOIN A_Tournaments

ON A_Tournaments.pk_tournament_id = A_Participation_Results.cpk_tournament_id

WHERE tournament_name='Tiger Woods Open');



```
SQL> SELECT DISTINCT Fname, Sname, score
     FROM A_members
     Inner JOIN A_Participation_Results
     ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id
     WHERE score = (SELECT MAX(score)FROM A_Participation_Results
     Inner JOIN A_Tournaments
     ON A_Tournaments.pk_tournament_id = A_Participation_Results.cpk_tournament_
id
     WHERE tournament_name='Tiger Woods Open');

FNAME                          SNAME                               SCORE
------------------------------ ------------------------------ ----------
Peter                          Toohey                                 78
Angie                          Nutley                                 78
Sharon                         Dooley                                 78

SQL>
```

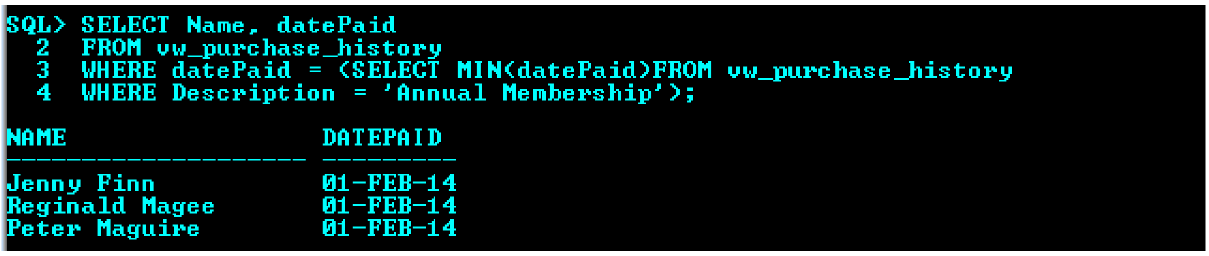
Who were the first members to pay their membership?

SELECT Name, datePaid

FROM vw_purchase_history

WHERE datePaid = (SELECT MIN(datePaid)FROM vw_purchase_history

WHERE Description = 'Annual Membership');



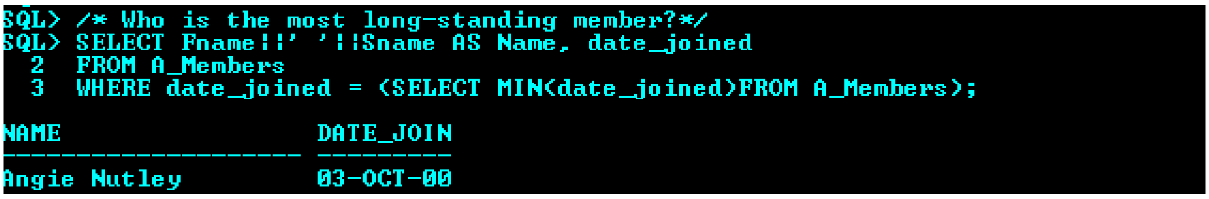
```
SQL> SELECT Name, datePaid
     FROM vw_purchase_history
     WHERE datePaid = (SELECT MIN(datePaid)FROM vw_purchase_history
     WHERE Description = 'Annual Membership');

NAME                     DATEPAID
------------------------ ---------
Jenny Finn               01-FEB-14
Reginald Magee           01-FEB-14
Peter Maguire            01-FEB-14
```

Who is the most long-standing member?

SELECT Fname||' '||Sname AS Name, date_joined

FROM A_Members

WHERE date_joined = (SELECT MIN(date_joined)FROM A_Members);



```
SQL> /* Who is the most long-standing member?*/
SQL> SELECT Fname||' '||Sname AS Name, date_joined
     FROM A_Members
     WHERE date_joined = (SELECT MIN(date_joined)FROM A_Members);

NAME                    DATE_JOIN
----------------------- ---------
Angie Nutley            03-OCT-00
```

Which players have average scores better than the overall average?

As handicaps are already taken to account when compiling all scores this is a good indicator of the form players

SELECT DISTINCT Fname, Sname, avg(score)

FROM A_members

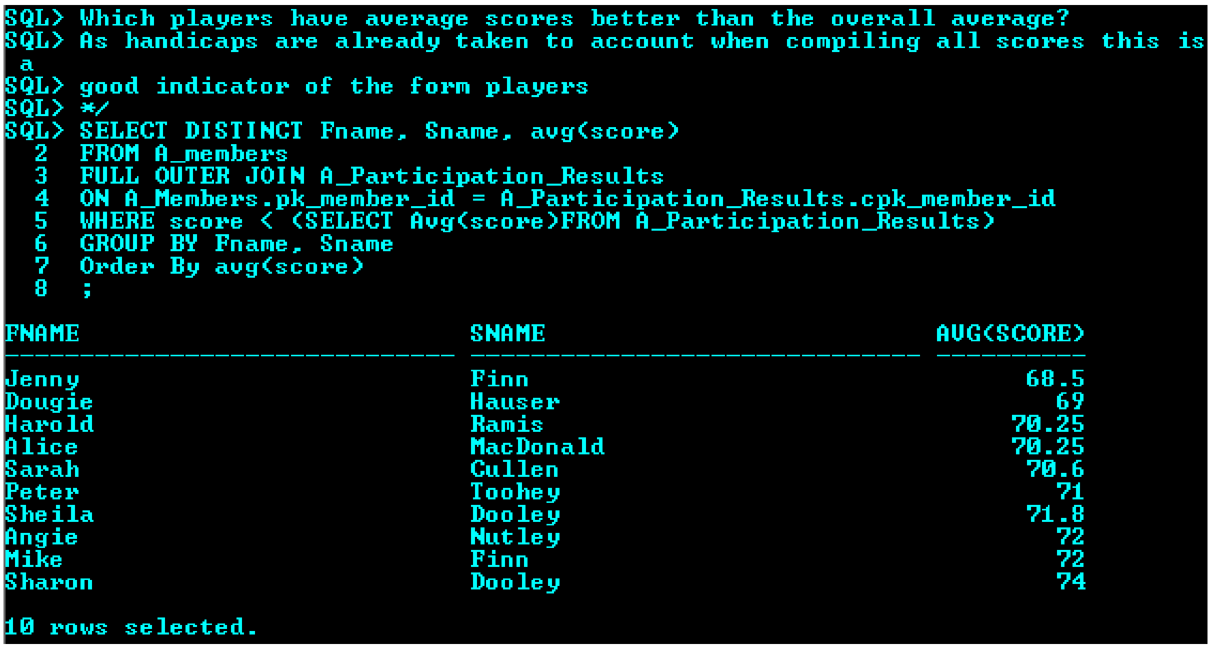
FULL OUTER JOIN A_Participation_Results

ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id

WHERE score < (SELECT Avg(score)FROM A_Participation_Results)

GROUP BY Fname, Sname

Order By avg(score);



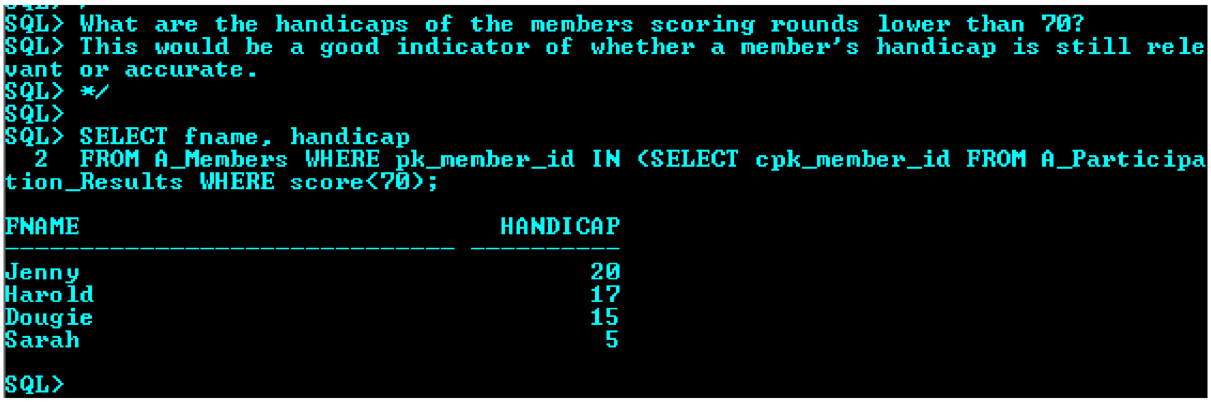
```
SQL> Which players have average scores better than the overall average?
SQL> As handicaps are already taken to account when compiling all scores this is
 a
SQL> good indicator of the form players
SQL> */
SQL> SELECT DISTINCT Fname, Sname, avg(score)
  2  FROM A_members
  3  FULL OUTER JOIN A_Participation_Results
  4  ON A_Members.pk_member_id = A_Participation_Results.cpk_member_id
  5  WHERE score < (SELECT Avg(score)FROM A_Participation_Results)
  6  GROUP BY Fname, Sname
  7  Order By avg(score)
  8  ;

FNAME                          SNAME                          AVG(SCORE)
------------------------------ ------------------------------ ----------
Jenny                          Finn                                 68.5
Dougie                         Hauser                                 69
Harold                         Ramis                               70.25
Alice                          MacDonald                           70.25
Sarah                          Cullen                               70.6
Peter                          Toohey                                 71
Sheila                         Dooley                               71.8
Angie                          Nutley                                 72
Mike                           Finn                                   72
Sharon                         Dooley                                 74

10 rows selected.
```

What are the handicaps of the members scoring rounds lower than 70?

This would be a good indicator of whether a member’s handicap is still relevant or accurate.

SELECT fname, handicap

FROM A_Members WHERE pk_member_id IN (SELECT cpk_member_id FROM A_Participation_Results WHERE score<70);


```
SQL> What are the handicaps of the members scoring rounds lower than 70?
SQL> This would be a good indicator of whether a member's handicap is still rele
vant or accurate.
SQL> */
SQL>
SQL> SELECT fname, handicap
  2  FROM A_Members WHERE pk_member_id IN (SELECT cpk_member_id FROM A_Participa
tion_Results WHERE score<70);

FNAME                            HANDICAP
------------------------------ ----------
Jenny                                  20
Harold                                 17
Dougie                                 15
Sarah                                   5

SQL>
```

5 PL/SQL procedures as part of one package.

Well 7 actually!!! The last two are my first attempts and while they work fine they really should have been functions (they are not changing anything in the database).Each of them illustrates the use of a cursor. I have used savepoint and rollback in the processPayment procedure. I do have 5 proper procedures for marking though. I just thought I would leave in the others for good measure.

I have included the Package project_package first and showed that it worked without errors. After this I have showed each procedure individually and the tests that show that it does what is required.

```
cl scr;

CREATE OR REPLACE PACKAGE project_package AS

PROCEDURE createLesson(member IN NUMBER, duration_in_hours IN NUMBER, datetime_lesson DATE, specialtyID IN NUMBER);

PROCEDURE deleteLesson(lesson IN NUMBER);

PROCEDURE createMember(Fname IN NVARCHAR2, Sname IN NVARCHAR2, address_line_1 IN NVARCHAR2,

address_line_2 IN NVARCHAR2, address_line_3 IN NVARCHAR2, tel_no IN NVARCHAR2);

PROCEDURE deleteMember(member IN NUMBER);

PROCEDURE processPayment(    amountOfItems IN NUMBER,

            member IN NUMBER , paymentType IN NUMBER, category IN NUMBER, lessonDate DATE := NULL);

PROCEDURE ShowMembershipLength;

PROCEDURE TournamentParticipants(tournamentNumber IN NUMBER);

END;

/


CREATE OR REPLACE PACKAGE BODY project_package AS

PROCEDURE createLesson(member IN NUMBER,

duration_in_hours IN NUMBER, datetime_lesson DATE, specialtyID IN NUMBER)

-- the procedure takes in parameters to create the lesson

IS
```

```
CURSOR dateToCheck IS SELECT lessonDate FROM vw_lesson_history

where Specialty = specialtyID;

-- the cursor is created to iterate through all of the currently taken lesson times for that specialty

-- as a way of checking whether the time is available

dateWanted DATE;

-- stores each date in order to check whether it is availabale

timeNotAvail EXCEPTION;

-- user defined exception

coach int;

-- we are not told which coach when someone books a lesson they just

-- enter the specialty that they want. The database needs to find a

-- coach with that specialty and assign them to the lesson.

-- The coach variable stores the coach_id


BEGIN


OPEN dateToCheck;

--open cursor

FETCH dateToCheck INTO dateWanted;

WHILE dateToCheck%FOUND LOOP

-- this iterates through the record set

        if dateWanted = datetime_lesson

                THEN

           raise_application_error(-20010, 'There are no coaches available for this time and date');


           -- if the date we want to book is already taken raise an exception

        end if;
```

```
FETCH dateToCheck INTO dateWanted;

END LOOP;

CLOSE dateToCheck;

--close cursor

-- If the date is available we need to find a coach that specialises in the requested

-- specialty

SELECT coach_id INTO coach FROM A_coach_specialty_junction

WHERE A_coach_specialty_junction.specialty_id = specialtyID AND ROWNUM <=1;


INSERT INTO A_Lessons(pk_lesson_id, duration_in_hours,  coach_id)

VALUES(A_Lessons_sequence.nextval,  duration_in_hours, coach );

-- create the lesson


INSERT INTO A_member_lesson(member_id, lesson_id, datetime_lesson)

VALUES(member, A_Lessons_sequence.currval, datetime_lesson);

--create the instance of the lesson

END createLesson;



PROCEDURE deleteLesson(lesson IN NUMBER)

IS

--do a check to see if the lesson exists

BEGIN

-- needs to be deleted in the following order in order to satisfy key constraints

DELETE FROM A_member_lesson WHERE lesson_id = lesson;

DELETE FROM A_Lessons WHERE pk_lesson_ID = lesson;

END deleteLesson;
```

```
PROCEDURE createMember(Fname IN NVARCHAR2, Sname IN NVARCHAR2, address_line_1 IN
NVARCHAR2,

address_line_2 IN NVARCHAR2, address_line_3 IN NVARCHAR2, tel_no IN NVARCHAR2)

IS

NoFirstName Exception;

-- this is the user defined exception

BEGIN

IF Fname IS NULL

  THEN

    RAISE NoFirstName;

END IF;

INSERT INTO A_Members

VALUES(A_Members_sequence.nextval, Fname, Sname, address_line_1, address_line_2,
address_line_3, tel_no, CURRENT_DATE, Null);

EXCEPTION

  WHEN NoFirstName THEN

            DBMS_OUTPUT.PUT_LINE('You have not entered a first name.');

            -- Prints to the console if the user leaves the first name blank

END createMember;




PROCEDURE deleteMember(member IN NUMBER)

IS

--do a check to see if the lesson exists

BEGIN


-- ******needs to be deleted in the following order in order to satisfy key constraints******
```

```
UPDATE A_Parking_Space SET member_id = NULL WHERE member_id = member;

-- if the member has a car parking space that needs to be unassigned(not deleted as all of the

  --parking spaces are always stored in the database)


DELETE FROM A_member_lesson WHERE member_id = member AND datetime_lesson > (sysdate);

-- this deletes an instance of a  future lesson. Only deletes the future lessons

--as we wish to keep a record of all past lessons


DELETE FROM A_Lessons WHERE pk_lesson_id = (SELECT lesson_id FROM A_member_lesson

WHERE member_id = member AND datetime_lesson > (sysdate));

--deletes the lesson so that the coach is no longer assigned.


DELETE FROM A_Participation_Results

WHERE cpk_member_id = member AND cpk_tournament_id = (SELECT pk_tournament_id FROM
A_Tournaments

WHERE tournament_date > (sysdate));

-- this deletes any possible registration that the member might have for a future tournament.


DELETE FROM A_Members WHERE pk_member_id = member;

--delete the member record in the Members table


END deleteMember;
```

```
PROCEDURE processPayment(    amountOfItems IN NUMBER,

            member IN NUMBER , paymentType IN NUMBER, category IN NUMBER, lessonDate DATE :=
NULL)

--by setting lessonDate DATE := NULL we have made lessonDate an optional parameter

-- this is because when booking a lesson a date is required whilst the other payments do not need a
date.

IS



Fname NVARCHAR2(30);

Sname NVARCHAR2(30);

tournament NUMBER;

duration_in_hours NUMBER;

specialty NUMBER;

space NUMBER;

membership_id NUMBER;

invalidCat EXCEPTION;

--membership_id is required because otherwise

--when we try to insert member into cpk_tournament we get an error as there is a

--possibility of a NUll vale being entered into a primary key field.

BEGIN



savepoint beforeInsert;

-- no field exists to be populated. This could happen if a member tries to pay for a

--tournament that does not exist yet.

-- or if they try to pay for a parking space but none are available.



SELECT A_Members.Fname INTO Fname FROM A_Members WHERE A_Members.pk_member_id =
member;

--stores the members firstname in Fname
```

```
SELECT A_Members.Sname INTO Sname FROM A_Members WHERE A_Members.pk_member_id =
member;

--stores the members firstname in Fname

membership_id := member;


SELECT pk_tournament_id INTO tournament FROM A_Tournaments WHERE (tournament_date -
sysdate) BETWEEN 0 AND 7;

-- This stores the closest future tournament in the variable tournament

-- There is a business rule that their is a tournament every 7 days.


if tournament is NULL then

  tournament :=-1;

end if;


if membership_id is NULL then

  membership_id :=-1;

end if;

-- these are required to avoid the null value being inserted into a the composite primary key for the
table

-- A_Participation_Results


INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_id, category_id)

VALUES(A_Payment_sequence.nextval, sysdate, amountOfItems, member, paymentType, category);

--this creates the payment


CASE category

-- this case asigns the correct duration and specialty based on the correct category_id being entered

-- Nothing needs to occur when category_id = 1 as this membership.
```

```
WHEN 1 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid their membership');


WHEN 2 THEN duration_in_hours := 1; Specialty := 1 ;

createLesson(member, duration_in_hours, lessonDate, specialty);

-- this calls another procedure to create the correct insert statements


WHEN 3 THEN duration_in_hours := 2; Specialty := 1;

createLesson(member, duration_in_hours, lessonDate, specialty);

-- this calls another procedure to create the correct insert statements etc


WHEN 4 THEN duration_in_hours := 1; Specialty :=  2;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 5 THEN duration_in_hours := 2; Specialty :=  2;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 6 THEN duration_in_hours := 1; Specialty :=  3;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 7 THEN duration_in_hours := 2; Specialty :=  3;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 8 THEN duration_in_hours := 1; Specialty :=  4;

createLesson(member, duration_in_hours, lessonDate, specialty);
```

```
WHEN 9 THEN duration_in_hours := 2; Specialty :=  4;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 10 THEN

-- 10 is the category type of a parking space

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a parking space');

SELECT pk_parking_space INTO space FROM A_Parking_Space where member_id IS NULL AND
ROWNUM <=1;

--selects the next available space

-- if all the parking spaces are already assigned the exception below will handle it by outputting the

-- error message and cancelling the payment.

UPDATE A_Parking_Space

SET member_id = member WHERE pk_parking_space = space;



WHEN 11 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);




WHEN 12 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);
```

```
WHEN 13 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);


WHEN 21 THEN duration_in_hours := 1; Specialty :=  5;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 22 THEN duration_in_hours := 2; Specialty :=  5;

createLesson(member, duration_in_hours, lessonDate, specialty);


ELSE DBMS_OUTPUT.PUT_LINE('No such category'); raise_application_error(-20011, 'This is not a
category_id '||SQLCODE||' -ERROR- '||SQLERRM);

END CASE;


EXCEPTION

WHEN NO_DATA_FOUND THEN

dbms_output.put_line('The field you are trying to populate does not exist');

ROLLBACK to beforeInsert;

-- cancel the payment by the member. This will only delete the car parking payment as a member
cannot

--          have more than one assigned parking space

-- and as a result there will only be one payment where category =10.


WHEN invalidCat THEN

raise_application_error (-20006,'You have entered the incorrect category_id.');


END processPayment;
```

```
PROCEDURE ShowMembershipLength

IS

CURSOR memlen IS SELECT * FROM A_members Order by date_joined;

--create cursor

memberRow memlen%ROWTYPE;

membershipLength NUMBER;

date_joined DATE;

-- variables needed

BEGIN

OPEN memlen;

--open the cursor

FETCH memlen INTO memberRow;

--store the information from the cursor in the variable memberRow

WHILE memlen%FOUND LOOP

-- iterate through the record set until there are no more records

date_joined := memberRow.date_joined;

membershipLength := round(MONTHS_BETWEEN(sysdate, date_joined)/12);

--calculate how many whiole years the member has been joined

DBMS_OUTPUT.PUT_LINE('Name: ' ||memberRow.Fname||' '||memberRow.Sname|| '
Membership Length: '||membershipLength);


FETCH memlen INTO memberRow;

END LOOP;

-- close loop

CLOSE memlen;

--close the cursor

EXCEPTION

WHEN OTHERS THEN
```

```
raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '||SQLERRM);

--generic exception to deal which returns the error message associated with the most recently raised error exception

END ShowMembershipLength;




PROCEDURE TournamentParticipants(tournamentNumber IN NUMBER)

IS

CURSOR member IS SELECT Fname, Sname, tournament_name FROM A_members INNER JOIN A_Participation_results

ON A_members.pk_member_id = A_Participation_results.cpk_member_id

INNER JOIN A_Tournaments

ON A_Participation_results.cpk_tournament_id = A_Tournaments.pk_tournament_id

WHERE tournamentNumber = A_Tournaments.pk_tournament_id;

-- creates a cursor from some joined tables.


memberRow member%ROWTYPE;


BEGIN

DBMS_OUTPUT.PUT_LINE('Member' ||'                         '||'Tournament ');

OPEN member;

--open cursor

FETCH member INTO memberRow;

--store the information from the cursor in the variable memberRow


WHILE member%FOUND LOOP

-- iterate through the record set until there are no more records
```

DBMS_OUTPUT.PUT_LINE(memberRow.Fname || ' ' || memberRow.Sname ||'   '|| memberRow.tournament_name);

FETCH member INTO memberRow;

END LOOP;

--end loop

CLOSE member;

--close cursor

EXCEPTION

WHEN OTHERS THEN

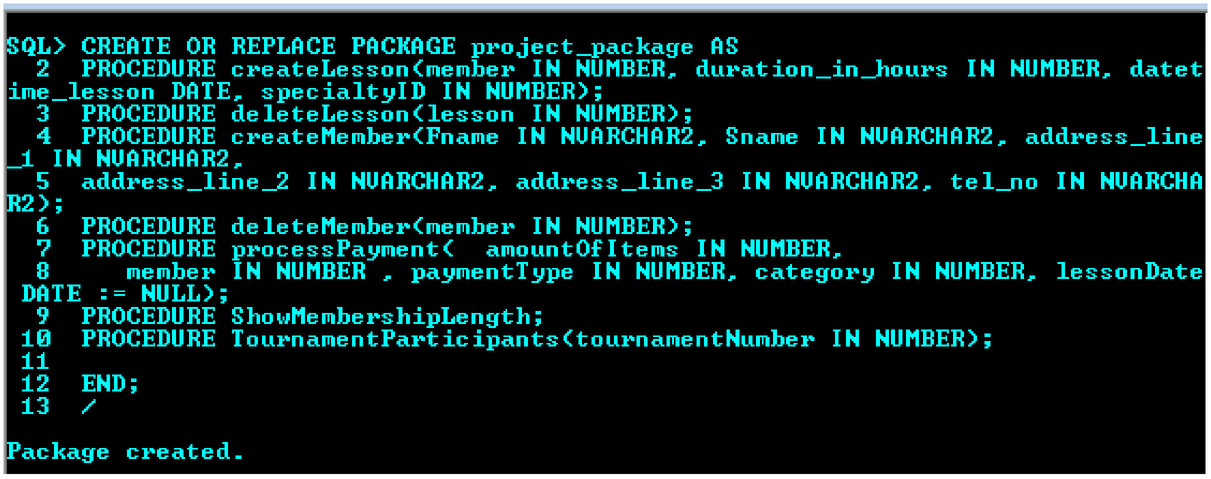
raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '||SQLERRM);

END TournamentParticipants;

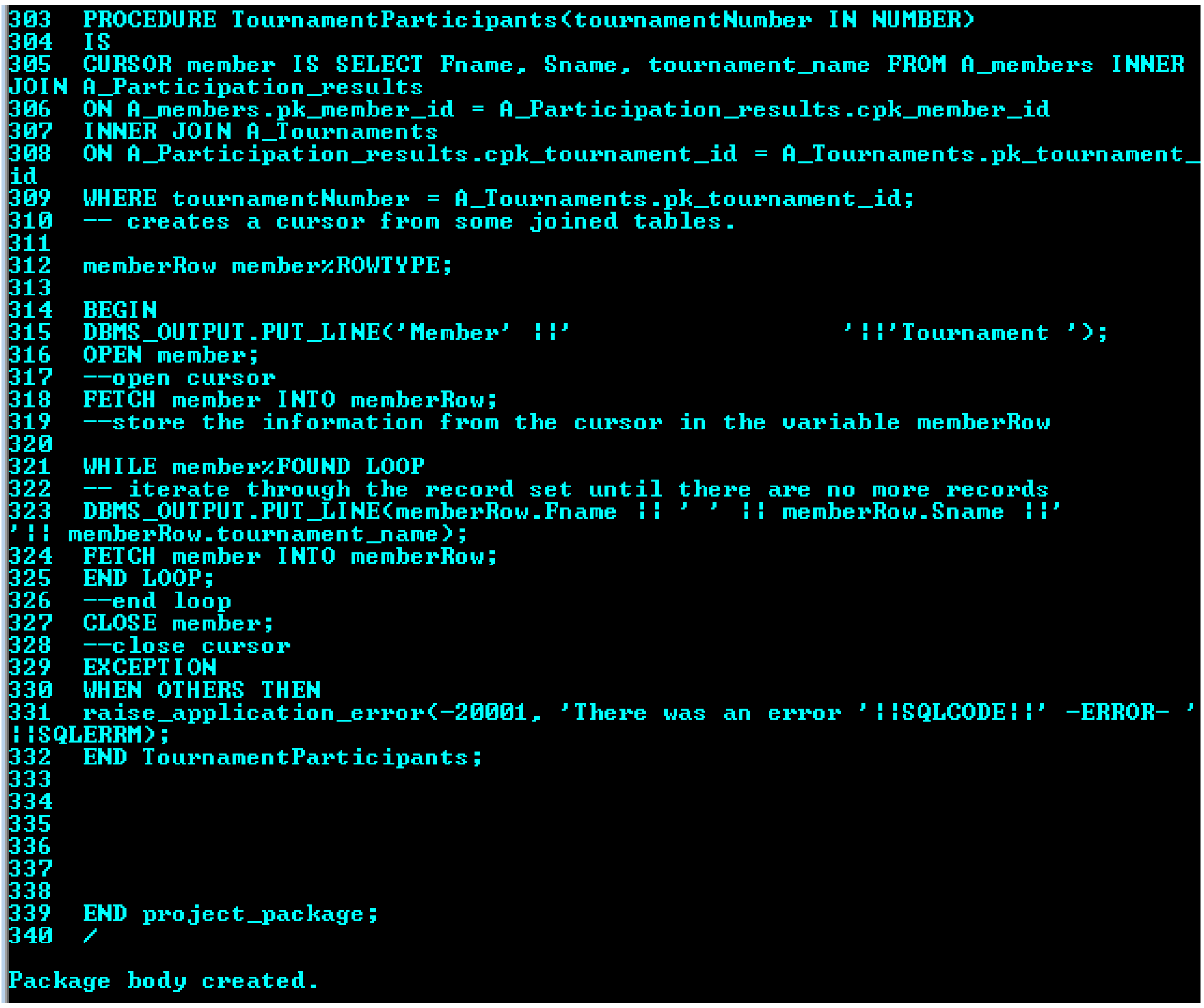
END project_package;

/

Again this is too large to fit in one(or even many screenshots) I have provided the package created screenshot and the package body created screenshot.



```
SQL> CREATE OR REPLACE PACKAGE project_package AS
  2   PROCEDURE createLesson(member IN NUMBER, duration_in_hours IN NUMBER, datet
ime_lesson DATE, specialtyID IN NUMBER);
  3   PROCEDURE deleteLesson(lesson IN NUMBER);
  4   PROCEDURE createMember(Fname IN NVARCHAR2, Sname IN NVARCHAR2, address_line
_1 IN NVARCHAR2,
  5   address_line_2 IN NVARCHAR2, address_line_3 IN NVARCHAR2, tel_no IN NVARCHA
R2);
  6   PROCEDURE deleteMember(member IN NUMBER);
  7   PROCEDURE processPayment(  amountOfItems IN NUMBER,
  8      member IN NUMBER , paymentType IN NUMBER, category IN NUMBER, lessonDate
 DATE := NULL);
  9   PROCEDURE ShowMembershipLength;
 10   PROCEDURE TournamentParticipants(tournamentNumber IN NUMBER);
 11
 12   END;
 13   /

Package created.
```

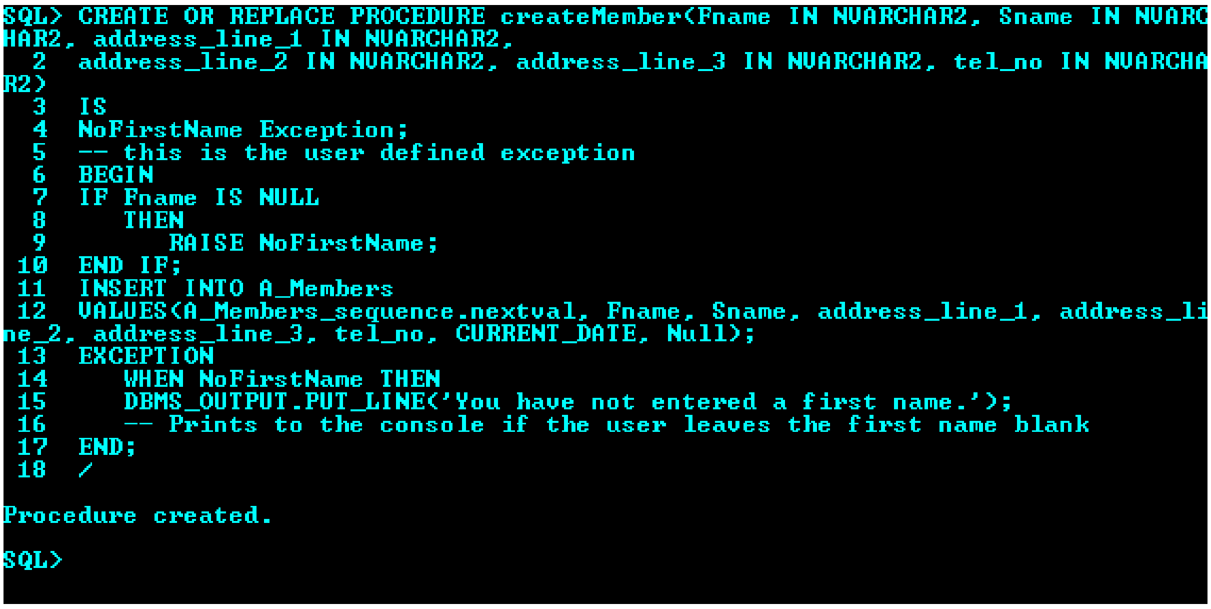


```
PROCEDURE TournamentParticipants(tournamentNumber IN NUMBER)
IS
CURSOR member IS SELECT Fname, Sname, tournament_name FROM A_members INNER JOIN A_Participation_results
ON A_members.pk_member_id = A_Participation_results.cpk_member_id
INNER JOIN A_Tournaments
ON A_Participation_results.cpk_tournament_id = A_Tournaments.pk_tournament_id
WHERE tournamentNumber = A_Tournaments.pk_tournament_id;
-- creates a cursor from some joined tables.

memberRow member%ROWTYPE;

BEGIN
DBMS_OUTPUT.PUT_LINE('Member' ||'                  '||'Tournament ');
OPEN member;
--open cursor
FETCH member INTO memberRow;
--store the information from the cursor in the variable memberRow

WHILE member%FOUND LOOP
-- iterate through the record set until there are no more records
DBMS_OUTPUT.PUT_LINE(memberRow.Fname || ' ' || memberRow.Sname ||'
'|| memberRow.tournament_name);
FETCH member INTO memberRow;
END LOOP;
--end loop
CLOSE member;
--close cursor
EXCEPTION
WHEN OTHERS THEN
raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '||SQLERRM);
END TournamentParticipants;





END project_package;
/

Package body created.
```

Individual Procedure breakdown and Tests.

Procedure 1 createMember

```
CREATE OR REPLACE PROCEDURE createMember(Fname IN NVARCHAR2, Sname IN NVARCHAR2,
address_line_1 IN NVARCHAR2,

address_line_2 IN NVARCHAR2, address_line_3 IN NVARCHAR2, tel_no IN NVARCHAR2)

IS

NoFirstName Exception;

-- this is the user defined exception

BEGIN

IF Fname IS NULL

  THEN

    RAISE NoFirstName;

END IF;

INSERT INTO A_Members

VALUES(A_Members_sequence.nextval, Fname, Sname, address_line_1, address_line_2,
address_line_3, tel_no, CURRENT_DATE, Null);

EXCEPTION

  WHEN NoFirstName THEN

            DBMS_OUTPUT.PUT_LINE('You have not entered a first name.');

            -- Prints to the console if the user leaves the first name blank

END;

/
```



```
SQL> CREATE OR REPLACE PROCEDURE createMember(Fname IN NVARCHAR2, Sname IN NVARC
HAR2, address_line_1 IN NVARCHAR2,
  2  address_line_2 IN NVARCHAR2, address_line_3 IN NVARCHAR2, tel_no IN NVARCHA
R2)
  3  IS
  4  NoFirstName Exception;
  5  -- this is the user defined exception
  6  BEGIN
  7  IF Fname IS NULL
  8     THEN
  9        RAISE NoFirstName;
 10  END IF;
 11  INSERT INTO A_Members
 12  VALUES(A_Members_sequence.nextval, Fname, Sname, address_line_1, address_li
ne_2, address_line_3, tel_no, CURRENT_DATE, Null);
 13  EXCEPTION
 14     WHEN NoFirstName THEN
 15     DBMS_OUTPUT.PUT_LINE('You have not entered a first name.');
 16     -- Prints to the console if the user leaves the first name blank
 17  END;
 18  /

Procedure created.

SQL>
```

/*

Below are tests to test the results of the createMember procedure.

*/

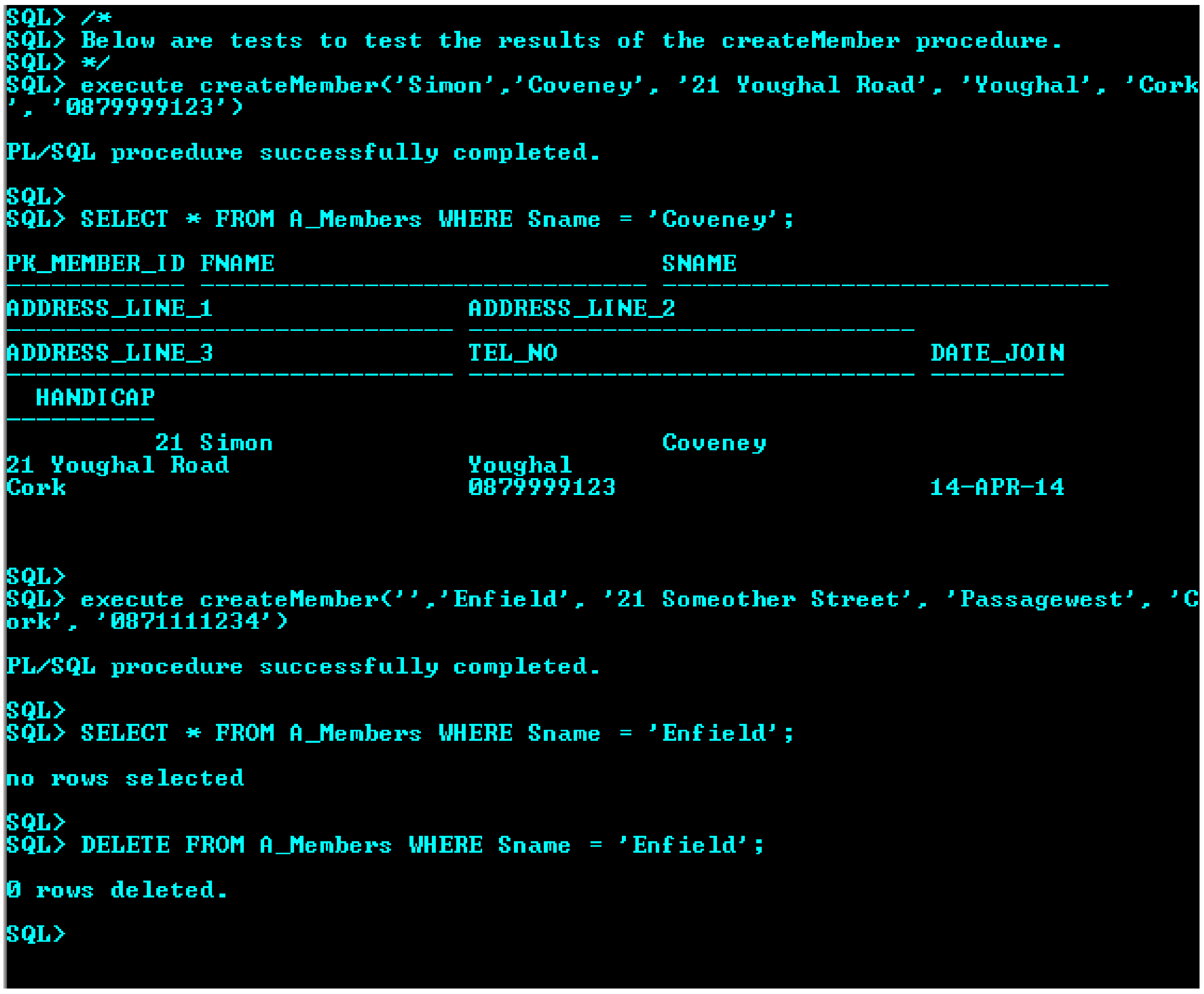
execute createMember('Simon','Coveney', '21 Youghal Road', 'Youghal', 'Cork', '0879999123')

SELECT * FROM A_Members WHERE Sname = 'Coveney';

execute createMember('','Enfield', '21 Someother Street', 'Passagewest', 'Cork', '0871111234')

SELECT * FROM A_Members WHERE Sname = 'Enfield';

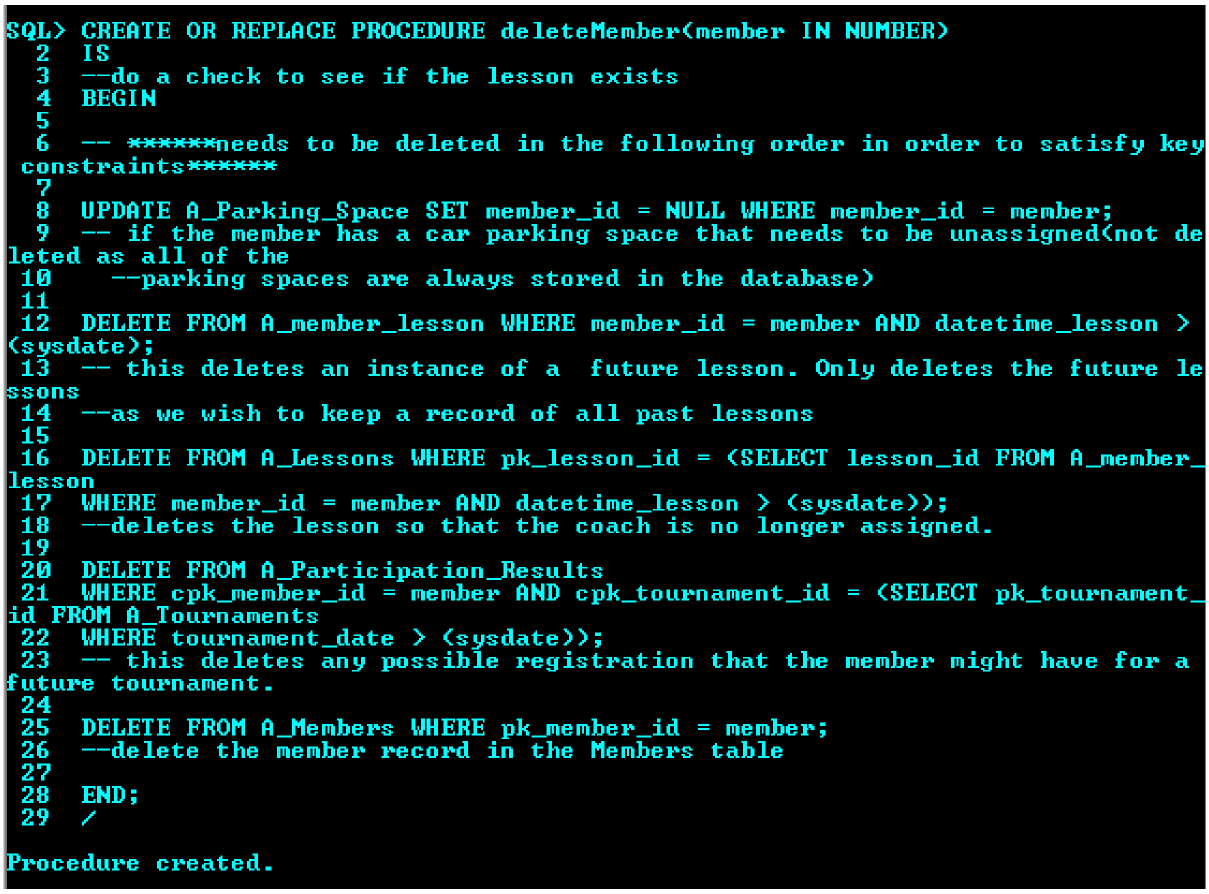
DELETE FROM A_Members WHERE Sname = 'Enfield';



```
SQL> /*
SQL> Below are tests to test the results of the createMember procedure.
SQL> */
SQL> execute createMember('Simon','Coveney', '21 Youghal Road', 'Youghal', 'Cork
', '0879999123')

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT * FROM A_Members WHERE Sname = 'Coveney';

PK_MEMBER_ID FNAME                          SNAME
------------ ------------------------------ ------------------------------
ADDRESS_LINE_1                 ADDRESS_LINE_2
------------------------------ ------------------------------
ADDRESS_LINE_3                 TEL_NO                         DATE_JOIN
------------------------------ ------------------------------ ---------
  HANDICAP
----------
          21 Simon                          Coveney
21 Youghal Road                Youghal
Cork                           0879999123                     14-APR-14



SQL>
SQL> execute createMember('','Enfield', '21 Someother Street', 'Passagewest', 'C
ork', '0871111234')

PL/SQL procedure successfully completed.

SQL>
SQL> SELECT * FROM A_Members WHERE Sname = 'Enfield';

no rows selected

SQL>
SQL> DELETE FROM A_Members WHERE Sname = 'Enfield';

0 rows deleted.

SQL>
```

Procedure 2 deleteMember

```
CREATE OR REPLACE PROCEDURE deleteMember(member IN NUMBER)

IS

--do a check to see if the lesson exists

BEGIN


-- ******needs to be deleted in the following order in order to satisfy key constraints******


UPDATE A_Parking_Space SET member_id = NULL WHERE member_id = member;

-- if the member has a car parking space that needs to be unassigned(not deleted as all of the

  --parking spaces are always stored in the database)

DELETE FROM A_member_lesson WHERE member_id = member AND datetime_lesson > (sysdate);

-- this deletes an instance of a  future lesson. Only deletes the future lessons

--as we wish to keep a record of all past lessons

DELETE FROM A_Lessons WHERE pk_lesson_id = (SELECT lesson_id FROM A_member_lesson

WHERE member_id = member AND datetime_lesson > (sysdate));

--deletes the lesson so that the coach is no longer assigned.

DELETE FROM A_Participation_Results

WHERE cpk_member_id = member AND cpk_tournament_id = (SELECT pk_tournament_id FROM
A_Tournaments

WHERE tournament_date > (sysdate));

-- this deletes any possible registration that the member might have for a future tournament.

DELETE FROM A_Members WHERE pk_member_id = member;

--delete the member record in the Members table

END;

/
```



```
SQL> CREATE OR REPLACE PROCEDURE deleteMember(member IN NUMBER)
IS
--do a check to see if the lesson exists
BEGIN

-- ******needs to be deleted in the following order in order to satisfy key constraints******

UPDATE A_Parking_Space SET member_id = NULL WHERE member_id = member;
-- if the member has a car parking space that needs to be unassigned(not deleted as all of the
  --parking spaces are always stored in the database)

DELETE FROM A_member_lesson WHERE member_id = member AND datetime_lesson > (sysdate);
-- this deletes an instance of a  future lesson. Only deletes the future lessons
--as we wish to keep a record of all past lessons

DELETE FROM A_Lessons WHERE pk_lesson_id = (SELECT lesson_id FROM A_member_lesson
WHERE member_id = member AND datetime_lesson > (sysdate));
--deletes the lesson so that the coach is no longer assigned.

DELETE FROM A_Participation_Results
WHERE cpk_member_id = member AND cpk_tournament_id = (SELECT pk_tournament_id FROM A_Tournaments
WHERE tournament_date > (sysdate));
-- this deletes any possible registration that the member might have for a future tournament.

DELETE FROM A_Members WHERE pk_member_id = member;
--delete the member record in the Members table

END;
/

Procedure created.
```

-- to ensure this works I created a member and registered him to a

--future tournament and future lessons

execute createMember('Simon','Coveney', '21 Youghal Road', 'Youghal', 'Cork', '0879999123')

--this creates the member Simon Coveney

The procedure used for the next test is createLesson which is Procedure 3. I have just used it here as it is easier than going through the whole process of creating a lesson

execute createLesson(A_Members_sequence.currval, 1, TO_DATE('2014/06/06 18:00:00','yyyy/mm/dd hh24:mi:ss'), 1)

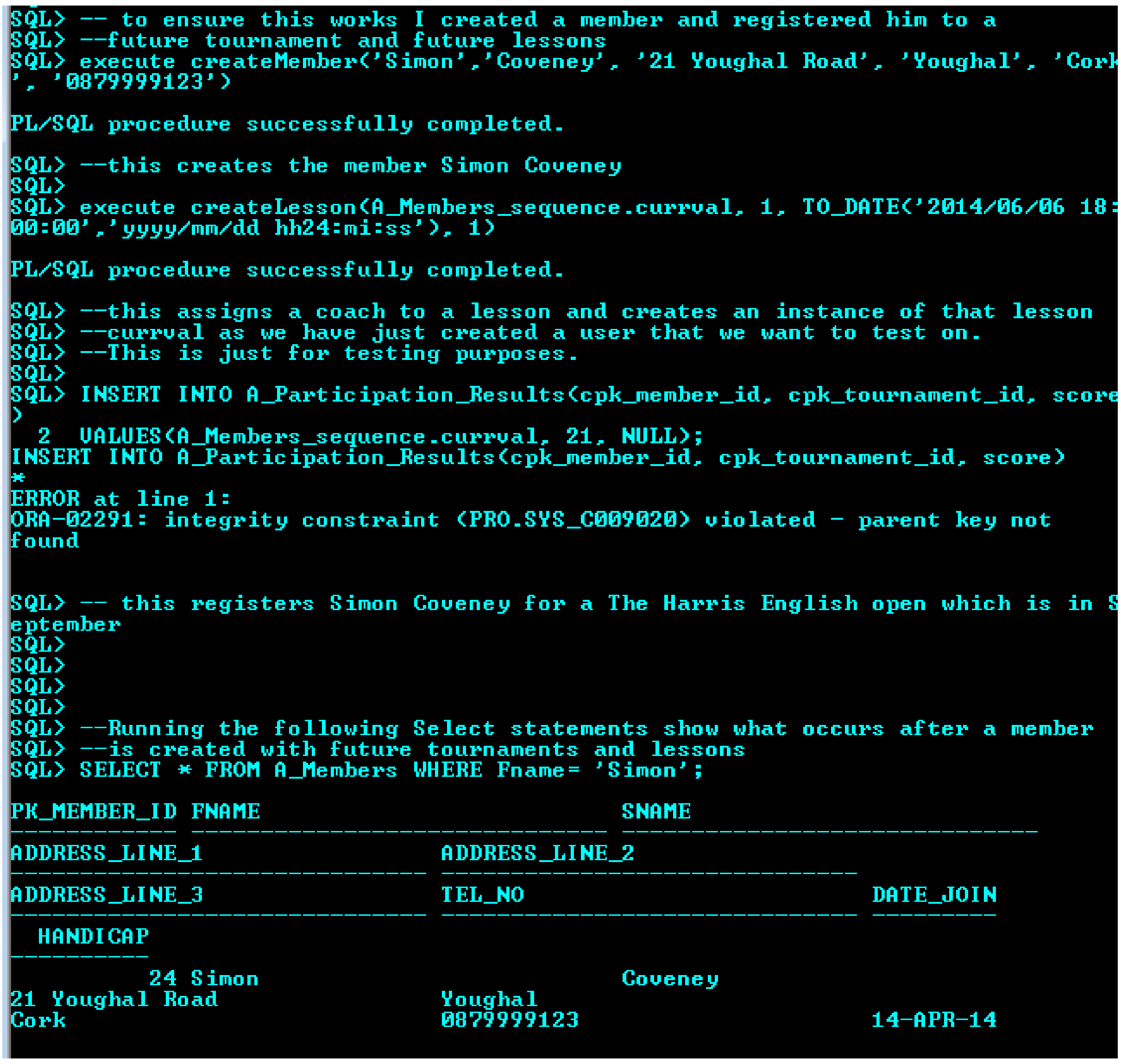
--this assigns a coach to a lesson and creates an instance of that lesson

--currval as we have just created a user that we want to test on.

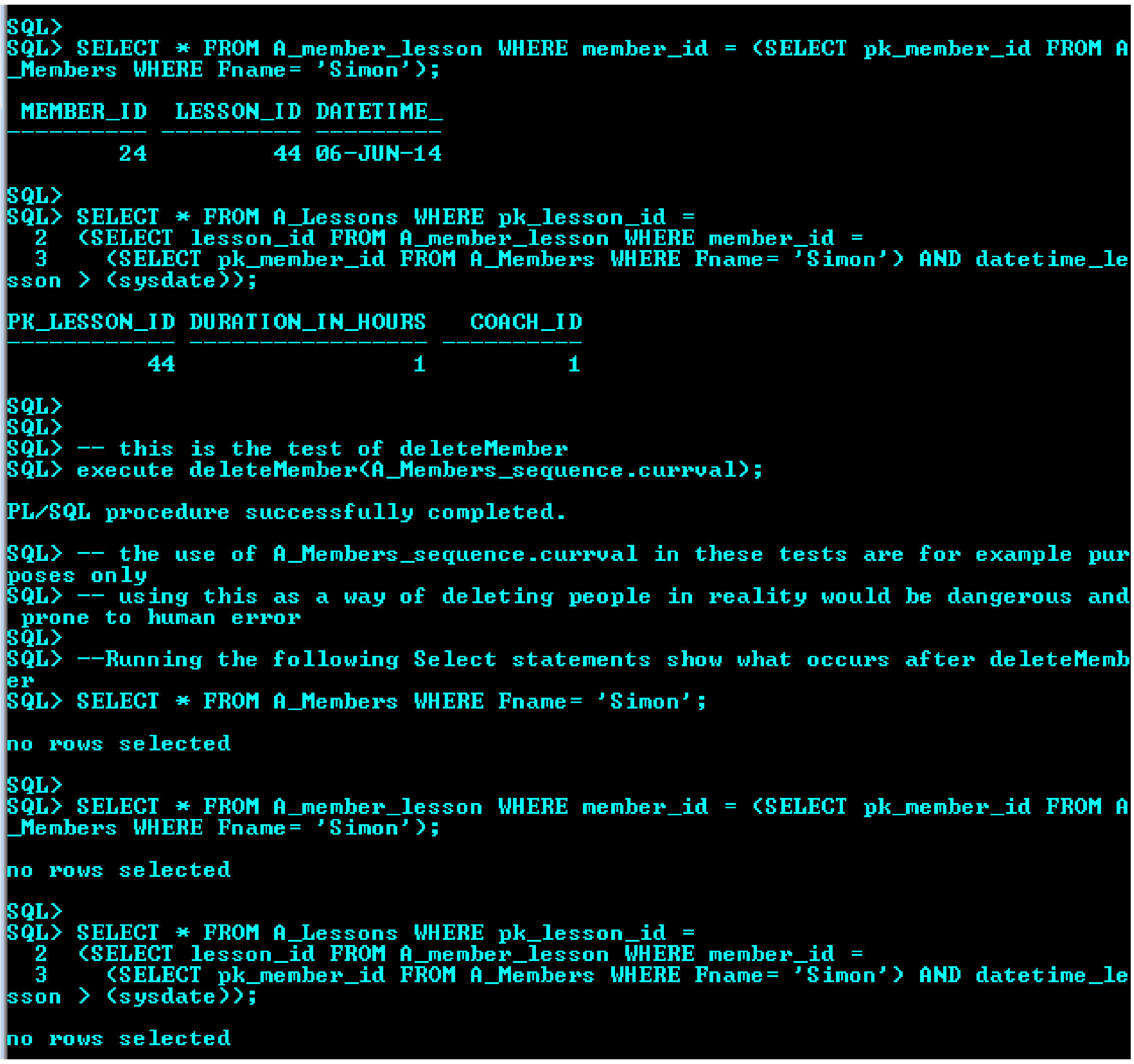
--This is just for testing purposes.

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

```
VALUES(A_Members_sequence.currval, 21, NULL);

-- this registers Simon Coveney for a The Harris English open which is in September

--Running the following Select statements show what occurs after a member

--is created with future tournaments and lessons

SELECT * FROM A_Members WHERE Fname= 'Simon';


SELECT * FROM A_member_lesson WHERE member_id = (SELECT pk_member_id FROM A_Members
WHERE Fname= 'Simon');


SELECT * FROM A_Lessons WHERE pk_lesson_id =

(SELECT lesson_id FROM A_member_lesson WHERE member_id =

  (SELECT pk_member_id FROM A_Members WHERE Fname= 'Simon') AND datetime_lesson >
(sysdate));


-- this is the test of deleteMember

execute deleteMember(A_Members_sequence.currval);

-- the use of A_Members_sequence.currval in these tests are for example purposes only

-- using this as a way of deleting people in reality would be dangerous and prone to human error


--Running the following Select statements show what occurs after deleteMember

SELECT * FROM A_Members WHERE Fname= 'Simon';


SELECT * FROM A_member_lesson WHERE member_id = (SELECT pk_member_id FROM A_Members
WHERE Fname= 'Simon');


SELECT * FROM A_Lessons WHERE pk_lesson_id =

(SELECT lesson_id FROM A_member_lesson WHERE member_id =

  (SELECT pk_member_id FROM A_Members WHERE Fname= 'Simon') AND datetime_lesson >
(sysdate));
```



```
SQL> -- to ensure this works I created a member and registered him to a
SQL> --future tournament and future lessons
SQL> execute createMember('Simon','Coveney', '21 Youghal Road', 'Youghal', 'Cork
', '0879999123')

PL/SQL procedure successfully completed.

SQL> --this creates the member Simon Coveney
SQL>
SQL> execute createLesson(A_Members_sequence.currval, 1, TO_DATE('2014/06/06 18:
00:00','yyyy/mm/dd hh24:mi:ss'), 1)

PL/SQL procedure successfully completed.

SQL> --this assigns a coach to a lesson and creates an instance of that lesson
SQL> --currval as we have just created a user that we want to test on.
SQL> --This is just for testing purposes.
SQL>
SQL> INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score
)
  2  VALUES(A_Members_sequence.currval, 21, NULL);
INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)
*
ERROR at line 1:
ORA-02291: integrity constraint (PRO.SYS_C009020) violated - parent key not
found


SQL> -- this registers Simon Coveney for a The Harris English open which is in S
eptember
SQL>
SQL>
SQL>
SQL>
SQL> --Running the following Select statements show what occurs after a member
SQL> --is created with future tournaments and lessons
SQL> SELECT * FROM A_Members WHERE Fname= 'Simon';

PK_MEMBER_ID FNAME                          SNAME
------------ ------------------------------ ------------------------------
ADDRESS_LINE_1                 ADDRESS_LINE_2
------------------------------ ------------------------------
ADDRESS_LINE_3                 TEL_NO                         DATE_JOIN
------------------------------ ------------------------------ ---------
  HANDICAP
----------
          24 Simon                          Coveney
21 Youghal Road                Youghal
Cork                           0879999123                     14-APR-14
```



```
SQL>
SQL> SELECT * FROM A_member_lesson WHERE member_id = (SELECT pk_member_id FROM A
_Members WHERE Fname= 'Simon');

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
        24         44 06-JUN-14

SQL>
SQL> SELECT * FROM A_Lessons WHERE pk_lesson_id =
  2  (SELECT lesson_id FROM A_member_lesson WHERE member_id =
  3    (SELECT pk_member_id FROM A_Members WHERE Fname= 'Simon') AND datetime_le
sson > (sysdate));

PK_LESSON_ID DURATION_IN_HOURS   COACH_ID
------------ ----------------- ----------
          44                 1          1

SQL>
SQL>
SQL> -- this is the test of deleteMember
SQL> execute deleteMember(A_Members_sequence.currval);

PL/SQL procedure successfully completed.

SQL> -- the use of A_Members_sequence.currval in these tests are for example pur
poses only
SQL> -- using this as a way of deleting people in reality would be dangerous and
 prone to human error
SQL>
SQL> --Running the following Select statements show what occurs after deleteMemb
er
SQL> SELECT * FROM A_Members WHERE Fname= 'Simon';

no rows selected

SQL>
SQL> SELECT * FROM A_member_lesson WHERE member_id = (SELECT pk_member_id FROM A
_Members WHERE Fname= 'Simon');

no rows selected

SQL>
SQL> SELECT * FROM A_Lessons WHERE pk_lesson_id =
  2  (SELECT lesson_id FROM A_member_lesson WHERE member_id =
  3    (SELECT pk_member_id FROM A_Members WHERE Fname= 'Simon') AND datetime_le
sson > (sysdate));

no rows selected
```

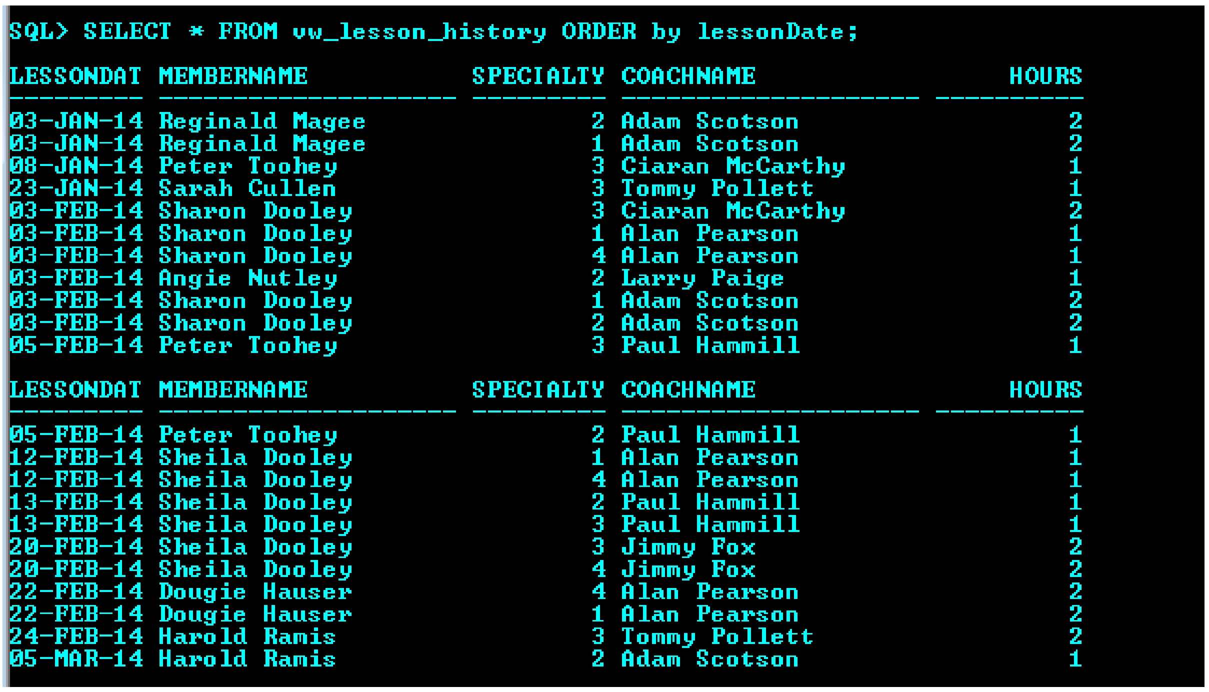
View Creation for Procedure 3

My third procedure 'createLesson' is working over multiple tables and as such it was easier to create a view and have it work off that. The view below shows a summary of all of the lessons and their related information.

```
column memberName        Format a20

column coachName         Format a20

column Specialty         Format 99


cl scr;

CREATE OR REPLACE VIEW vw_lesson_history AS

SELECT A_member_lesson.datetime_lesson as lessonDate,

A_Members.Fname||' '||A_Members.Sname AS memberName,

A_coach_specialty_junction.specialty_id AS Specialty,

A_Coaches.Fname ||' '|| A_Coaches.Sname AS coachName,

A_Lessons.duration_in_hours hours

FROM A_Members

INNER JOIN A_member_lesson

ON A_Members.pk_member_id = A_member_lesson.member_id

INNER JOIN A_Lessons

ON A_member_lesson.lesson_id = A_Lessons.pk_lesson_id

INNER JOIN A_Coaches

ON A_Coaches.pk_coach_id = A_Lessons.coach_id

INNER JOIN A_coach_specialty_junction

ON A_Coaches.pk_coach_id = A_coach_specialty_junction.coach_id

--INNER JOIN A_Specialty

--ON A_Specialty.pk_specialty_id = A_coach_specialty_junction.specialty_id

;
```

SELECT * FROM vw_lesson_history ORDER by lessonDate;

--Shows the lessonhistory

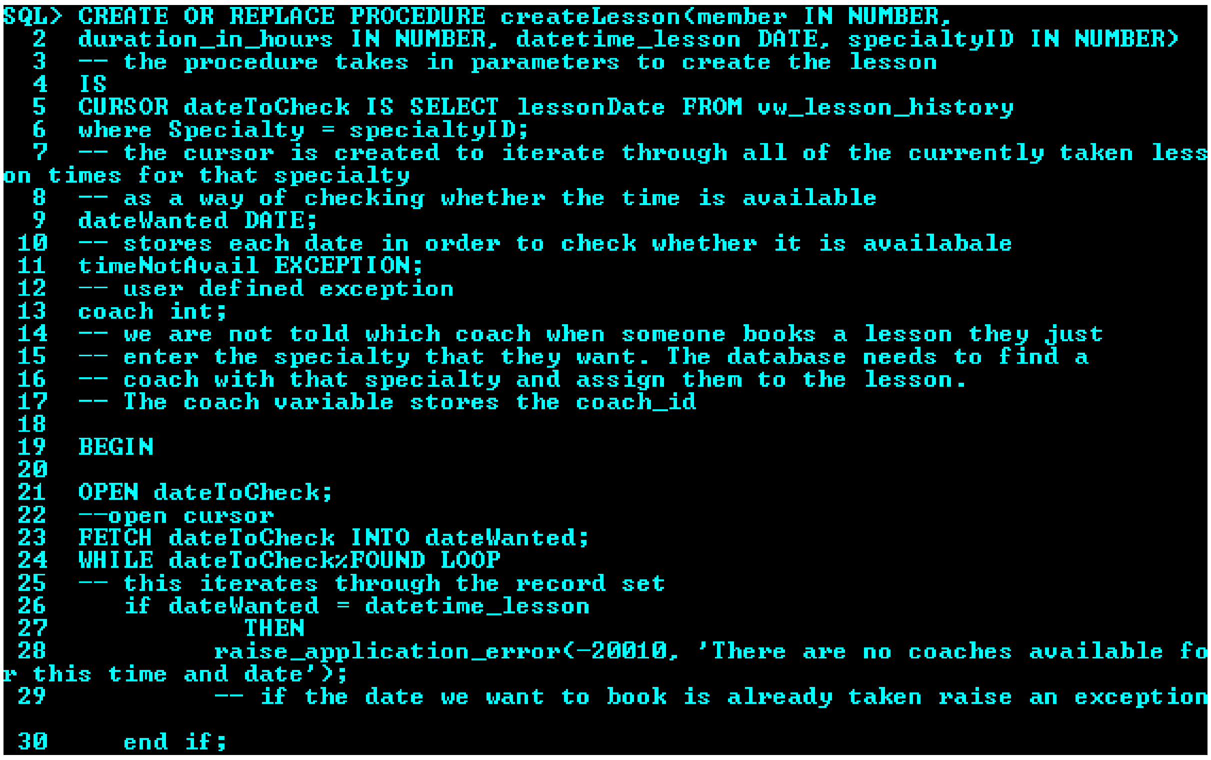


```
SQL> SELECT * FROM vw_lesson_history ORDER by lessonDate;

LESSONDAT MEMBERNAME           SPECIALTY COACHNAME                 HOURS
--------- -------------------- --------- -------------------- ----------
03-JAN-14 Reginald Magee               2 Adam Scotson                  2
03-JAN-14 Reginald Magee               1 Adam Scotson                  2
08-JAN-14 Peter Toohey                 3 Ciaran McCarthy               1
23-JAN-14 Sarah Cullen                 3 Tommy Pollett                 1
03-FEB-14 Sharon Dooley                3 Ciaran McCarthy               2
03-FEB-14 Sharon Dooley                1 Alan Pearson                  1
03-FEB-14 Sharon Dooley                4 Alan Pearson                  1
03-FEB-14 Angie Nutley                 2 Larry Paige                   1
03-FEB-14 Sharon Dooley                1 Adam Scotson                  2
03-FEB-14 Sharon Dooley                2 Adam Scotson                  2
05-FEB-14 Peter Toohey                 3 Paul Hammill                  1

LESSONDAT MEMBERNAME           SPECIALTY COACHNAME                 HOURS
--------- -------------------- --------- -------------------- ----------
05-FEB-14 Peter Toohey                 2 Paul Hammill                  1
12-FEB-14 Sheila Dooley                1 Alan Pearson                  1
12-FEB-14 Sheila Dooley                4 Alan Pearson                  1
13-FEB-14 Sheila Dooley                2 Paul Hammill                  1
13-FEB-14 Sheila Dooley                3 Paul Hammill                  1
20-FEB-14 Sheila Dooley                3 Jimmy Fox                     2
20-FEB-14 Sheila Dooley                4 Jimmy Fox                     2
22-FEB-14 Dougie Hauser                4 Alan Pearson                  2
22-FEB-14 Dougie Hauser                1 Alan Pearson                  2
24-FEB-14 Harold Ramis                 3 Tommy Pollett                 2
05-MAR-14 Harold Ramis                 2 Adam Scotson                  1
```

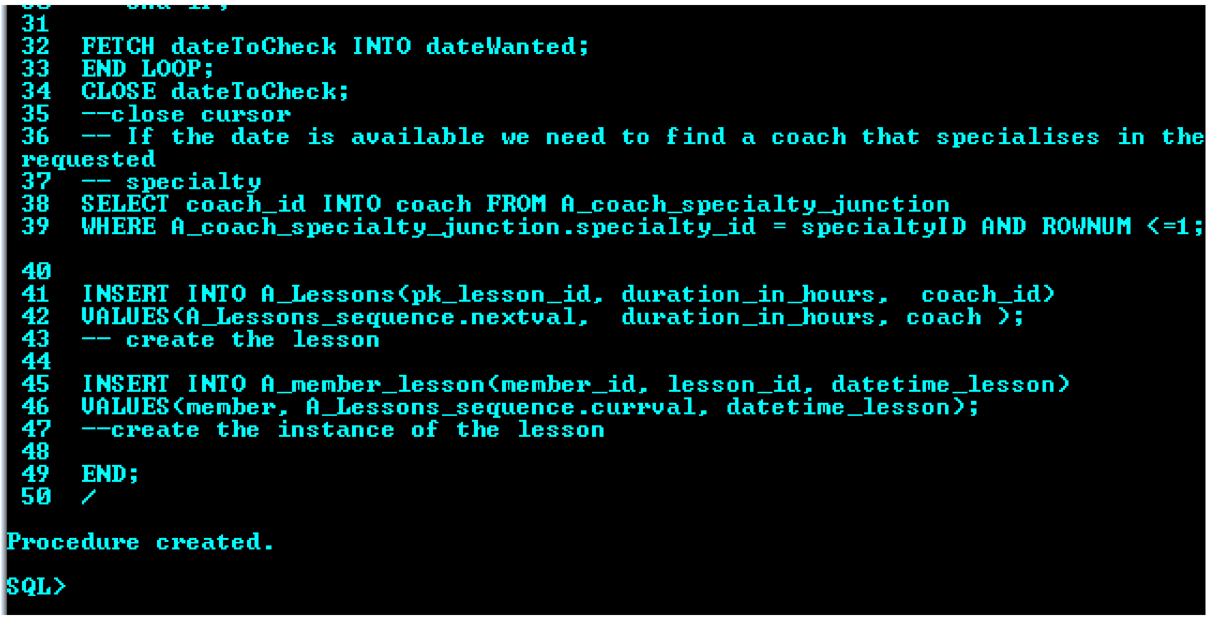
Procedure 3 createLesson

```
CREATE OR REPLACE PROCEDURE createLesson(member IN NUMBER,

duration_in_hours IN NUMBER, datetime_lesson DATE, specialtyID IN NUMBER)

-- the procedure takes in parameters to create the lesson

IS

CURSOR dateToCheck IS SELECT lessonDate FROM vw_lesson_history

where Specialty = specialtyID;

-- the cursor is created to iterate through all of the currently taken lesson times for that specialty

-- as a way of checking whether the time is available

dateWanted DATE;

-- stores each date in order to check whether it is availabale

timeNotAvail EXCEPTION;

-- user defined exception

coach int;

-- we are not told which coach when someone books a lesson they just

-- enter the specialty that they want. The database needs to find a

-- coach with that specialty and assign them to the lesson.

-- The coach variable stores the coach_id


BEGIN


OPEN dateToCheck;

--open cursor

FETCH dateToCheck INTO dateWanted;

WHILE dateToCheck%FOUND LOOP

-- this iterates through the record set

        if dateWanted = datetime_lesson
```

```
                THEN
            raise_application_error(-20010, 'There are no coaches available for this time and date');

            -- if the date we want to book is already taken raise an exception
        end if;


FETCH dateToCheck INTO dateWanted;
END LOOP;
CLOSE dateToCheck;
--close cursor
-- If the date is available we need to find a coach that specialises in the requested
-- specialty
SELECT coach_id INTO coach FROM A_coach_specialty_junction
WHERE A_coach_specialty_junction.specialty_id = specialtyID AND ROWNUM <=1;


INSERT INTO A_Lessons(pk_lesson_id, duration_in_hours,  coach_id)
VALUES(A_Lessons_sequence.nextval,  duration_in_hours, coach );
-- create the lesson


INSERT INTO A_member_lesson(member_id, lesson_id, datetime_lesson)
VALUES(member, A_Lessons_sequence.currval, datetime_lesson);
--create the instance of the lesson


END;
/
```



```
SQL> CREATE OR REPLACE PROCEDURE createLesson(member IN NUMBER,
duration_in_hours IN NUMBER, datetime_lesson DATE, specialtyID IN NUMBER)
-- the procedure takes in parameters to create the lesson
IS
CURSOR dateToCheck IS SELECT lessonDate FROM vw_lesson_history
where Specialty = specialtyID;
-- the cursor is created to iterate through all of the currently taken lesson times for that specialty
-- as a way of checking whether the time is available
dateWanted DATE;
-- stores each date in order to check whether it is availabale
timeNotAvail EXCEPTION;
-- user defined exception
coach int;
-- we are not told which coach when someone books a lesson they just
-- enter the specialty that they want. The database needs to find a
-- coach with that specialty and assign them to the lesson.
-- The coach variable stores the coach_id

BEGIN

OPEN dateToCheck;
--open cursor
FETCH dateToCheck INTO dateWanted;
WHILE dateToCheck%FOUND LOOP
-- this iterates through the record set
   if dateWanted = datetime_lesson
           THEN
         raise_application_error(-20010, 'There are no coaches available for this time and date');
         -- if the date we want to book is already taken raise an exception

   end if;
```



```

FETCH dateToCheck INTO dateWanted;
END LOOP;
CLOSE dateToCheck;
--close cursor
-- If the date is available we need to find a coach that specialises in the requested
-- specialty
SELECT coach_id INTO coach FROM A_coach_specialty_junction
WHERE A_coach_specialty_junction.specialty_id = specialtyID AND ROWNUM <=1;

INSERT INTO A_Lessons(pk_lesson_id, duration_in_hours,  coach_id)
VALUES(A_Lessons_sequence.nextval,  duration_in_hours, coach );
-- create the lesson

INSERT INTO A_member_lesson(member_id, lesson_id, datetime_lesson)
VALUES(member, A_Lessons_sequence.currval, datetime_lesson);
--create the instance of the lesson

END;
/

Procedure created.

SQL>
```

execute createLesson(1, 2, TO_DATE('2014/05/05 19:00:00','yyyy/mm/dd hh24:mi:ss'), 3)

-- member_id = 1 ---- number of hours = 2 ------specialty = 3 (Irons)

--tests

SELECT * from A_member_lesson WHERE member_ID = 1;

--this shows that the instance of a lesson has been created

SELECT * from A_Lessons WHERE pk_lesson_ID >20;

--this shows that a lesson has been assigned a coach who specialises in specialty = 3 (Irons)

-- >20 is just from my knowledge that only 20 lessons currently exist in the

-- database so any created by the procedure should be dispalyed as they will have a primary

--key >20 due to it being created by the A_Lessons.sequence

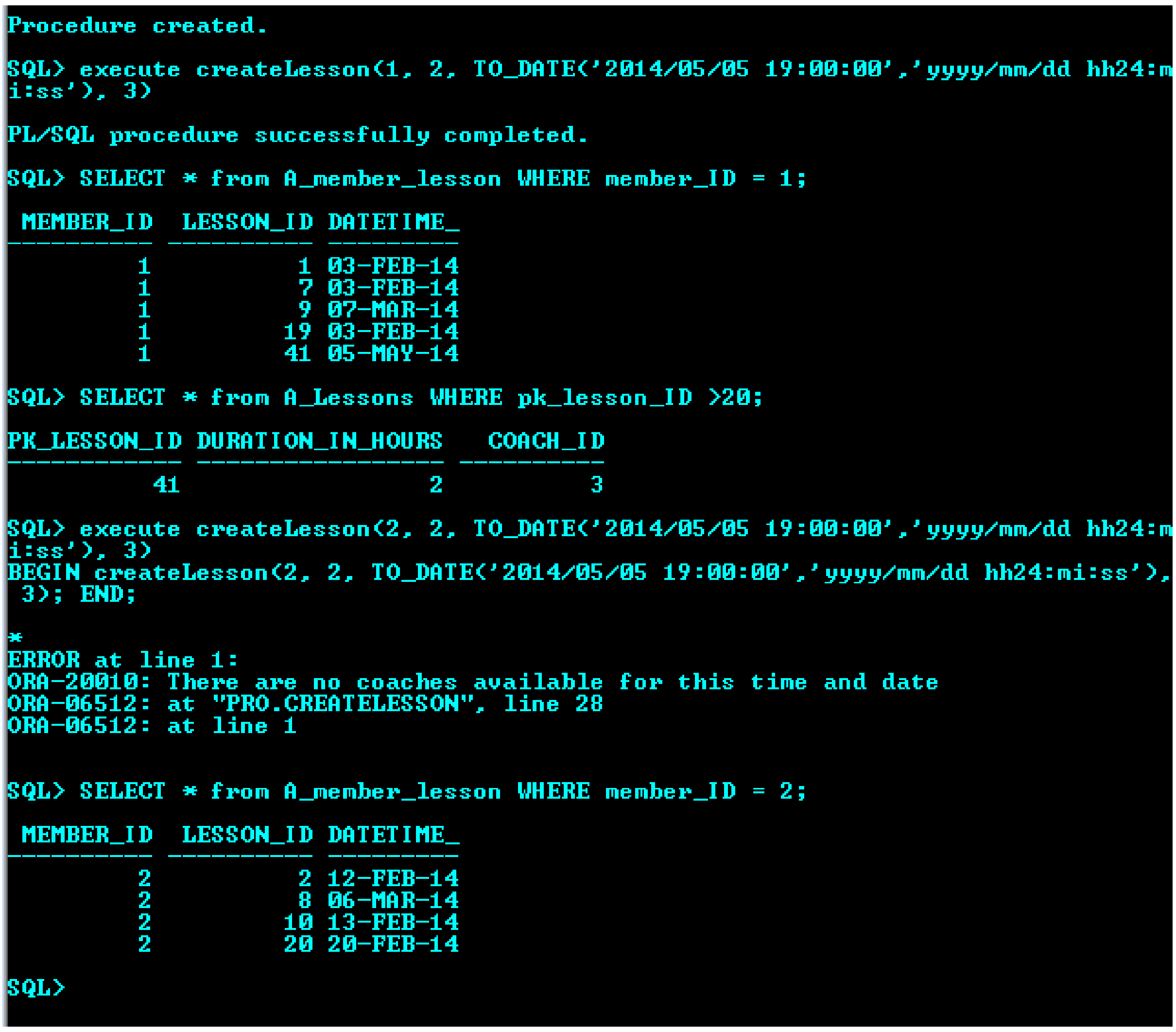
execute createLesson(2, 2, TO_DATE('2014/05/05 19:00:00','yyyy/mm/dd hh24:mi:ss'), 3)

--this shows that if any member(in this case member_id =2) tries to book a

--Irons lessona t this time and date an error will be thrown

SELECT * from A_member_lesson WHERE member_ID = 2;

--this shows that the instance of a lesson has not been created

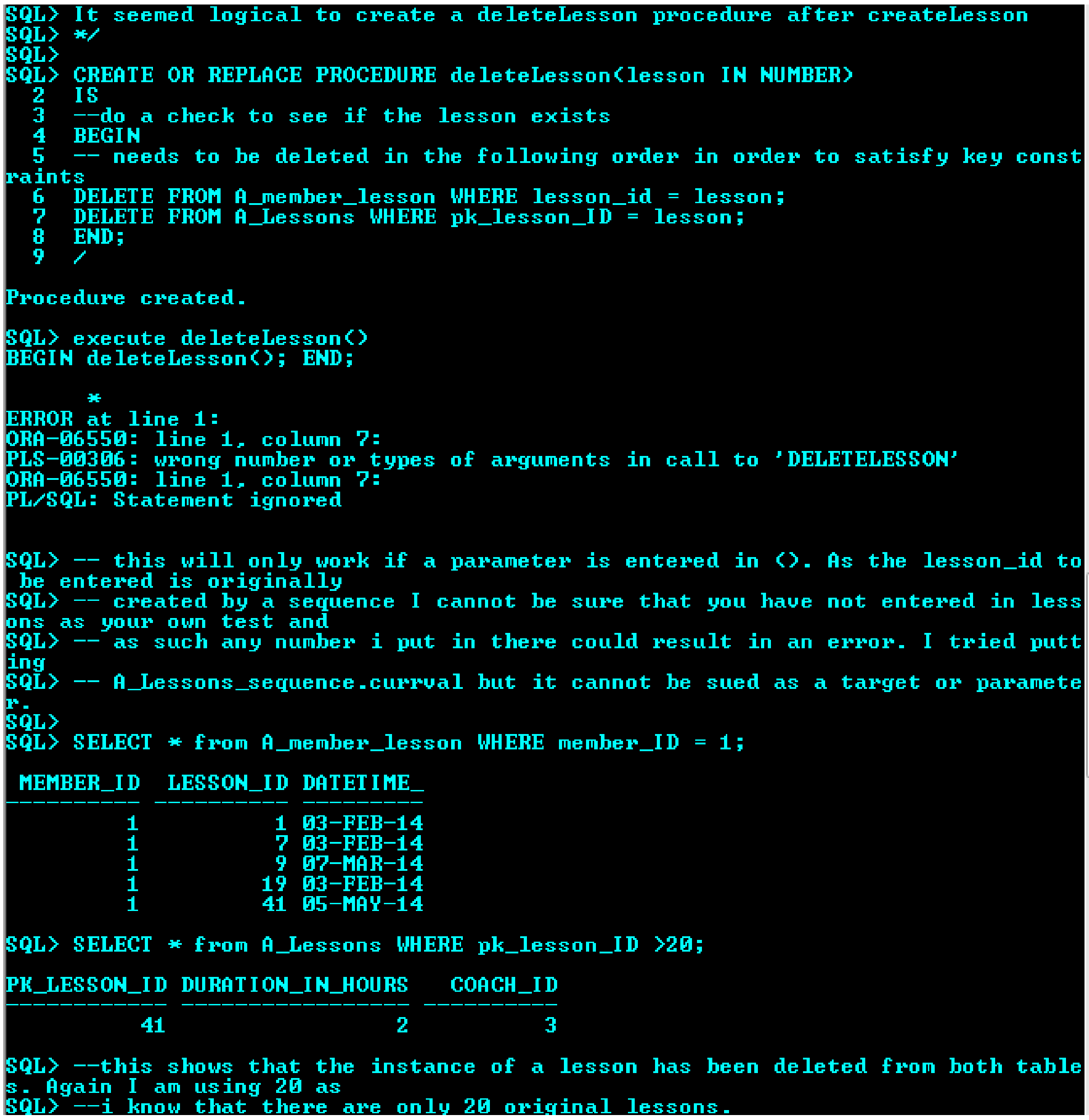


```
Procedure created.

SQL> execute createLesson(1, 2, TO_DATE('2014/05/05 19:00:00','yyyy/mm/dd hh24:m
i:ss'), 3)

PL/SQL procedure successfully completed.

SQL> SELECT * from A_member_lesson WHERE member_ID = 1;

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
         1          1 03-FEB-14
         1          7 03-FEB-14
         1          9 07-MAR-14
         1         19 03-FEB-14
         1         41 05-MAY-14

SQL> SELECT * from A_Lessons WHERE pk_lesson_ID >20;

PK_LESSON_ID DURATION_IN_HOURS   COACH_ID
------------ ----------------- ----------
          41                 2          3

SQL> execute createLesson(2, 2, TO_DATE('2014/05/05 19:00:00','yyyy/mm/dd hh24:m
i:ss'), 3)
BEGIN createLesson(2, 2, TO_DATE('2014/05/05 19:00:00','yyyy/mm/dd hh24:mi:ss'),
 3); END;

*
ERROR at line 1:
ORA-20010: There are no coaches available for this time and date
ORA-06512: at "PRO.CREATELESSON", line 28
ORA-06512: at line 1


SQL> SELECT * from A_member_lesson WHERE member_ID = 2;

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
         2          2 12-FEB-14
         2          8 06-MAR-14
         2         10 13-FEB-14
         2         20 20-FEB-14

SQL>
```

Procedure 4 deleteLesson

It seemed logical to create a deleteLesson procedure after createLesson

```
CREATE OR REPLACE PROCEDURE deleteLesson(lesson IN NUMBER)

IS

--do a check to see if the lesson exists

BEGIN

-- needs to be deleted in the following order in order to satisfy key constraints

DELETE FROM A_member_lesson WHERE lesson_id = lesson;

DELETE FROM A_Lessons WHERE pk_lesson_ID = lesson;

END;

/


execute deleteLesson()

-- this will only work if a parameter is entered in (). As the lesson_id to be entered is originally

-- created by a sequence I cannot be sure that you have not entered in lessons as your own test and

-- as such any number i put in there could result in an error. I tried putting

-- A_Lessons_sequence.currval but it cannot be sued as a target or parameter.


SELECT * from A_member_lesson WHERE member_ID = 1;

SELECT * from A_Lessons WHERE pk_lesson_ID >20;

--this shows that the instance of a lesson has been deleted from both tables. Again I am using 20 as

--i know that there are only 20 original lessons.
```



```
SQL> It seemed logical to create a deleteLesson procedure after createLesson
SQL> */
SQL>
SQL> CREATE OR REPLACE PROCEDURE deleteLesson(lesson IN NUMBER)
  2  IS
  3  --do a check to see if the lesson exists
  4  BEGIN
  5  -- needs to be deleted in the following order in order to satisfy key const
raints
  6  DELETE FROM A_member_lesson WHERE lesson_id = lesson;
  7  DELETE FROM A_Lessons WHERE pk_lesson_ID = lesson;
  8  END;
  9  /

Procedure created.

SQL> execute deleteLesson()
BEGIN deleteLesson(); END;

      *
ERROR at line 1:
ORA-06550: line 1, column 7:
PLS-00306: wrong number or types of arguments in call to 'DELETELESSON'
ORA-06550: line 1, column 7:
PL/SQL: Statement ignored


SQL> -- this will only work if a parameter is entered in (). As the lesson_id to
 be entered is originally
SQL> -- created by a sequence I cannot be sure that you have not entered in less
ons as your own test and
SQL> -- as such any number i put in there could result in an error. I tried putt
ing
SQL> -- A_Lessons_sequence.currval but it cannot be sued as a target or paramete
r.
SQL>
SQL> SELECT * from A_member_lesson WHERE member_ID = 1;

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
         1          1 03-FEB-14
         1          7 03-FEB-14
         1          9 07-MAR-14
         1         19 03-FEB-14
         1         41 05-MAY-14

SQL> SELECT * from A_Lessons WHERE pk_lesson_ID >20;

PK_LESSON_ID DURATION_IN_HOURS   COACH_ID
------------ ----------------- ----------
          41                 2          3

SQL> --this shows that the instance of a lesson has been deleted from both table
s. Again I am using 20 as
SQL> --i know that there are only 20 original lessons.
```

Procedure 5 processPayment

This procedure processes a payment and should be the default option for any attempted payments.

Whatever the member pays for the correct tables will be updated in the database.

This also shows the use of Savepoint and Rollback as the tests will show.

I have only shown that the procedure was successfully executed as it is too long to shown all of it.

```
cl scr;

CREATE OR REPLACE PROCEDURE processPayment(        amountOfItems IN NUMBER,

            member IN NUMBER , paymentType IN NUMBER, category IN NUMBER, lessonDate DATE :=
NULL)

--by setting lessonDate DATE := NULL we have made lessonDate an optional parameter

-- this is because when booking a lesson a date is required whilst the other payments do not need a
date.

IS


Fname NVARCHAR2(30);

Sname NVARCHAR2(30);

tournament NUMBER;

duration_in_hours NUMBER;

specialty NUMBER;

space NUMBER;

membership_id NUMBER;

invalidCat EXCEPTION;

--membership_id is required because otherwise

--when we try to insert member into cpk_tournament we get an error as there is a

--possibility of a NUll vale being entered into a primary key field.

BEGIN
```

```
savepoint beforeInsert;
-- no field exists to be populated. This could happen if a member tries to pay for a
--tournament that does not exist yet.
-- or if they try to pay for a parking space but none are available.

SELECT A_Members.Fname INTO Fname FROM A_Members WHERE A_Members.pk_member_id = member;
--stores the members firstname in Fname

SELECT A_Members.Sname INTO Sname FROM A_Members WHERE A_Members.pk_member_id = member;
--stores the members firstname in Fname
membership_id := member;

SELECT pk_tournament_id INTO tournament FROM A_Tournaments WHERE (tournament_date - sysdate) BETWEEN 0 AND 7;
-- This stores the closest future tournament in the variable tournament
-- There is a business rule that their is a tournament every 7 days.

if tournament is NULL then
  tournament :=-1;
end if;

if membership_id is NULL then
  membership_id :=-1;
end if;
-- these are required to avoid the null value being inserted into a the composite primary key for the table
-- A_Participation_Results
```

```
INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_id, category_id)

VALUES(A_Payment_sequence.nextval, sysdate, amountOfItems, member, paymentType, category);

--this creates the payment


CASE category

-- this case asigns the correct duration and specialty based on the correct category_id being entered

-- Nothing needs to occur when category_id = 1 as this membership.


WHEN 1 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid their membership');


WHEN 2 THEN duration_in_hours := 1; Specialty := 1 ;

createLesson(member, duration_in_hours, lessonDate, specialty);

-- this calls another procedure to create the correct insert statements


WHEN 3 THEN duration_in_hours := 2; Specialty := 1;

createLesson(member, duration_in_hours, lessonDate, specialty);

-- this calls another procedure to create the correct insert statements etc


WHEN 4 THEN duration_in_hours := 1; Specialty :=  2;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 5 THEN duration_in_hours := 2; Specialty :=  2;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 6 THEN duration_in_hours := 1; Specialty :=  3;
```

```
createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 7 THEN duration_in_hours := 2; Specialty :=  3;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 8 THEN duration_in_hours := 1; Specialty :=  4;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 9 THEN duration_in_hours := 2; Specialty :=  4;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 10 THEN

-- 10 is the category type of a parking space

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a parking space');

SELECT pk_parking_space INTO space FROM A_Parking_Space where member_id IS NULL AND
ROWNUM <=1;

--selects the next available space

-- if all the parking spaces are already assigned the exception below will handle it by outputting the

-- error message and cancelling the payment.

UPDATE A_Parking_Space

SET member_id = member WHERE pk_parking_space = space;



WHEN 11 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);
```

```
WHEN 12 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);



WHEN 13 THEN

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament');

INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score)

VALUES(membership_id, tournament, -1);



WHEN 21 THEN duration_in_hours := 1; Specialty :=  5;

createLesson(member, duration_in_hours, lessonDate, specialty);


WHEN 22 THEN duration_in_hours := 2; Specialty :=  5;

createLesson(member, duration_in_hours, lessonDate, specialty);


ELSE DBMS_OUTPUT.PUT_LINE('No such category'); raise_application_error(-20011, 'This is not a
category_id '||SQLCODE||' -ERROR- '||SQLERRM);

END CASE;


EXCEPTION

WHEN NO_DATA_FOUND THEN

dbms_output.put_line('The field you are trying to populate does not exist');

ROLLBACK to beforeInsert;

-- cancel the payment by the member. This will only delete the car parking payment as a member
cannot
```

--   have more than one assigned parking space

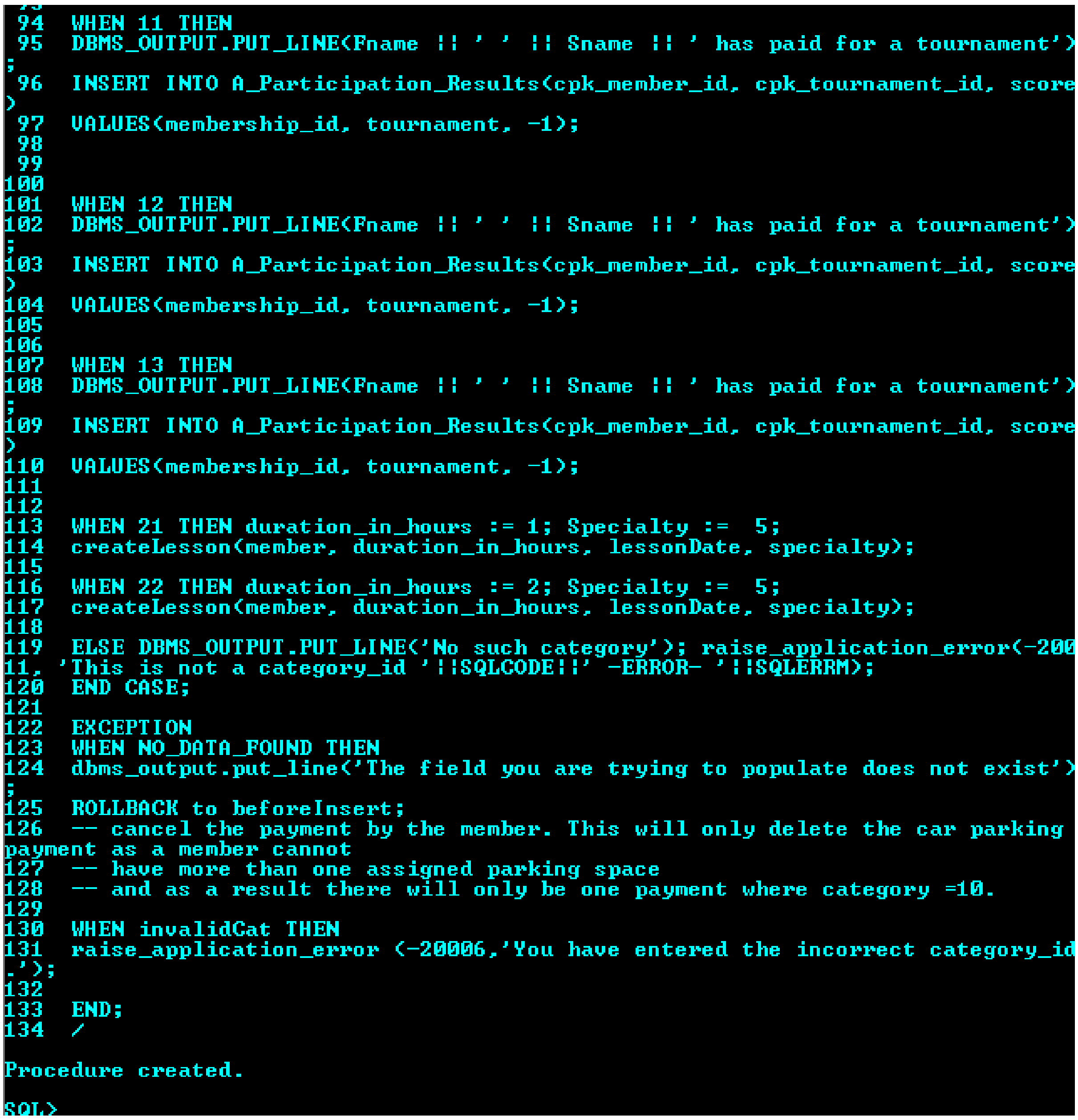
-- and as a result there will only be one payment where category =10.

WHEN invalidCat THEN

raise_application_error (-20006,'You have entered the incorrect category_id.');

END;

/



```
WHEN 11 THEN
DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament')
;
INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score
)
VALUES(membership_id, tournament, -1);



WHEN 12 THEN
DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament')
;
INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score
)
VALUES(membership_id, tournament, -1);


WHEN 13 THEN
DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a tournament')
;
INSERT INTO A_Participation_Results(cpk_member_id, cpk_tournament_id, score
)
VALUES(membership_id, tournament, -1);


WHEN 21 THEN duration_in_hours := 1; Specialty :=  5;
createLesson(member, duration_in_hours, lessonDate, specialty);

WHEN 22 THEN duration_in_hours := 2; Specialty :=  5;
createLesson(member, duration_in_hours, lessonDate, specialty);

ELSE DBMS_OUTPUT.PUT_LINE('No such category'); raise_application_error(-200
11, 'This is not a category_id '||SQLCODE||' -ERROR- '||SQLERRM);
END CASE;

EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('The field you are trying to populate does not exist')
;
ROLLBACK to beforeInsert;
-- cancel the payment by the member. This will only delete the car parking
payment as a member cannot
-- have more than one assigned parking space
-- and as a result there will only be one payment where category =10.

WHEN invalidCat THEN
raise_application_error (-20006,'You have entered the incorrect category_id
.');

END;
/

Procedure created.

SQL>
```

I have performed a number of tests to show the workings of this procedure. They show all of the various different options.

execute processPayment(1, 2, 3, 11)

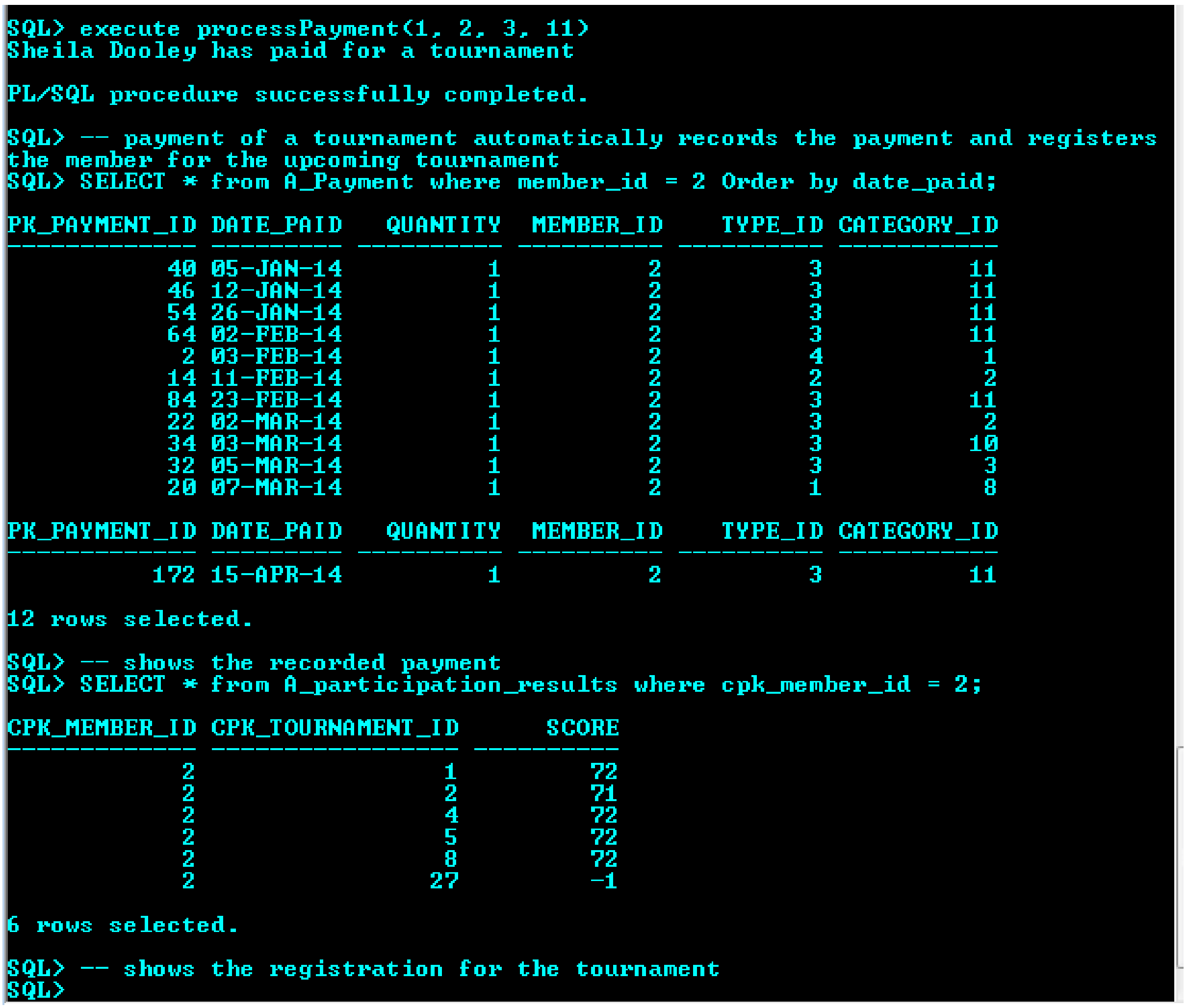
-- payment of a tournament automatically records the payment and registers the member for the upcoming tournament

SELECT * from A_Payment where member_id = 2 Order by date_paid;

-- shows the recorded payment

SELECT * from A_participation_results where cpk_member_id = 2;

-- shows the registration for the tournament



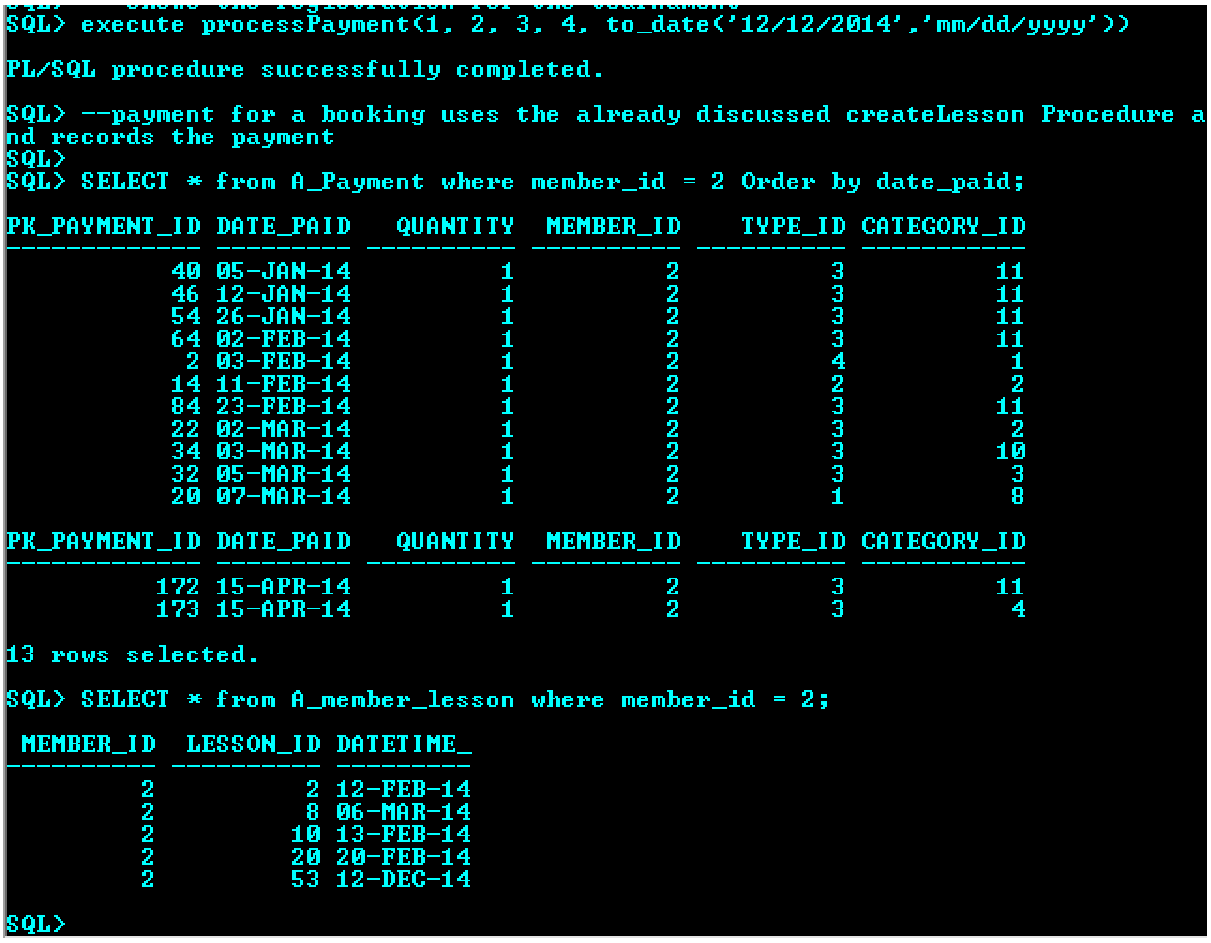
```
SQL> execute processPayment(1, 2, 3, 11)
Sheila Dooley has paid for a tournament

PL/SQL procedure successfully completed.

SQL> -- payment of a tournament automatically records the payment and registers
the member for the upcoming tournament
SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
          172 15-APR-14          1          2          3          11

12 rows selected.

SQL> -- shows the recorded payment
SQL> SELECT * from A_participation_results where cpk_member_id = 2;

CPK_MEMBER_ID CPK_TOURNAMENT_ID      SCORE
------------- ----------------- ----------
            2                 1         72
            2                 2         71
            2                 4         72
            2                 5         72
            2                 8         72
            2                27         -1

6 rows selected.

SQL> -- shows the registration for the tournament
SQL>
```

execute processPayment(1, 2, 3, 4, to_date('12/12/2014','mm/dd/yyyy'))

--payment for a booking uses the already discussed createLesson Procedure and records the payment

SELECT * from A_Payment where member_id = 2 Order by date_paid;

SELECT * from A_member_lesson where member_id = 2;



```
SQL> execute processPayment(1, 2, 3, 4, to_date('12/12/2014','mm/dd/yyyy'))

PL/SQL procedure successfully completed.

SQL> --payment for a booking uses the already discussed createLesson Procedure a
nd records the payment
SQL>
SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
          172 15-APR-14          1          2          3          11
          173 15-APR-14          1          2          3           4

13 rows selected.

SQL> SELECT * from A_member_lesson where member_id = 2;

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
         2          2 12-FEB-14
         2          8 06-MAR-14
         2         10 13-FEB-14
         2         20 20-FEB-14
         2         53 12-DEC-14

SQL>
```

execute processPayment(1, 2, 3, 4, to_date('12/12/2014','mm/dd/yyyy'))

SELECT * from A_Payment where member_id = 2 Order by date_paid;

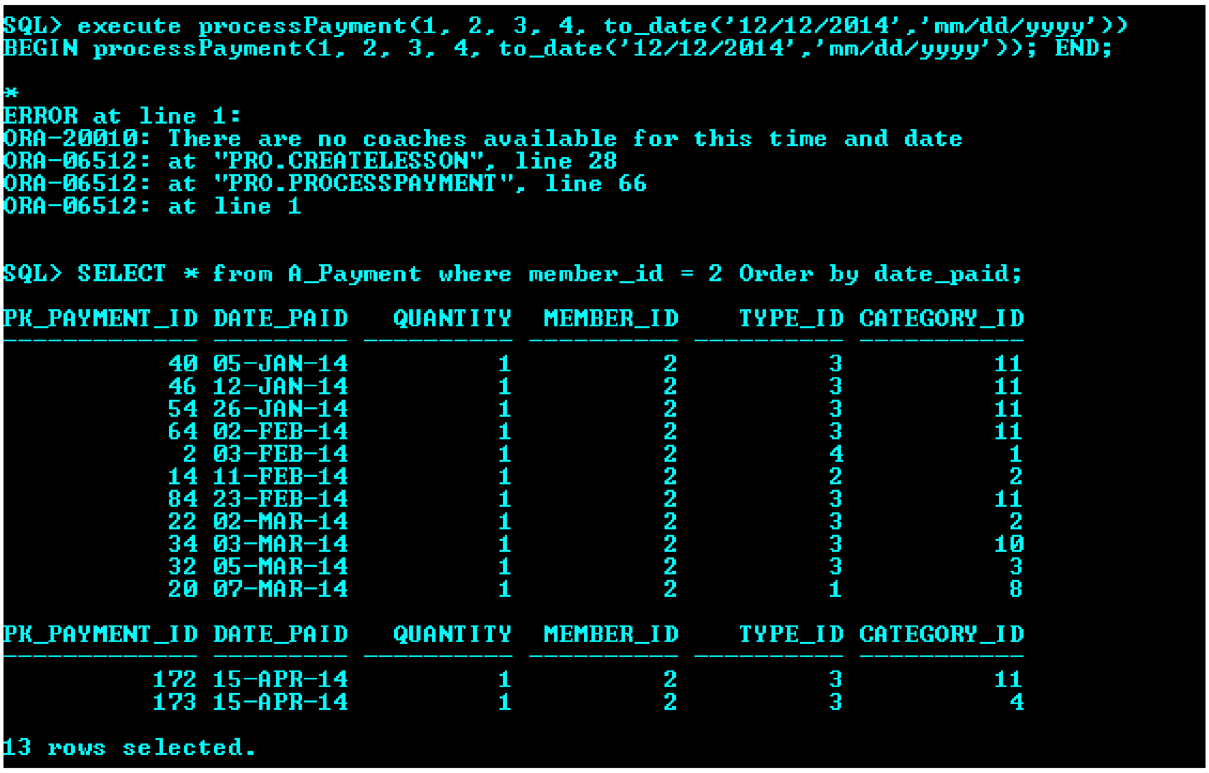
--shows that any attempt to create a lesson with that coach at the

--same time as above time will throw an error and no new payment will be recorded because of the error handling

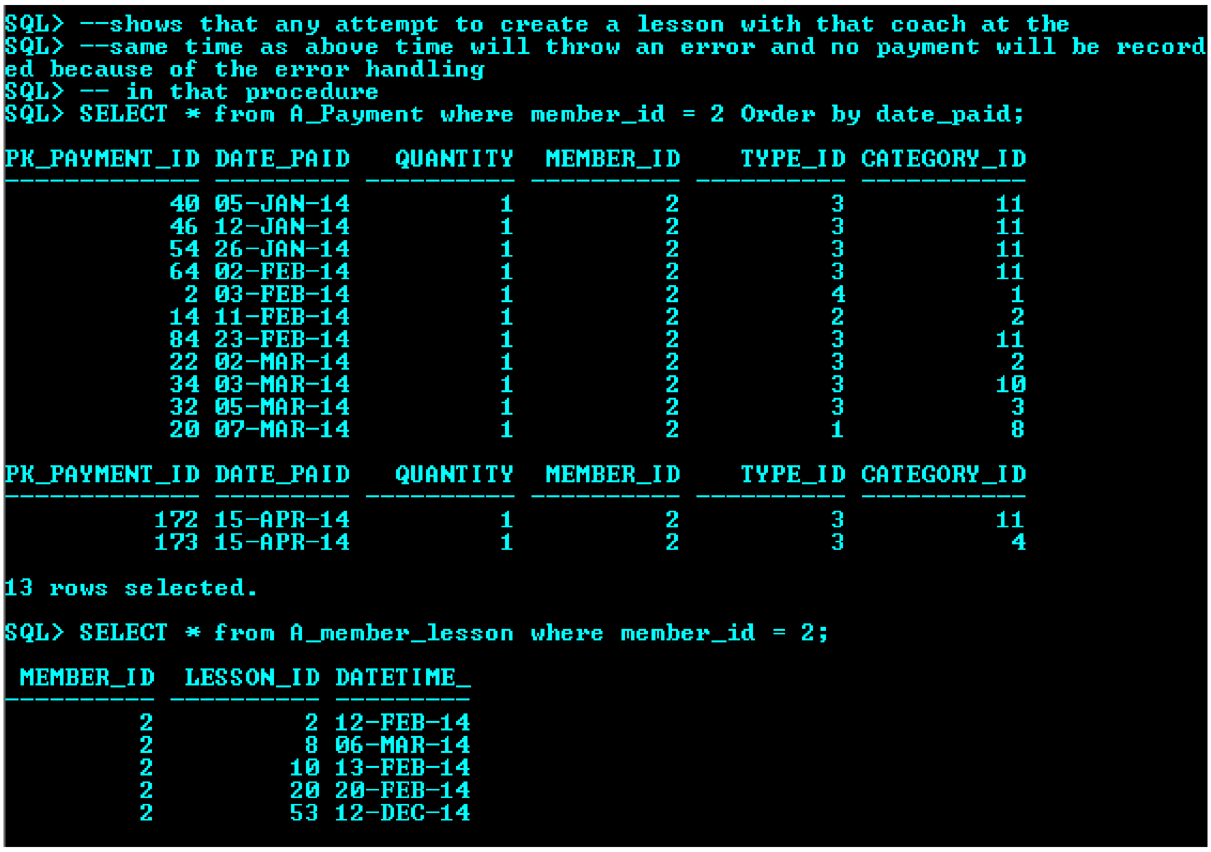
-- in that procedure

SELECT * from A_Payment where member_id = 2 Order by date_paid;

SELECT * from A_member_lesson where member_id = 2;



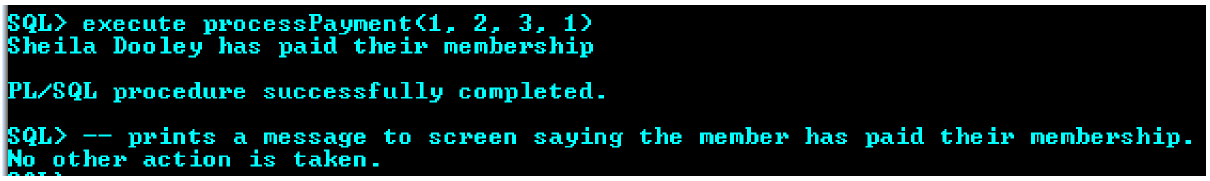
```
SQL> execute processPayment(1, 2, 3, 4, to_date('12/12/2014','mm/dd/yyyy'))
BEGIN processPayment(1, 2, 3, 4, to_date('12/12/2014','mm/dd/yyyy')); END;

*
ERROR at line 1:
ORA-20010: There are no coaches available for this time and date
ORA-06512: at "PRO.CREATELESSON", line 28
ORA-06512: at "PRO.PROCESSPAYMENT", line 66
ORA-06512: at line 1


SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
          172 15-APR-14          1          2          3          11
          173 15-APR-14          1          2          3           4

13 rows selected.
```



```
SQL> --shows that any attempt to create a lesson with that coach at the
SQL> --same time as above time will throw an error and no payment will be record
ed because of the error handling
SQL> -- in that procedure
SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
          172 15-APR-14          1          2          3          11
          173 15-APR-14          1          2          3           4

13 rows selected.

SQL> SELECT * from A_member_lesson where member_id = 2;

 MEMBER_ID  LESSON_ID DATETIME_
---------- ---------- ---------
         2          2 12-FEB-14
         2          8 06-MAR-14
         2         10 13-FEB-14
         2         20 20-FEB-14
         2         53 12-DEC-14
```

execute processPayment(1, 2, 3, 1)

-- prints a message to screen saying the member has paid their membership. No other action is taken.



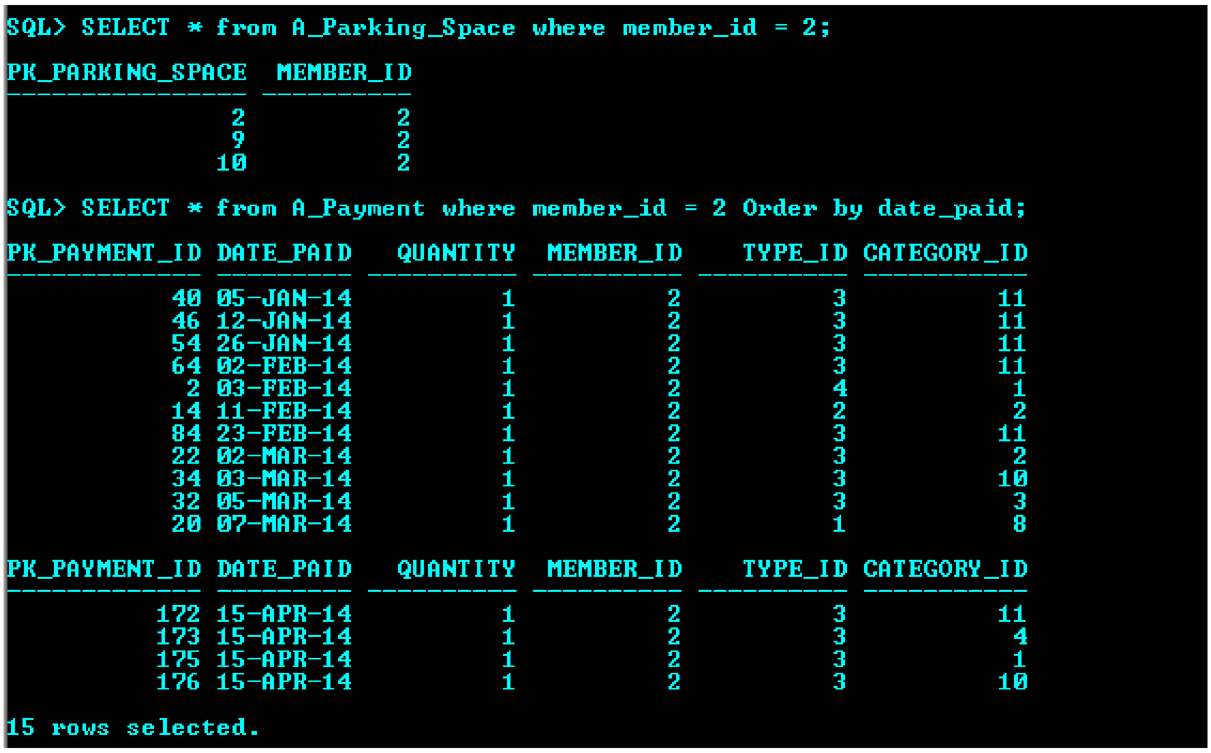
```
SQL> execute processPayment(1, 2, 3, 1)
Sheila Dooley has paid their membership

PL/SQL procedure successfully completed.

SQL> -- prints a message to screen saying the member has paid their membership.
No other action is taken.
```

execute processPayment(1, 2, 3, 10)

SELECT * from A_Parking_Space where member_id = 2;

SELECT * from A_Payment where member_id = 2 Order by date_paid;

-- shows the payment of a parking space and the assigned space



```
SQL> SELECT * from A_Parking_Space where member_id = 2;

PK_PARKING_SPACE  MEMBER_ID
---------------- ----------
               2          2
               9          2
              10          2

SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
          172 15-APR-14          1          2          3          11
          173 15-APR-14          1          2          3           4
          175 15-APR-14          1          2          3           1
          176 15-APR-14          1          2          3          10

15 rows selected.
```

UPDATE A_Parking_Space SET member_id = null

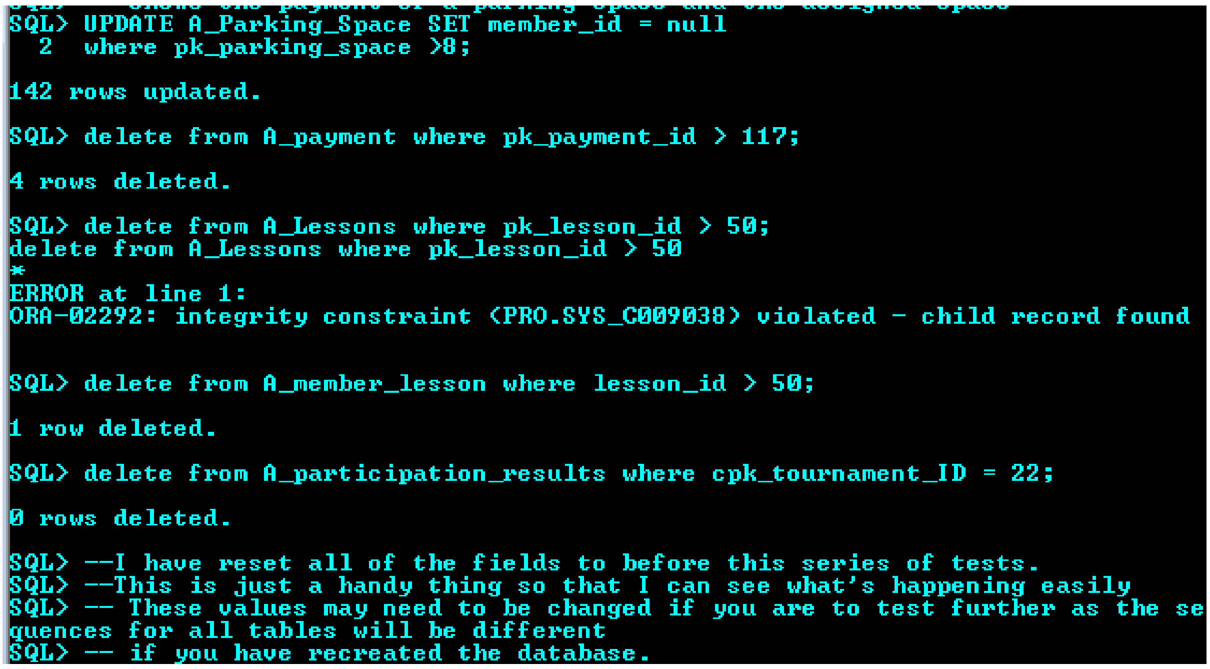
where pk_parking_space >8;

delete from A_payment where pk_payment_id > 117;

delete from A_Lessons where pk_lesson_id > 50;

delete from A_member_lesson where lesson_id > 50;

delete from A_participation_results where cpk_tournament_ID = 22;



```
SQL> UPDATE A_Parking_Space SET member_id = null
  2  where pk_parking_space >8;

142 rows updated.

SQL> delete from A_payment where pk_payment_id > 117;

4 rows deleted.

SQL> delete from A_Lessons where pk_lesson_id > 50;
delete from A_Lessons where pk_lesson_id > 50
*
ERROR at line 1:
ORA-02292: integrity constraint (PRO.SYS_C009038) violated - child record found


SQL> delete from A_member_lesson where lesson_id > 50;

1 row deleted.

SQL> delete from A_participation_results where cpk_tournament_ID = 22;

0 rows deleted.

SQL> --I have reset all of the fields to before this series of tests.
SQL> --This is just a handy thing so that I can see what's happening easily
SQL> -- These values may need to be changed if you are to test further as the se
quences for all tables will be different
SQL> -- if you have recreated the database.
```

-- in order to show the rollback in effect I have deleted the

DELETE from A_Tournaments WHERE tournament_date> sysdate;

-- now if I run the execute statement for a tournament payment below I will receive back the

--user-defined error and if we run the tests we can see that no transaction occurred as it

--rolled back to the savepoint beforeInsert

execute processPayment(1, 2, 3, 11)

SELECT * from A_Payment where member_id = 2 Order by date_paid;

-- shows the recorded payment

SELECT * from A_participation_results where cpk_member_id = 2;

-- shows the registration for the tournament

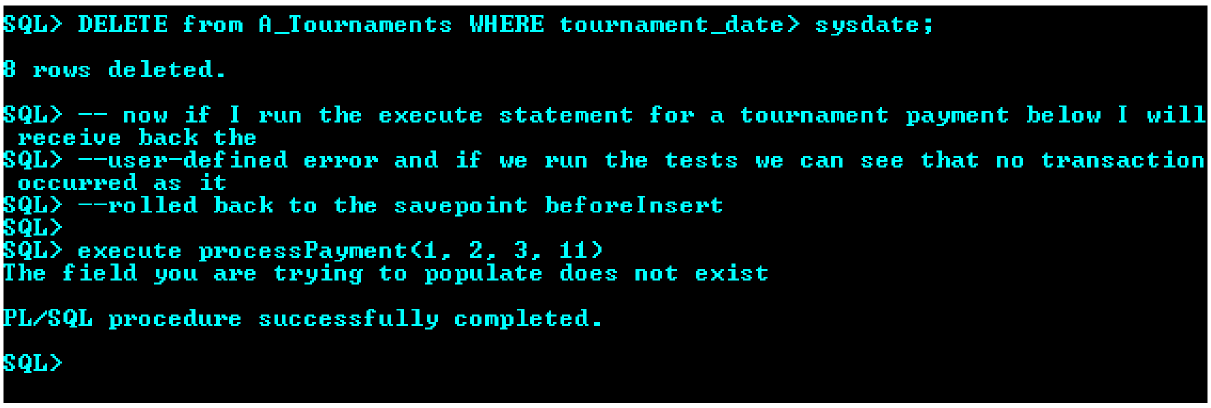
DELETE from A_Tournaments WHERE tournament_date> sysdate;

-- now if I run the execute statement for a tournament payment below I will receive back the

--user-defined error and if we run the tests we can see that no transaction occurred as it

--rolled back to the savepoint beforeInsert

execute processPayment(1, 2, 3, 11)



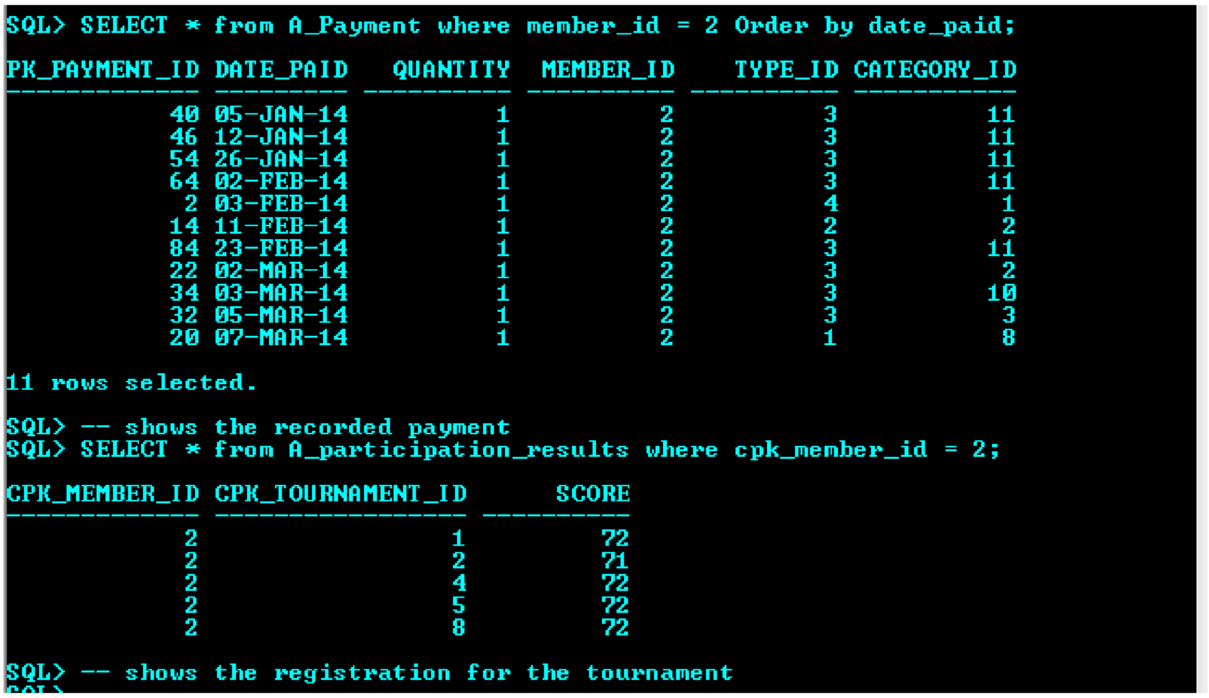
```
SQL> DELETE from A_Tournaments WHERE tournament_date> sysdate;

8 rows deleted.

SQL> -- now if I run the execute statement for a tournament payment below I will
 receive back the
SQL> --user-defined error and if we run the tests we can see that no transaction
 occurred as it
SQL> --rolled back to the savepoint beforeInsert
SQL>
SQL> execute processPayment(1, 2, 3, 11)
The field you are trying to populate does not exist

PL/SQL procedure successfully completed.

SQL>
```

SELECT * from A_Payment where member_id = 2 Order by date_paid;

-- shows the recorded payment

SELECT * from A_participation_results where cpk_member_id = 2;

-- shows the registration for the tournament


```
SQL> SELECT * from A_Payment where member_id = 2 Order by date_paid;

PK_PAYMENT_ID DATE_PAID   QUANTITY  MEMBER_ID    TYPE_ID CATEGORY_ID
------------- --------- ---------- ---------- ---------- -----------
           40 05-JAN-14          1          2          3          11
           46 12-JAN-14          1          2          3          11
           54 26-JAN-14          1          2          3          11
           64 02-FEB-14          1          2          3          11
            2 03-FEB-14          1          2          4           1
           14 11-FEB-14          1          2          2           2
           84 23-FEB-14          1          2          3          11
           22 02-MAR-14          1          2          3           2
           34 03-MAR-14          1          2          3          10
           32 05-MAR-14          1          2          3           3
           20 07-MAR-14          1          2          1           8

11 rows selected.

SQL> -- shows the recorded payment
SQL> SELECT * from A_participation_results where cpk_member_id = 2;

CPK_MEMBER_ID CPK_TOURNAMENT_ID      SCORE
------------- ----------------- ----------
            2                 1         72
            2                 2         71
            2                 4         72
            2                 5         72
            2                 8         72

SQL> -- shows the registration for the tournament
```

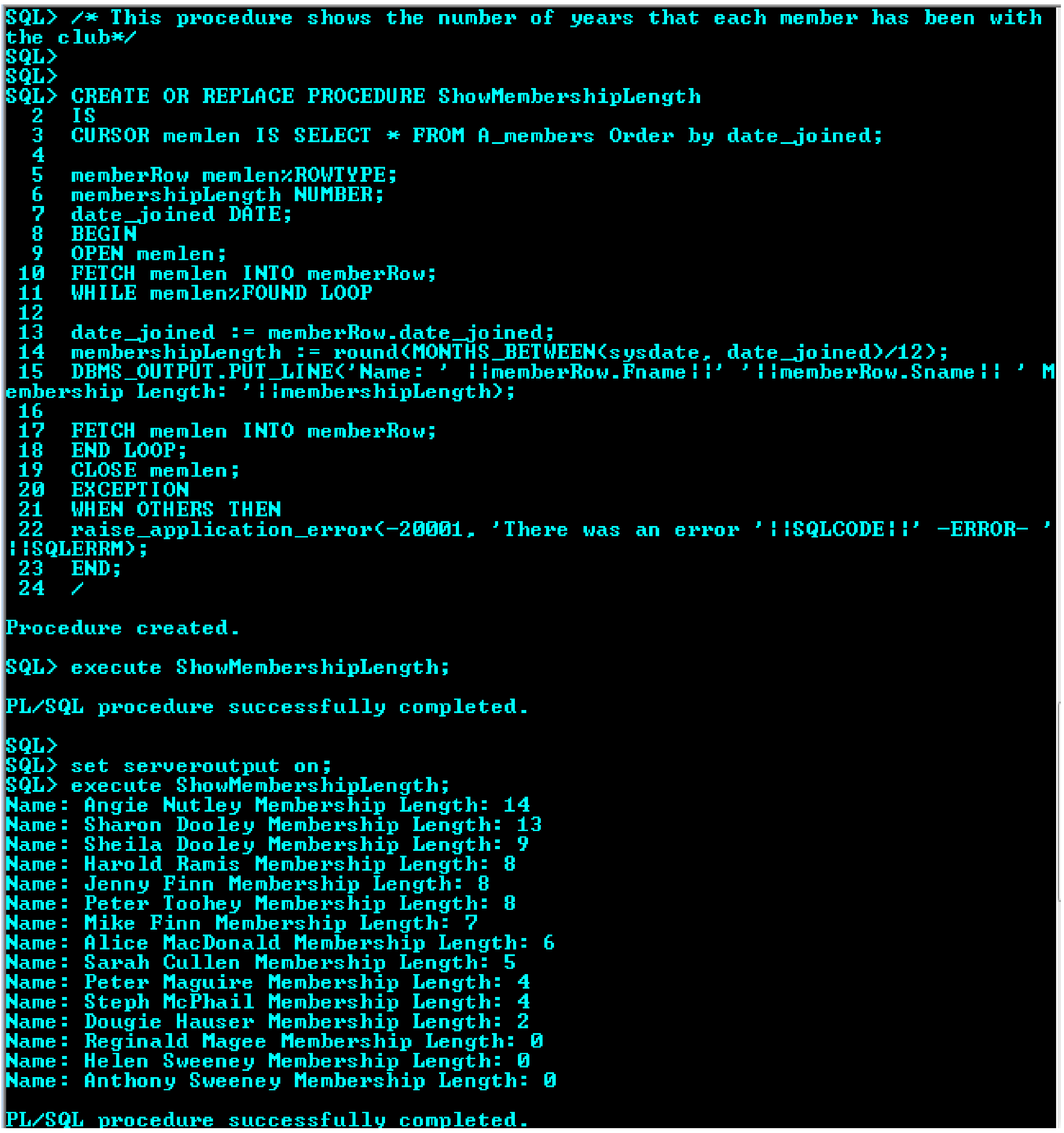
Procedure 6 ShowMembershipLength

This procedure shows the number of years that each member has been with the club

```
CREATE OR REPLACE PROCEDURE ShowMembershipLength

IS

CURSOR memlen IS SELECT * FROM A_members Order by date_joined;


memberRow memlen%ROWTYPE;

membershipLength NUMBER;

date_joined DATE;

BEGIN

OPEN memlen;

FETCH memlen INTO memberRow;

WHILE memlen%FOUND LOOP


date_joined := memberRow.date_joined;

membershipLength := round(MONTHS_BETWEEN(sysdate, date_joined)/12);

DBMS_OUTPUT.PUT_LINE('Name: ' ||memberRow.Fname||' '||memberRow.Sname|| '
Membership Length: '||membershipLength);


FETCH memlen INTO memberRow;

END LOOP;

CLOSE memlen;

EXCEPTION

WHEN OTHERS THEN

raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '||SQLERRM);

END;
```

/

execute ShowMembershipLength;



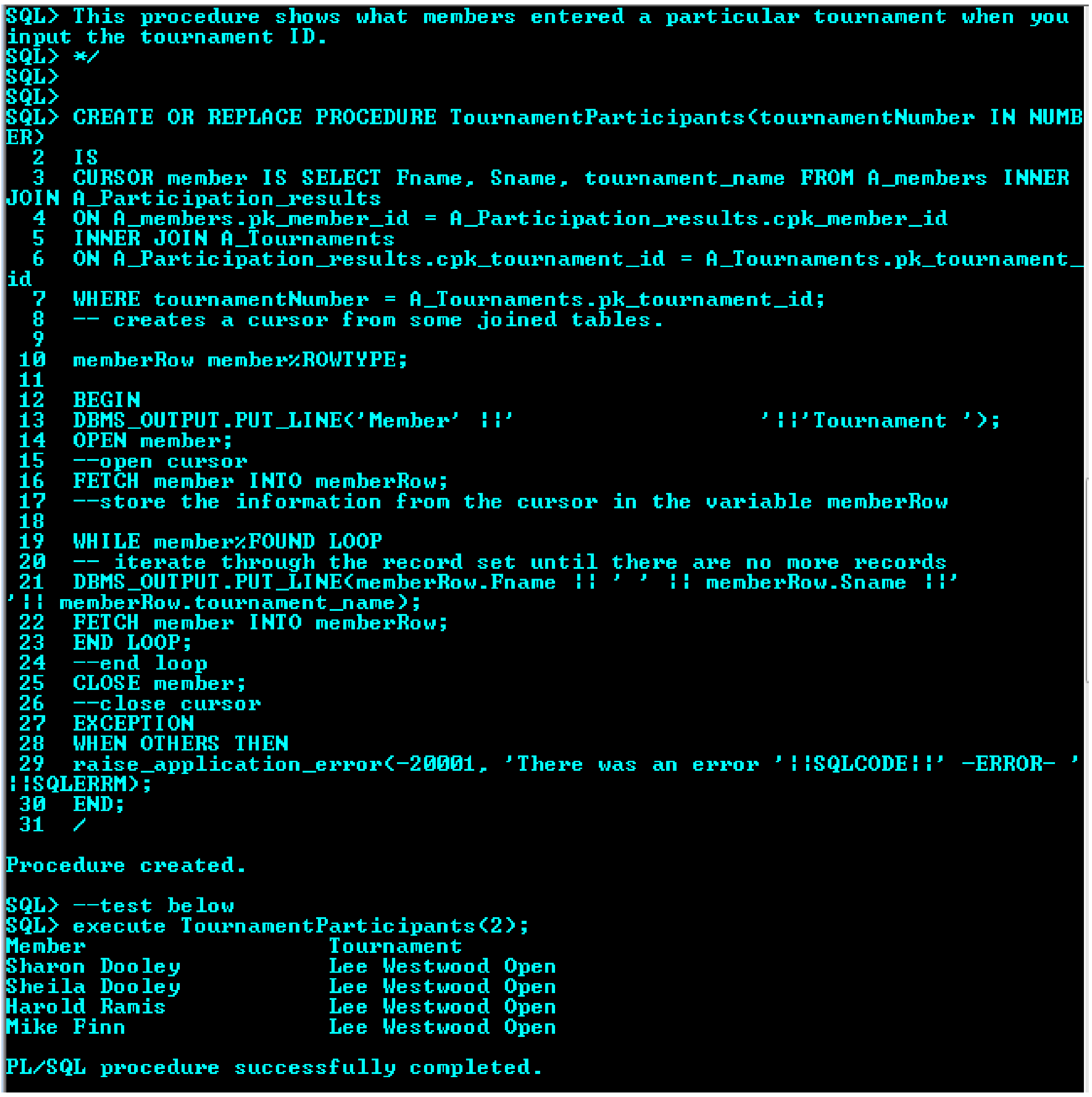
```
SQL> /* This procedure shows the number of years that each member has been with
the club*/
SQL>
SQL>
SQL> CREATE OR REPLACE PROCEDURE ShowMembershipLength
     IS
     CURSOR memlen IS SELECT * FROM A_members Order by date_joined;

     memberRow memlen%ROWTYPE;
     membershipLength NUMBER;
     date_joined DATE;
     BEGIN
     OPEN memlen;
     FETCH memlen INTO memberRow;
     WHILE memlen%FOUND LOOP

     date_joined := memberRow.date_joined;
     membershipLength := round(MONTHS_BETWEEN(sysdate, date_joined)/12);
     DBMS_OUTPUT.PUT_LINE('Name: ' ||memberRow.Fname||' '||memberRow.Sname|| ' M
embership Length: '||membershipLength);

     FETCH memlen INTO memberRow;
     END LOOP;
     CLOSE memlen;
     EXCEPTION
     WHEN OTHERS THEN
     raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '
||SQLERRM);
     END;
     /

Procedure created.

SQL> execute ShowMembershipLength;

PL/SQL procedure successfully completed.

SQL>
SQL> set serveroutput on;
SQL> execute ShowMembershipLength;
Name: Angie Nutley Membership Length: 14
Name: Sharon Dooley Membership Length: 13
Name: Sheila Dooley Membership Length: 9
Name: Harold Ramis Membership Length: 8
Name: Jenny Finn Membership Length: 8
Name: Peter Toohey Membership Length: 8
Name: Mike Finn Membership Length: 7
Name: Alice MacDonald Membership Length: 6
Name: Sarah Cullen Membership Length: 5
Name: Peter Maguire Membership Length: 4
Name: Steph McPhail Membership Length: 4
Name: Dougie Hauser Membership Length: 2
Name: Reginald Magee Membership Length: 0
Name: Helen Sweeney Membership Length: 0
Name: Anthony Sweeney Membership Length: 0

PL/SQL procedure successfully completed.
```

Procedure 7 TournamentParticipants

This procedure shows what members entered a particular tournament when you input the tournament ID.

```
CREATE OR REPLACE PROCEDURE TournamentParticipants(tournamentNumber IN NUMBER)

IS

CURSOR member IS SELECT Fname, Sname, tournament_name FROM A_members INNER JOIN
A_Participation_results

ON A_members.pk_member_id = A_Participation_results.cpk_member_id

INNER JOIN A_Tournaments

ON A_Participation_results.cpk_tournament_id = A_Tournaments.pk_tournament_id

WHERE tournamentNumber = A_Tournaments.pk_tournament_id;

-- creates a cursor from some joined tables.


memberRow member%ROWTYPE;


BEGIN

DBMS_OUTPUT.PUT_LINE('Member' ||'                         '||'Tournament ');

OPEN member;

--open cursor

FETCH member INTO memberRow;

--store the information from the cursor in the variable memberRow


WHILE member%FOUND LOOP

-- iterate through the record set until there are no more records

DBMS_OUTPUT.PUT_LINE(memberRow.Fname || ' ' || memberRow.Sname ||'              '||
memberRow.tournament_name);

FETCH member INTO memberRow;

END LOOP;
```

```
--end loop

CLOSE member;

--close cursor

EXCEPTION

WHEN OTHERS THEN

raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '||SQLERRM);

END;

/

--test below

execute TournamentParticipants(2);
```

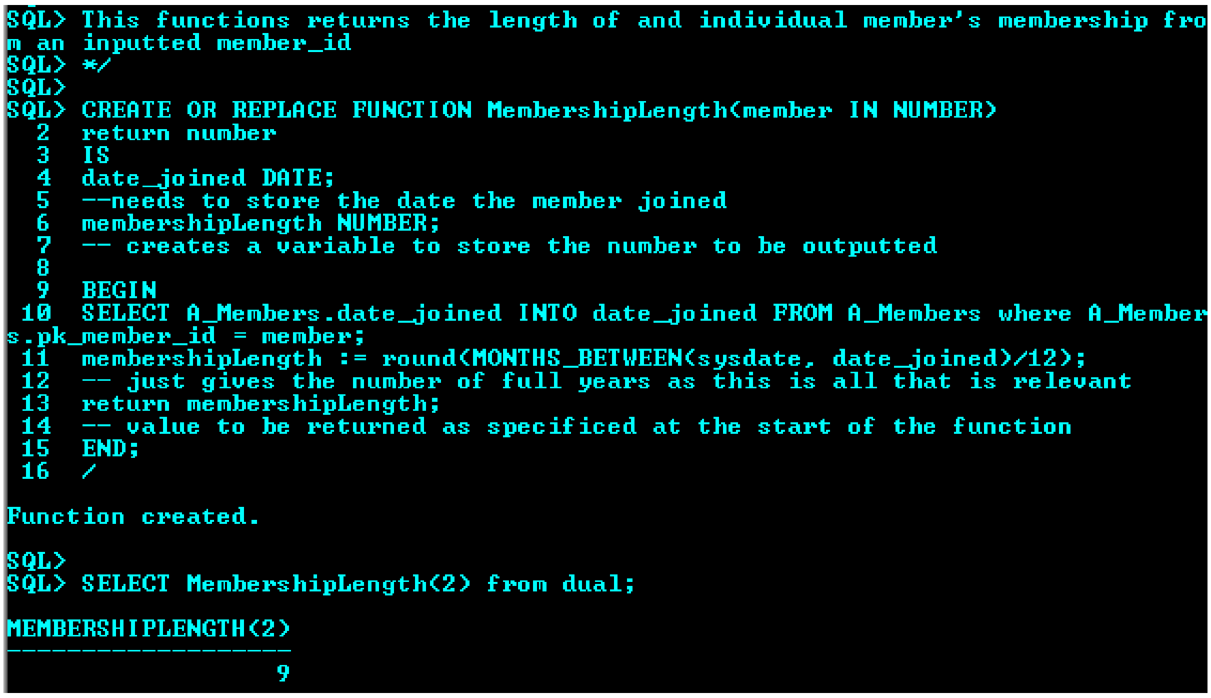


```
SQL> This procedure shows what members entered a particular tournament when you
input the tournament ID.
SQL> */
SQL>
SQL>
SQL> CREATE OR REPLACE PROCEDURE TournamentParticipants(tournamentNumber IN NUMB
ER)
     IS
     CURSOR member IS SELECT Fname, Sname, tournament_name FROM A_members INNER
JOIN A_Participation_results
     ON A_members.pk_member_id = A_Participation_results.cpk_member_id
     INNER JOIN A_Tournaments
     ON A_Participation_results.cpk_tournament_id = A_Tournaments.pk_tournament_
id
     WHERE tournamentNumber = A_Tournaments.pk_tournament_id;
     -- creates a cursor from some joined tables.

     memberRow member%ROWTYPE;

     BEGIN
     DBMS_OUTPUT.PUT_LINE('Member' ||'                   '||'Tournament ');
     OPEN member;
     --open cursor
     FETCH member INTO memberRow;
     --store the information from the cursor in the variable memberRow

     WHILE member%FOUND LOOP
     -- iterate through the record set until there are no more records
     DBMS_OUTPUT.PUT_LINE(memberRow.Fname || ' ' || memberRow.Sname ||'
'|| memberRow.tournament_name);
     FETCH member INTO memberRow;
     END LOOP;
     --end loop
     CLOSE member;
     --close cursor
     EXCEPTION
     WHEN OTHERS THEN
     raise_application_error(-20001, 'There was an error '||SQLCODE||' -ERROR- '
||SQLERRM);
     END;
     /

Procedure created.

SQL> --test below
SQL> execute TournamentParticipants(2);
Member                   Tournament
Sharon Dooley            Lee Westwood Open
Sheila Dooley            Lee Westwood Open
Harold Ramis             Lee Westwood Open
Mike Finn                Lee Westwood Open

PL/SQL procedure successfully completed.
```

2 Functions

This functions returns the length of an individual member's membership from an inputted member_id

```
CREATE OR REPLACE FUNCTION MembershipLength(member IN NUMBER)

return number

IS

date_joined DATE;

--needs to store the date the member joined

membershipLength NUMBER;

-- creates a variable to store the number to be outputted


BEGIN

SELECT A_Members.date_joined INTO date_joined FROM A_Members where
A_Members.pk_member_id = member;

membershipLength := round(MONTHS_BETWEEN(sysdate, date_joined)/12);

-- just gives the number of full years as this is all that is relevant

return membershipLength;

-- value to be returned as specificed at the start of the function

END;

/


SELECT MembershipLength(2) from dual;
```



```
SQL> This functions returns the length of and individual member's membership fro
m an inputted member_id
SQL> */
SQL>
SQL> CREATE OR REPLACE FUNCTION MembershipLength(member IN NUMBER)
     return number
     IS
     date_joined DATE;
     --needs to store the date the member joined
     membershipLength NUMBER;
     -- creates a variable to store the number to be outputted

     BEGIN
     SELECT A_Members.date_joined INTO date_joined FROM A_Members where A_Member
s.pk_member_id = member;
     membershipLength := round(MONTHS_BETWEEN(sysdate, date_joined)/12);
     -- just gives the number of full years as this is all that is relevant
     return membershipLength;
     -- value to be returned as specificed at the start of the function
     END;
     /

Function created.

SQL>
SQL> SELECT MembershipLength(2) from dual;

MEMBERSHIPLENGTH(2)
-------------------
                  9
```

This function returns the winner of a tournament when the tournament_id is supplied

as a parameter

CREATE OR REPLACE FUNCTION tournamentWinner(tournament IN NUMBER)

return NVARCHAR2

IS

lowestScore NUMBER;

--needs to store the date the member joined

nameOfWinner NVARCHAR2(30);

--value to be returned

BEGIN

SELECT MIN(SCORE) INTO lowestScore FROM A_Participation_Results WHERE cpk_tournament_id = tournament;

--stores the lowest score from the tournament requested

SELECT Fname || ' ' || Sname AS fullname INTO nameOfWinner FROM A_members

WHERE pk_member_id = (SELECT cpk_member_id FROM A_Participation_Results

WHERE score =lowestScore AND cpk_tournament_id = tournament);

-- A subquery to find the details of the member who has the lowest score reorded in A_Participation_Results

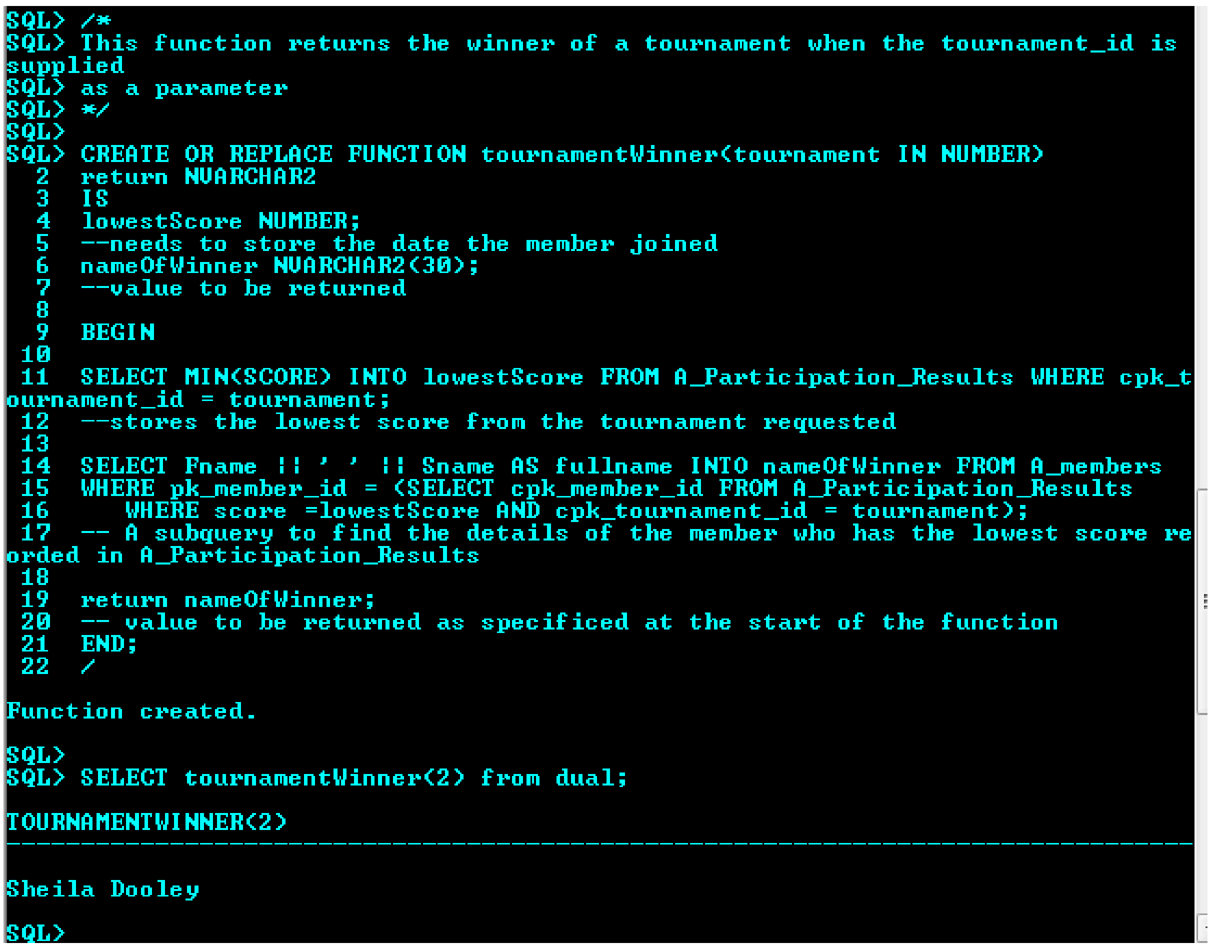
return nameOfWinner;

-- value to be returned as specificed at the start of the function

END;

/

SELECT tournamentWinner(2) from dual;



```
SQL> /*
SQL> This function returns the winner of a tournament when the tournament_id is
supplied
SQL> as a parameter
SQL> */
SQL>
SQL> CREATE OR REPLACE FUNCTION tournamentWinner(tournament IN NUMBER)
     return NVARCHAR2
     IS
     lowestScore NUMBER;
     --needs to store the date the member joined
     nameOfWinner NVARCHAR2(30);
     --value to be returned

     BEGIN

     SELECT MIN(SCORE) INTO lowestScore FROM A_Participation_Results WHERE cpk_t
ournament_id = tournament;
     --stores the lowest score from the tournament requested

     SELECT Fname || ' ' || Sname AS fullname INTO nameOfWinner FROM A_members
     WHERE pk_member_id = (SELECT cpk_member_id FROM A_Participation_Results
        WHERE score =lowestScore AND cpk_tournament_id = tournament);
     -- A subquery to find the details of the member who has the lowest score re
orded in A_Participation_Results

     return nameOfWinner;
     -- value to be returned as specificed at the start of the function
     END;
     /

Function created.

SQL>
SQL> SELECT tournamentWinner(2) from dual;

TOURNAMENTWINNER(2)
--------------------------------------------------------------------------------

Sheila Dooley

SQL>
```

3 Triggers

This trigger alerts us when a member's handicap goes below 5 and reminds us to send them a congratulatory certificate. It only alerts if the player had a handicap over 5 before the update.

```
CREATE OR REPLACE TRIGGER trig_lowHandicap

AFTER UPDATE ON A_Members

FOR EACH ROW

BEGIN

IF :new.handicap < 5 AND :old.Handicap >= 5 THEN

DBMS_OUTPUT.PUT_LINE(:new.Fname || ' ' || :new.Sname ||' has received a new handicap of '||
:new.handicap || '. Send them a congratulatory certificate to : ');

DBMS_OUTPUT.PUT_LINE(:new.address_line_1);

DBMS_OUTPUT.PUT_LINE(:new.address_line_2);

DBMS_OUTPUT.PUT_LINE(:new.address_line_3);

END IF;

END;

/




--Test

UPDATE A_Members

SET handicap = 6

WHERE Fname = 'Mike';

-- updates Mike's handicap to 6 for to set up the next tests

UPDATE A_Members

SET handicap = 4

WHERE Fname = 'Mike';

-- outputs the alert message
```

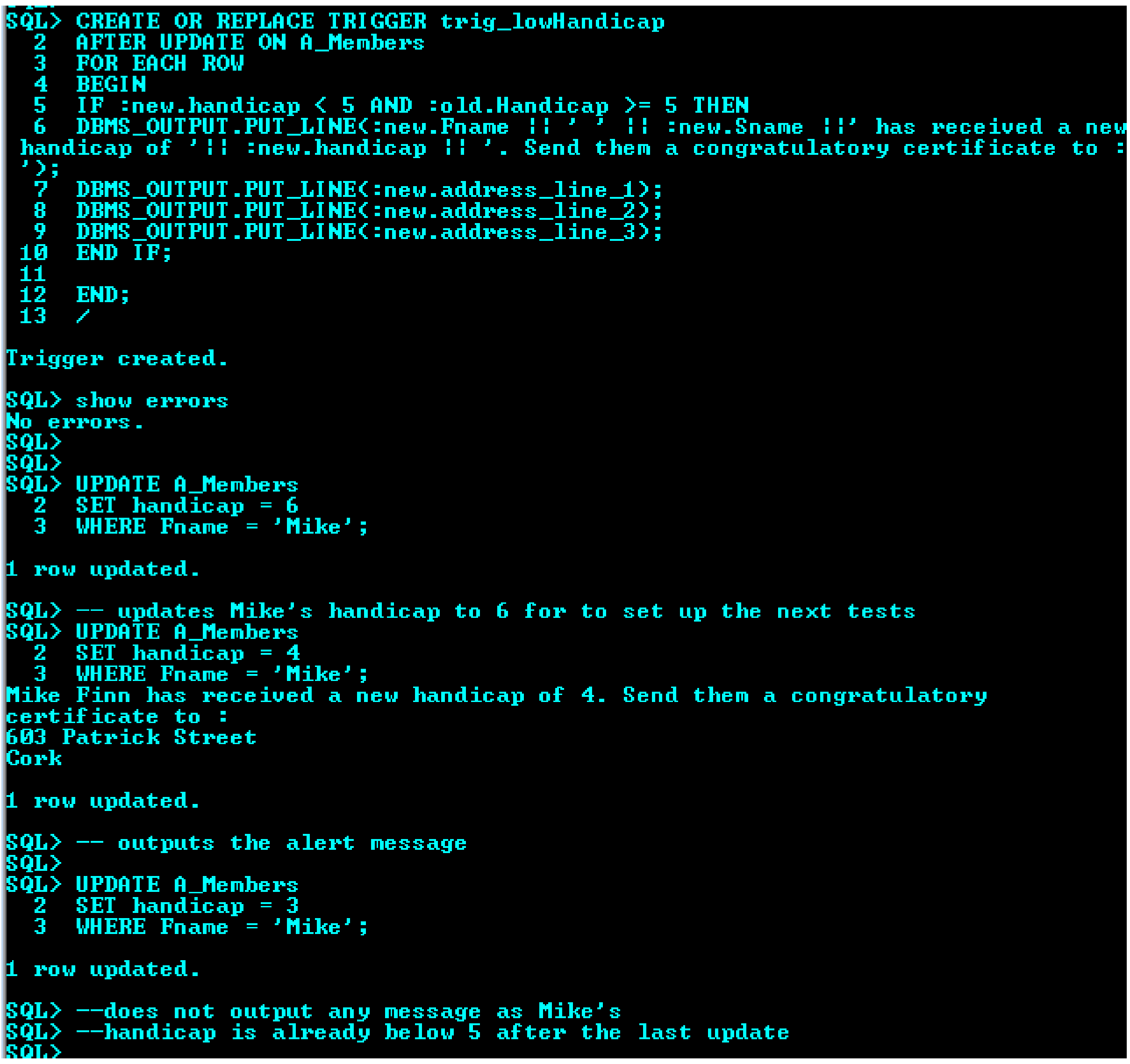
UPDATE A_Members

SET handicap = 3

WHERE Fname = 'Mike';

--does not output any message as Mike's

--handicap is already below 5 after the last update



```
SQL> CREATE OR REPLACE TRIGGER trig_lowHandicap
     AFTER UPDATE ON A_Members
     FOR EACH ROW
     BEGIN
     IF :new.handicap < 5 AND :old.Handicap >= 5 THEN
     DBMS_OUTPUT.PUT_LINE(:new.Fname || ' ' || :new.Sname ||' has received a new
 handicap of '|| :new.handicap || '. Send them a congratulatory certificate to :
 ');
     DBMS_OUTPUT.PUT_LINE(:new.address_line_1);
     DBMS_OUTPUT.PUT_LINE(:new.address_line_2);
     DBMS_OUTPUT.PUT_LINE(:new.address_line_3);
     END IF;

     END;
     /

Trigger created.

SQL> show errors
No errors.
SQL>
SQL>
SQL> UPDATE A_Members
     SET handicap = 6
     WHERE Fname = 'Mike';

1 row updated.

SQL> -- updates Mike's handicap to 6 for to set up the next tests
SQL> UPDATE A_Members
     SET handicap = 4
     WHERE Fname = 'Mike';
Mike Finn has received a new handicap of 4. Send them a congratulatory
certificate to :
603 Patrick Street
Cork

1 row updated.

SQL> -- outputs the alert message
SQL>
SQL> UPDATE A_Members
     SET handicap = 3
     WHERE Fname = 'Mike';

1 row updated.

SQL> --does not output any message as Mike's
SQL> --handicap is already below 5 after the last update
SQL>
```

This trigger checks all payments in case a member without a handicap tries to pay for a tournament.

This is strictly not allowed at the club.

```
CREATE OR REPLACE TRIGGER noHandicap

BEFORE INSERT ON A_Payment

FOR EACH ROW

DECLARE

Fname NVARCHAR2(30);

Sname NVARCHAR2(30);

handicap NUMBER;


BEGIN

SELECT A_Members.handicap INTO handicap FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;

SELECT A_Members.Fname INTO Fname FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;

SELECT A_Members.Sname INTO Sname FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;


if :new.category_id in (11, 12, 13) THEN

-- 11,12,13 are the category types of tournaments

  if handicap = -1 THEN

  -- -1 is the default for no handicap

        raise_application_error( -20001, Fname || ' ' || Sname ||

                        ' has attempted to pay for a tournament without a handicap');

    END IF;

end if;
```

END;

/

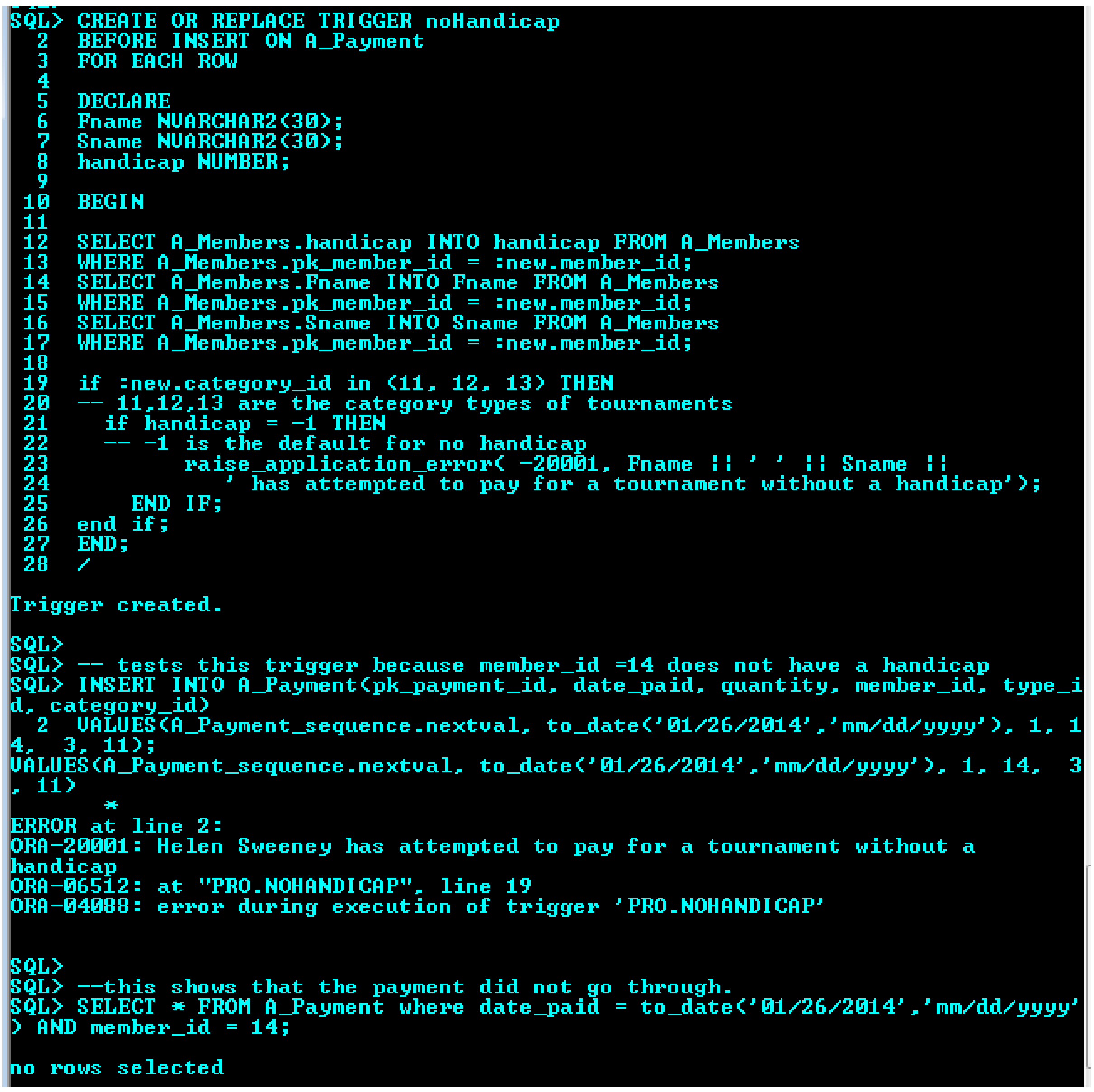
-- tests this trigger because member_id =14 does not have a handicap

INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_id, category_id)

VALUES(A_Payment_sequence.nextval, to_date('01/26/2014','mm/dd/yyyy'), 1, 14, 3, 11);

--this shows that the payment did not go through.

SELECT * FROM A_Payment where date_paid = to_date('01/26/2014','mm/dd/yyyy') AND member_id = 14;



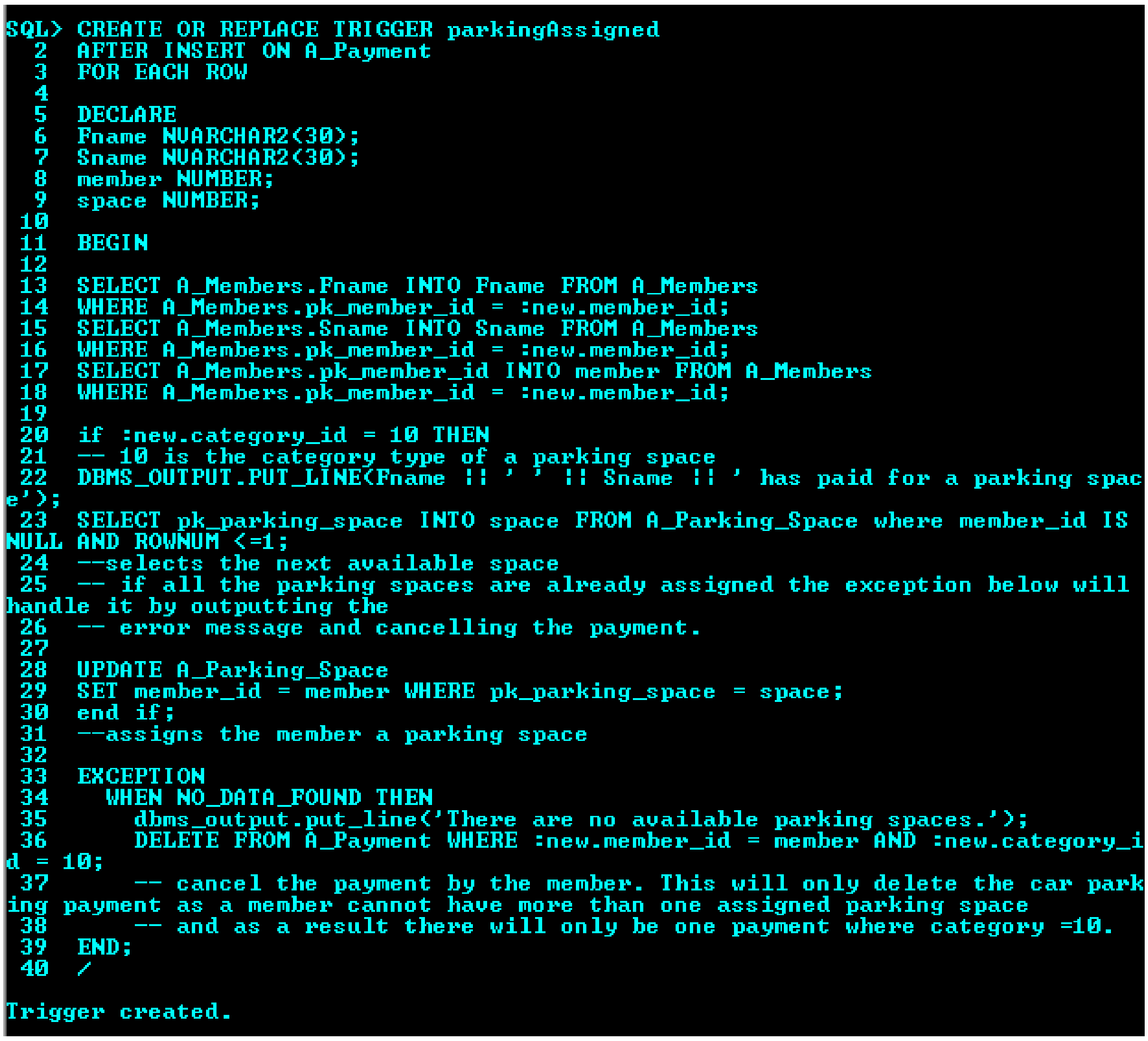
```
SQL> CREATE OR REPLACE TRIGGER noHandicap
  2  BEFORE INSERT ON A_Payment
  3  FOR EACH ROW
  4
  5  DECLARE
  6  Fname NVARCHAR2(30);
  7  Sname NVARCHAR2(30);
  8  handicap NUMBER;
  9
 10  BEGIN
 11
 12  SELECT A_Members.handicap INTO handicap FROM A_Members
 13  WHERE A_Members.pk_member_id = :new.member_id;
 14  SELECT A_Members.Fname INTO Fname FROM A_Members
 15  WHERE A_Members.pk_member_id = :new.member_id;
 16  SELECT A_Members.Sname INTO Sname FROM A_Members
 17  WHERE A_Members.pk_member_id = :new.member_id;
 18
 19  if :new.category_id in (11, 12, 13) THEN
 20  -- 11,12,13 are the category types of tournaments
 21    if handicap = -1 THEN
 22    -- -1 is the default for no handicap
 23          raise_application_error( -20001, Fname || ' ' || Sname ||
 24             ' has attempted to pay for a tournament without a handicap');
 25      END IF;
 26  end if;
 27  END;
 28  /

Trigger created.

SQL>
SQL> -- tests this trigger because member_id =14 does not have a handicap
SQL> INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_i
d, category_id)
  2  VALUES(A_Payment_sequence.nextval, to_date('01/26/2014','mm/dd/yyyy'), 1, 1
4,  3, 11);
VALUES(A_Payment_sequence.nextval, to_date('01/26/2014','mm/dd/yyyy'), 1, 14,  3
, 11)
              *
ERROR at line 2:
ORA-20001: Helen Sweeney has attempted to pay for a tournament without a
handicap
ORA-06512: at "PRO.NOHANDICAP", line 19
ORA-04088: error during execution of trigger 'PRO.NOHANDICAP'


SQL>
SQL> --this shows that the payment did not go through.
SQL> SELECT * FROM A_Payment where date_paid = to_date('01/26/2014','mm/dd/yyyy'
) AND member_id = 14;

no rows selected
```

A trigger created to show when a member is assigned a car parking space.

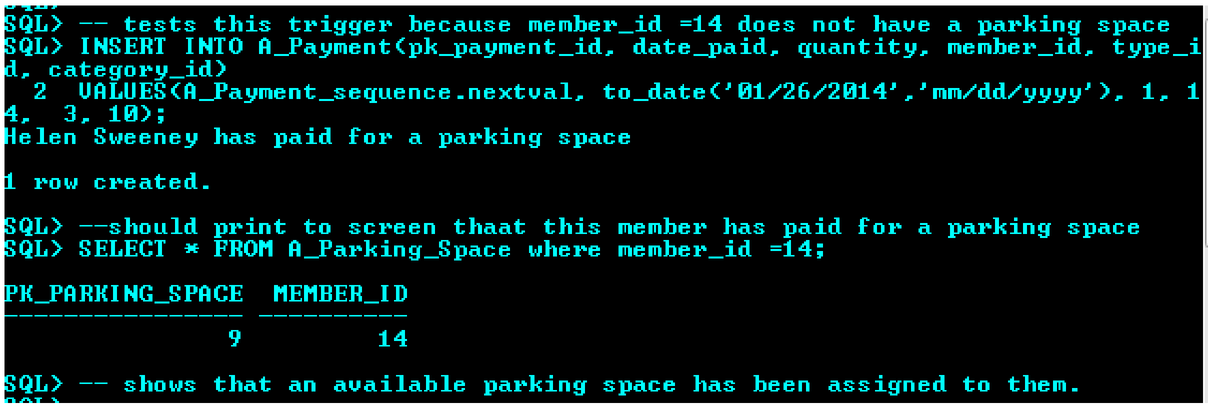
This will only occur on UPDATE as all available car parking sapces are already stored in the database.

```
cl scr;

CREATE OR REPLACE TRIGGER parkingAssigned

AFTER INSERT ON A_Payment

FOR EACH ROW


DECLARE

Fname NVARCHAR2(30);

Sname NVARCHAR2(30);

member NUMBER;

space NUMBER;


BEGIN


SELECT A_Members.Fname INTO Fname FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;

SELECT A_Members.Sname INTO Sname FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;

SELECT A_Members.pk_member_id INTO member FROM A_Members

WHERE A_Members.pk_member_id = :new.member_id;


if :new.category_id = 10 THEN

-- 10 is the category type of a parking space

DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a parking space');

SELECT pk_parking_space INTO space FROM A_Parking_Space where member_id IS NULL AND
ROWNUM <=1;
```

```
--selects the next available space

-- if all the parking spaces are already assigned the exception below will handle it by outputting the

-- error message and cancelling the payment.


UPDATE A_Parking_Space

SET member_id = member WHERE pk_parking_space = space;

end if;

--assigns the member a parking space


EXCEPTION

  WHEN NO_DATA_FOUND THEN

    dbms_output.put_line('There are no available parking spaces.');

    DELETE FROM A_Payment WHERE :new.member_id = member AND :new.category_id = 10;

    -- cancel the payment by the member. This will only delete the car parking payment as a member
cannot have more than one assigned parking space

    -- and as a result there will only be one payment where category =10.

END;

/

show errors


-- tests this trigger because member_id =14 does not have a parking space

INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_id, category_id)

VALUES(A_Payment_sequence.nextval, to_date('01/26/2014','mm/dd/yyyy'), 1, 14,  3, 10);

--should print to screen thaat this member has paid for a parking space

SELECT * FROM A_Parking_Space where member_id =14;

-- shows that an available parking space has been assigned to them.
```



```
SQL> CREATE OR REPLACE TRIGGER parkingAssigned
  2  AFTER INSERT ON A_Payment
  3  FOR EACH ROW
  4
  5  DECLARE
  6  Fname NVARCHAR2(30);
  7  Sname NVARCHAR2(30);
  8  member NUMBER;
  9  space NUMBER;
 10
 11  BEGIN
 12
 13  SELECT A_Members.Fname INTO Fname FROM A_Members
 14  WHERE A_Members.pk_member_id = :new.member_id;
 15  SELECT A_Members.Sname INTO Sname FROM A_Members
 16  WHERE A_Members.pk_member_id = :new.member_id;
 17  SELECT A_Members.pk_member_id INTO member FROM A_Members
 18  WHERE A_Members.pk_member_id = :new.member_id;
 19
 20  if :new.category_id = 10 THEN
 21  -- 10 is the category type of a parking space
 22  DBMS_OUTPUT.PUT_LINE(Fname || ' ' || Sname || ' has paid for a parking spac
e');
 23  SELECT pk_parking_space INTO space FROM A_Parking_Space where member_id IS
NULL AND ROWNUM <=1;
 24  --selects the next available space
 25  -- if all the parking spaces are already assigned the exception below will
handle it by outputting the
 26  -- error message and cancelling the payment.
 27
 28  UPDATE A_Parking_Space
 29  SET member_id = member WHERE pk_parking_space = space;
 30  end if;
 31  --assigns the member a parking space
 32
 33  EXCEPTION
 34    WHEN NO_DATA_FOUND THEN
 35      dbms_output.put_line('There are no available parking spaces.');
 36      DELETE FROM A_Payment WHERE :new.member_id = member AND :new.category_i
d = 10;
 37      -- cancel the payment by the member. This will only delete the car park
ing payment as a member cannot have more than one assigned parking space
 38      -- and as a result there will only be one payment where category =10.
 39  END;
 40  /

Trigger created.
```



```
SQL> -- tests this trigger because member_id =14 does not have a parking space
SQL> INSERT INTO A_Payment(pk_payment_id, date_paid, quantity, member_id, type_i
d, category_id)
  2  VALUES(A_Payment_sequence.nextval, to_date('01/26/2014','mm/dd/yyyy'), 1, 1
4,  3, 10);
Helen Sweeney has paid for a parking space

1 row created.

SQL> --should print to screen thaat this member has paid for a parking space
SQL> SELECT * FROM A_Parking_Space where member_id =14;

PK_PARKING_SPACE  MEMBER_ID
---------------- ----------
               9         14

SQL> -- shows that an available parking space has been assigned to them.
```

Weaknesses

While I think it satisfies the requirements for this project I would not consider this database even close to being finished. I think that the database could benefit from a lot more triggers and procedures which are outside the scope of this project.

I guess the biggest weakness was the original design document. Even though we were told several times that the ERD is the most important thing to get right it wasn't until I had attempted to create this database a few times that I truly understood this. I thought that I had to model it on the examples from class and as such I made it a little too vague. Hopefully I have cleared up any issues in the introduction of this document and the revised ERD.

If I was recreating the database I would have the Lessons and Tournaments linked to Payments. I was able to get around this in my database by having the payment categories field but I found it a bit messy at times and think the database would benefit from these relationships.

I would have liked to do more error checking on all of the procedures as obviously there is a lot that could go wrong. Hopefully I have shown that I understand the concepts and could apply them more thoroughly given extra time.

Given that this is a database project for this unit and not a real life situation I should have reduced the number of insert statements. I believed that it would aid me in creating more substantially results from my queries. In the end it hindered me because some of my result sets were so large that the use of screenshots to illustrate them became difficult.