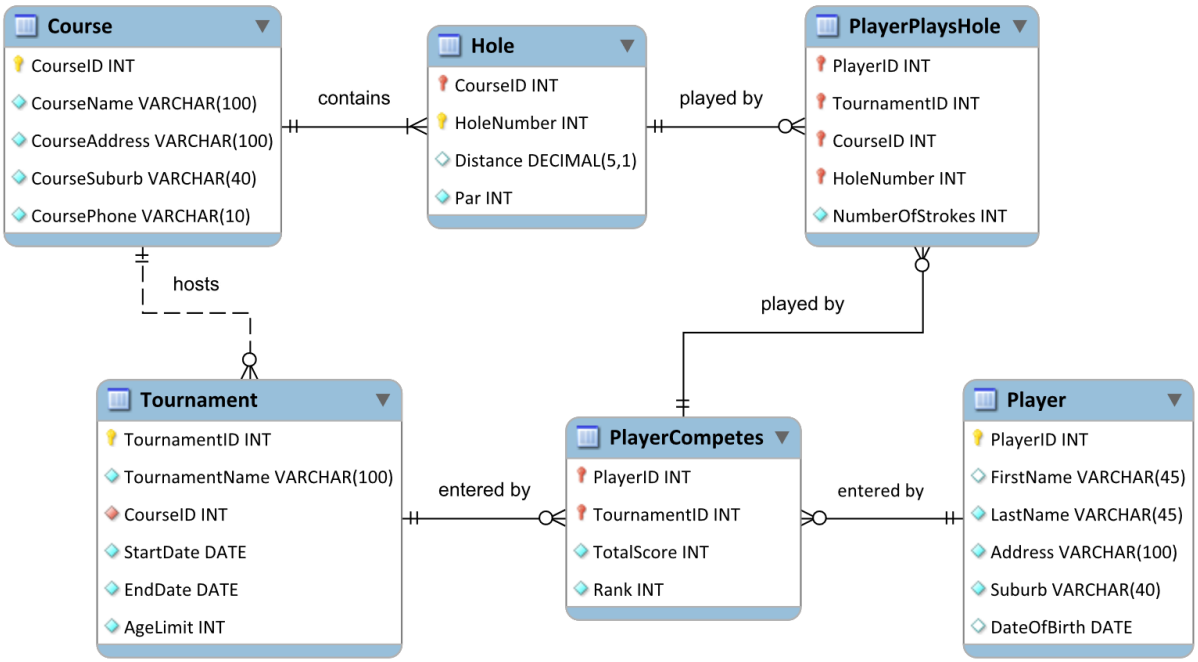
The University of Melbourne
School of Computing and Information Systems
INFO20003 Database Systems

SQL: Junior golf in Victoria

Every year, golf clubs and sporting organisations around Victoria run a series of tournaments for junior (under-18) golfers. Golf Victoria, the peak body for the sport in Victoria, has decided to implement a central database to record participants and scores in all these junior tournaments, to help them award prizes to deserving players at the end of the season, and also to track participation levels in competitive junior golf for strategic planning purposes.

Golf Victoria's database designers have developed the following physical schema:



If you're not familiar with golf, here's what you need to know:

- A golf course is made up of 9 or 18 holes. To play a round of golf, you need to successfully complete every hole on the course.
- To complete a hole, you need to hit the ball into the hole using the fewest "strokes" (number of hits) you can. Each hole has a "par", which is the typical number of strokes most people should take to complete the hole.
- Your "score" for a hole is worked out as your number of strokes minus that hole's par. For example, if you complete a par 3 hole using two strokes, your score is –1 (read as "1 under par"). Lower scores are better in golf.

In these questions, we assume that tournaments only have one round, and that no tournaments carry on over the New Year. Write a *single* SQL statement for each question – subqueries are allowed from question 21 onwards.

DDL: Creating tables

Write MySQL DDL statements to create the Course, Hole and Tournament tables in the database. Think carefully about the order in which you create the tables.

DML part 1: Projection, selection and joins (no grouping or subqueries)

1. Write a query to display the tournament name, start date and end date of the tournament with ID 36.

2. Write a query to find the golf course name and hole number for every par 3 hole in the database. Order the results alphabetically by course name.

3. Write a query that lists the first and last name, suburb, and total score of every player who competed in the "Box Hill Junior Open" tournament. Display the lowest-scoring players first.

4. Write a query to show the first and last names of **all** players who live in Reservoir, the names of any tournaments they have competed in, and their rank in each tournament.

5. Write a query to return the names of tournaments, played at courses in the suburb of Heatherton, where Bella Sandbury ranked in the top ten.
   Hint: Use `SELECT DISTINCT` *to ensure your answer doesn't contain duplicate rows.*

6. Write a query that returns the first and last name of players who have made a hole-in-one (the NumberOfStrokes column is 1), the name of the course they were playing at, and the hole number. Leave out holes-in-one where the distance is known to be less than 100 m.

7. Write a query to find Bella Sandbury's *score* (number of strokes minus par) on every hole she played during the first six months of the year 2024. Return the course name, hole number and score, sorting the results by score from highest to lowest.

8. Rewrite the query from question 7 so it shows Bella's score on every hole she played when she was aged 15.
   *Hint: To add on **n** years to the date of birth, use* `(DateOfBirth + INTERVAL n YEAR)`
   *Notice that the word "YEAR" is **not** plural in this syntax.*

9. Write a query to list the first and last names of players who won a tournament even though their score was more than +2 on at least one hole.

10. Write a query to show the course name and hole number for the longest-distance par 3 hole on any course located in Kew East, Balwyn North or Ivanhoe. Assume there is only one hole that fits this description.
    Hint: You do not need to use subqueries to answer this question!

DML part 2: Aggregate functions and grouping (no subqueries)

11. Write a query to show the number of players in each suburb. Order the results by suburb name.

12. Write a query to show the player ID, average total score and highest rank of every player. Order the results by average total score, lowest to highest.
    Hint: The "highest" possible rank is 1. Which aggregate function should you use?

13. Write a query to print the course name, address, suburb, number of holes, and total hole distance of every golf course.

14. Write a query that returns the first and last name of **all** players, the total number of holes played by that player over all time, and the total number of strokes they took.

15. Write a query to show the number of under-16 tournaments (AgeLimit = 16) each player has taken part in. Output the player's first and last name and number of tournaments, sorting results from most tournaments to least.

16. Write a query to show the players who have played in at least five tournaments. Output the players' first and last names and the number of tournaments, sorting results from most tournaments to least.

17. Write a query to show the first and last names of players who have played in exactly three under-16 tournaments.

18. Write a query to find the first and last names of players who have scored below par more than once on the 18th hole at any golf course. Sort the players by age, from youngest to oldest.
    Hint: "Below par" means their score is less than zero, that is, their number of strokes is less than that hole's par.

19. Write a query to show the start and end dates of the 2024 junior season. (The season starts when the first tournament of the year starts and ends when the last tournament of the year ends.)
    Hint: Output one row with two columns. Also have a go at generating a single column with two rows – it's a bit trickier that way.

20. Sometimes, amid the chaos in the golf clubhouse at the end of a tournament, the wrong total scores get recorded in the PlayerCompetes table. Write a query to identify these incorrect TotalScore values. Output the player ID, first and last name, tournament name, the incorrect TotalScore value, and the correct total score for the player in that tournament. Order by the discrepancy between the stored TotalScore and the correct score, placing the worst discrepancies at the top.
    Hint: Think carefully about what column(s) should appear in your GROUP BY clause!

DML part 3: Subqueries

21. Using a subquery approach, write a query that shows the names of tournaments played at courses in Bundoora.

22. Write a query that shows the first and last names of players who have never played hole 18 at the "Kooringal Golf Club".

23. Write a query to find the first and last names and addresses of players who live in the same suburb as Bella Sandbury.

24. Write a query to display the first and last names of players who have played on every golf course.

25. Write a query that returns each player's first and last name, the name(s) of the tournament(s) where they achieved their best rank, and the rank they achieved.

26. Write a query to display the names of courses on which Bella Sandbury has played every hole at least once.

27. Write a query to name courses that hosted tournaments in 2024 but not in 2025.

28. Write a query to list the first and last names of players who live at the same residence (address and suburb) as at least one other player.

29. Write a query to list players who scored below par on every hole in a tournament but did not win. Display the players' first and last name, tournament name, and their total score.

30. Write a query to display the first and last names of players who have competed against Bella Sandbury at least twice.

DML part 4: CRUD

31. Write a query to set hole 10 at the "Sunshine Golf Club" to par 4.

32. Write a query to add a player called Olivia Fairbanks, who lives at 4 Mayfair Avenue, Huntingdale. She was born on 5 January 2007. Her ID number is 394.

33. The Sunshine Golf Club was accidentally added a second time to the Course table under the name "Sunshine Golf Course". Write a query to delete the erroneous entry.

34. Due to a scoring error during the "Medway Junior Open" tournament, Bella Sandbury was recorded as having taken one more stroke than she actually did on every hole. Write a query to correct Bella's scores.

35. Write a query to remove all courses from the database where no tournaments have been played *and* which have less than 9 holes recorded in the system.

DML part 5: Bringing it all together

36. To be eligible to enter a tournament, a player must be under the age limit on the day the tournament starts. Write a query to identify tournaments in which ineligible players were accepted to compete, listing the tournament name and how many ineligible competitors were accepted. Sort the output in descending order of number of ineligible competitors.

37. Rewrite the query from question 23 (finding players who live in the same suburb as Bella Sandbury) without using a subquery.

38. Bella Sandbury's golf game is maturing fast. Lately she's been proving tough to beat. Write a query to find the first and last names of players who have played against Bella at least once but have never beaten her.

39. In question 20, you wrote a query to identify incorrect TotalScore values. Now write a query to identify incorrect Rank values in the PlayerCompetes table. Output the player ID, first and last name, tournament name, the incorrect Rank value, and the player's actual rank in that tournament.

40. List the first and last names of players who have played in exactly the same set of tournaments as at least one other player.

Bonus DML challenge question

41. A player is "on top of the leaderboard" for a particular hole during a tournament if they have the lowest cumulative score over the holes played up to and including that hole. Write a query to find the first and last name(s) of the player(s) who were on top of the leaderboard, and their cumulative score, for each hole played during the most recent tournament at the "Royal Park Golf Course".

Solutions – DDL

```
CREATE TABLE Course (
  CourseID INT NOT NULL,
  CourseName VARCHAR(100) NOT NULL,
  CourseAddress VARCHAR(100) NOT NULL,
  CourseSuburb VARCHAR(40) NOT NULL,
  CoursePhone VARCHAR(10) NOT NULL,
  PRIMARY KEY (CourseID)
);

CREATE TABLE Hole (
  CourseID INT NOT NULL,
  HoleNumber INT NOT NULL,
  Distance DECIMAL(5,1),
  Par INT NOT NULL,
  PRIMARY KEY (CourseID, HoleNumber),
  FOREIGN KEY (CourseID) REFERENCES Course (CourseID)
);

CREATE TABLE Tournament (
  TournamentID INT NOT NULL,
  TournamentName VARCHAR(100) NOT NULL,
  CourseID INT NOT NULL,
  StartDate DATE NOT NULL,
  EndDate DATE NOT NULL,
  AgeLimit INT NOT NULL,
  PRIMARY KEY (TournamentID),
  FOREIGN KEY (CourseID) REFERENCES Course (CourseID)
);
```

The CREATE TABLE statements for the remaining tables are provided here for your interest:

```
CREATE TABLE Player (
  PlayerID INT NOT NULL,
  FirstName VARCHAR(45),
  LastName VARCHAR(45) NOT NULL,
  Address VARCHAR(100) NOT NULL,
  Suburb VARCHAR(40) NOT NULL,
  DateOfBirth DATE,
  PRIMARY KEY (PlayerID)
);

CREATE TABLE PlayerCompetes (
  PlayerID INT NOT NULL,
  TournamentID INT NOT NULL,
  TotalScore INT NOT NULL,
  Rank INT NOT NULL,
  PRIMARY KEY (PlayerID, TournamentID),
  FOREIGN KEY (PlayerID) REFERENCES Player (PlayerID),
  FOREIGN KEY (TournamentID) REFERENCES Tournament (TournamentID)
);
```

```
CREATE TABLE PlayerPlaysHole (
  PlayerID INT NOT NULL,
  TournamentID INT NOT NULL,
  CourseID INT NOT NULL,
  HoleNumber INT NOT NULL,
  NumberOfStrokes INT NOT NULL,
  PRIMARY KEY (PlayerID, TournamentID, CourseID, HoleNumber),
  FOREIGN KEY (PlayerID, TournamentID)
    REFERENCES PlayerCompetes (PlayerID, TournamentID),
  FOREIGN KEY (CourseID, HoleNumber)
    REFERENCES Hole (CourseID, HoleNumber)
);
```

Solutions – DML part 1

1.
```
SELECT TournamentName, StartDate, EndDate
FROM Tournament
WHERE TournamentID = 36;
```

2. *You can complete simple joins like this one using several different SQL syntaxes:*
```
SELECT CourseName, HoleNumber
FROM Course NATURAL JOIN Hole
WHERE Par = 3
ORDER BY CourseName;
```
or…
```
SELECT CourseName, HoleNumber
FROM Course INNER JOIN Hole ON Course.CourseID = Hole.CourseID
WHERE Par = 3
ORDER BY CourseName;
```
or…
```
SELECT CourseName, HoleNumber
FROM Course INNER JOIN Hole USING (CourseID)
WHERE Par = 3
ORDER BY CourseName;
```
or…
```
SELECT CourseName, HoleNumber
FROM Course, Hole
WHERE Course.CourseID = Hole.CourseID
  AND Par = 3
ORDER BY CourseName;
```
Throughout the rest of these solutions, the NATURAL JOIN approach will be used.

3.
```
SELECT FirstName, LastName, Suburb, TotalScore
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN Tournament
WHERE TournamentName = 'Box Hill Junior Open'
ORDER BY TotalScore;
```

4. *Because **all** players living in Reservoir must be shown, even if they did not compete in a tournament, we need to use an outer join:*
```
SELECT FirstName, LastName, TournamentName, Rank
FROM Player
  LEFT JOIN PlayerCompetes ON Player.PlayerID = PlayerCompetes.PlayerID
  NATURAL JOIN Tournament
WHERE Suburb = 'Reservoir';
```

5.
```sql
SELECT DISTINCT TournamentName
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN Tournament
  NATURAL JOIN Course
WHERE CourseSuburb = 'Heatherton'
  AND PlayerFirstName = 'Bella'
  AND PlayerLastName = 'Sandbury'
  AND Rank <= 10;
```

6. *Even though the Player and PlayerPlaysHole tables are not directly related, the PlayerID foreign key on PlayerPlaysHole means we can natural join the two tables.*
There's another trick here – the Distance column can be NULL. The question asks us to include holes where the distance is not known, so a separate IS NULL condition must be included.
```sql
SELECT DISTINCT FirstName, LastName, CourseName, HoleNumber
FROM Player NATURAL JOIN PlayerPlaysHole NATURAL JOIN Hole
  NATURAL JOIN Course
WHERE NumberOfStrokes = 1
  AND (Distance >= 100 OR Distance IS NULL);
```

7. *We don't know **exactly** when she played each hole, but it must have been between the StartDate and EndDate of the tournament…*
```sql
SELECT CourseName, HoleNumber, (NumberOfStrokes - Par) AS Score
FROM Player NATURAL JOIN PlayerPlaysHole NATURAL JOIN Hole
  NATURAL JOIN Course NATURAL JOIN Tournament
WHERE StartDate >= '2024-01-01'
  AND EndDate <= '2024-06-30'
  AND FirstName = 'Bella'
  AND LastName = 'Sandbury'
ORDER BY Score DESC;
```

8.
```sql
SELECT CourseName, HoleNumber, (NumberOfStrokes - Par) AS Score
FROM Player NATURAL JOIN PlayerPlaysHole NATURAL JOIN Hole
  NATURAL JOIN Course NATURAL JOIN Tournament
WHERE StartDate >= (DateOfBirth + INTERVAL 15 YEAR)
  AND EndDate < (DateOfBirth + INTERVAL 16 YEAR)
  AND FirstName = 'Bella'
  AND LastName = 'Sandbury'
ORDER BY Score DESC;
```

9.
```sql
SELECT DISTINCT FirstName, LastName
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN PlayerPlaysHole
  NATURAL JOIN Hole
WHERE Rank = 1
  AND NumberOfStrokes - Par > 2;
```

10.
```sql
SELECT CourseName, HoleNumber
FROM Hole NATURAL JOIN Course
WHERE CourseSuburb IN ('Kew East', 'Balwyn North', 'Ivanhoe')
  AND Par = 3
ORDER BY Distance DESC
LIMIT 1;
```

Solutions – DML part 2

11. *"But the question doesn't ask for the suburb name," you might ask. True. But without the suburb name, you just get a meaningless list of numbers.*

```
SELECT Suburb, COUNT(*) AS NumPlayers
FROM Player
GROUP BY Suburb
ORDER BY Suburb;
```

12.
```
SELECT PlayerID, AVG(TotalScore) AS AverageTotalScore,
  MIN(Rank) AS HighRank
FROM PlayerCompetes
GROUP BY PlayerID
ORDER BY AverageTotalScore;
```

13. *Because CourseID is the primary key of the Course table, you can group by CourseID alone, and all other columns of the Course table will be available in the SELECT clause of the query.*

```
SELECT CourseName, CourseAddress, CourseSuburb, COUNT(*) AS NumHoles,
  SUM(Distance) AS TotalDistance
FROM Course NATURAL JOIN Hole
GROUP BY CourseID;
```

14.
```
SELECT FirstName, LastName, COUNT(HoleNumber) AS NumHoles,
  SUM(NumberOfStrokes) AS TotalStrokes
FROM Player
  LEFT JOIN PlayerPlaysHole
  ON Player.PlayerID = PlayerPlaysHole.PlayerID
GROUP BY Player.PlayerID;
```

Because of the non-natural join, there are two PlayerID columns. We need to say which one of the two columns we want to group by (even though they both contain the same data).

15. *We want to filter out the over-16 tournaments before any grouping takes place. For this, we use a WHERE clause.*

```
SELECT FirstName, LastName, COUNT(*) AS NumTournaments
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN Tournament
WHERE AgeLimit = 16
GROUP BY PlayerID
ORDER BY NumTournaments DESC;
```

16. *This time, we want to filter out players according to how many tournaments they played in. We only know this information after the grouping is done, so we need to use a HAVING clause.*

```
SELECT FirstName, LastName, COUNT(*) AS NumTournaments
FROM Player NATURAL JOIN PlayerCompetes
GROUP BY PlayerID
HAVING NumTournaments >= 5
ORDER BY NumTournaments DESC;
```

17. *This query brings together the ideas from queries 15 and 16, showing how WHERE and HAVING can be used together.*

```
SELECT FirstName, LastName
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN Tournament
WHERE AgeLimit = 16
GROUP BY PlayerID
HAVING COUNT(*) = 3;
```

18.
```
SELECT FirstName, LastName
FROM Player NATURAL JOIN PlayerPlaysHole NATURAL JOIN Hole
WHERE NumberOfStrokes < Par
  AND HoleNumber = 18
GROUP BY PlayerID
HAVING COUNT(*) >= 2
ORDER BY DateOfBirth DESC;
```

19. *In this query, there's nothing to group – we simply aggregate the entire table into a single row.*
```
SELECT MIN(StartDate), MAX(EndDate)
FROM Tournament
WHERE StartDate >= '2024-01-01'
  AND EndDate <= '2024-12-31';
```
To achieve this output as two rows in one column, we need to use UNION:
```
SELECT MIN(StartDate) AS Date
FROM Tournament
WHERE StartDate >= '2024-01-01'
UNION
SELECT MAX(EndDate) AS Date
FROM Tournament
WHERE EndDate <= '2024-12-31';
```
Both these queries assume that at least one tournament was held in 2024.

20.
```
SELECT PlayerID, FirstName, LastName, TournamentName,
  TotalScore AS IncorrectTotalScore,
  SUM(NumberOfStrokes - Par) AS CorrectTotalScore
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN PlayerPlaysHole
  NATURAL JOIN Hole NATURAL JOIN Tournament
GROUP BY PlayerID, TournamentID
HAVING IncorrectTotalScore <> CorrectTotalScore
ORDER BY ABS(IncorrectTotalScore - CorrectTotalScore) DESC;
```
The SELECT clause includes columns which depend on PlayerID (FirstName, LastName, and PlayerID itself), a column which depends on TournamentID (TournamentName), and a column which depends on both (TotalScore). That is why we group by these two columns.
And why ABS(…) in the ORDER BY clause? Because we're asked to put the worst discrepancies first. Maybe IncorrectTotalScore is too large, maybe it's too small – we don't know. The magnitude of the discrepancy is what we need to sort by.

Solutions – DML part 3

21.
```
SELECT TournamentName
FROM Tournament
WHERE CourseID IN (SELECT CourseID
                   FROM Course
                   WHERE CourseSuburb = 'Bundoora');
```

22.
```
SELECT FirstName, LastName
FROM Player
WHERE PlayerID NOT IN (SELECT PlayerID
                       FROM PlayerPlaysHole NATURAL JOIN Course
                       WHERE CourseName = 'Kooringal Golf Club'
                         AND HoleNumber = 18);
```

23.
```
SELECT FirstName, LastName, Address
FROM Player
WHERE Suburb = (SELECT Suburb
                FROM Player
                WHERE FirstName = 'Bella'
                  AND LastName = 'Sandbury');
```

24.
```
SELECT FirstName, LastName
FROM Player NATURAL JOIN PlayerPlaysHole NATURAL JOIN Course
GROUP BY PlayerID
HAVING COUNT(DISTINCT CourseID) = (SELECT COUNT(*)
                                  FROM Course);
```
*The DISTINCT keyword inside the COUNT function is essential here. The query needs to return players for whom the number of **different** courses they have played on is equal to the number of courses in the database. We do not want to count the same course more than once. Think about what would be counted if DISTINCT had not been included.*

25.
```
SELECT FirstName, LastName, TournamentName, Rank
FROM Player NATURAL JOIN PlayerCompetes AS PC1 NATURAL JOIN Tournament
WHERE Rank = (SELECT MIN(PC2.Rank)
              FROM PlayerCompetes AS PC2
              WHERE PC1.PlayerID = PC2.PlayerID);
```

26.
```
SELECT CourseName
FROM Course NATURAL JOIN PlayerPlaysHole NATURAL JOIN Player
WHERE FirstName = 'Bella'
  AND LastName = 'Sandbury'
GROUP BY CourseID
HAVING COUNT(DISTINCT HoleNumber) = (SELECT COUNT(*)
                                     FROM Hole
                                     WHERE Hole.CourseID =
                                           Course.CourseID);
```
An alternative approach could be to use a derived table. We can't simply group by CourseID in the outer query, because derived tables don't have primary keys:
```
SELECT CourseName
FROM (SELECT CourseID, CourseName, COUNT(*) AS NumHoles
      FROM Course NATURAL JOIN Hole
      GROUP BY CourseID) AS a
  NATURAL JOIN PlayerPlaysHole NATURAL JOIN Player
WHERE FirstName = 'Bella'
  AND LastName = 'Sandbury'
GROUP BY CourseName, NumHoles
HAVING COUNT(DISTINCT HoleNumber) = NumHoles;
```

27.
```
SELECT CourseName
FROM Course
WHERE CourseID IN (SELECT CourseID
                   FROM Tournament
                   WHERE StartDate >= '2024-01-01'
                     AND EndDate <= '2024-12-31')
  AND CourseID NOT IN (SELECT CourseID
                       FROM Tournament
                       WHERE StartDate >= '2025-01-01'
                         AND EndDate <= '2025-12-31');
```

28. *WHERE Address IN (…) AND Suburb IN (…) won't cut it. We need to make sure both the address and the suburb simultaneously match.*

```
SELECT FirstName, LastName
FROM Player AS Outer
WHERE (Address, Suburb) IN (SELECT Address, Suburb
                            FROM Player AS Inner
                            WHERE Outer.PlayerID <> Inner.PlayerID);
```

Alternatively, we can use a COUNT-based approach:

```
SELECT FirstName, LastName
FROM Player AS Outer
WHERE (SELECT COUNT(*)
       FROM Player AS Inner
       WHERE Outer.Address = Inner.Address
         AND Outer.Suburb = Inner.Suburb) > 1;
```

29.
```
SELECT FirstName, LastName, TournamentName, TotalScore
FROM Player NATURAL JOIN PlayerCompetes NATURAL JOIN Tournament
WHERE Rank <> 1
  AND PlayerID NOT IN (SELECT P2.PlayerID
                       FROM Player NATURAL JOIN PlayerPlaysHole
                         NATURAL JOIN Hole
                       WHERE NumberOfStrokes >= Par
                         AND PlayerPlaysHole.TournamentID =
                             Tournament.TournamentID);
```

30.
```
SELECT FirstName, LastName
FROM Player NATURAL JOIN PlayerCompetes
WHERE TournamentID IN (SELECT TournamentID
                       FROM Player NATURAL JOIN PlayerCompetes
                       WHERE FirstName = 'Bella'
                         AND LastName = 'Sandbury')
GROUP BY PlayerID
HAVING COUNT(*) >= 2;
```

Solutions – DML part 4

31.
```
UPDATE Hole
SET Par = 4
WHERE CourseID = (SELECT CourseID
                  FROM Course
                  WHERE CourseName = 'Sunshine Golf Club')
  AND HoleNumber = 10;
```

32.
```
INSERT INTO Player (PlayerID, FirstName, LastName, Address,
  Suburb, DateOfBirth)
VALUES (394, 'Olivia', 'Fairbanks', '4 Mayfair Avenue',
  'Huntingdale', '2007-01-05');
```

33.
```
DELETE FROM Course
WHERE CourseName = 'Sunshine Golf Course';
```

34.
```
UPDATE PlayerPlaysHole
SET NumberOfStrokes = NumberOfStrokes - 1
WHERE PlayerID = (SELECT PlayerID
                  FROM Player
                  WHERE FirstName = 'Bella'
                    AND LastName = 'Sandbury')
  AND TournamentID = (SELECT TournamentID
                      FROM Tournament
                      WHERE TournamentName = 'Medway Junior Open');
```

35.
```
DELETE FROM Course
WHERE CourseID NOT IN (SELECT CourseID
                       FROM Tournament)
  AND CourseID NOT IN (SELECT CourseID
                       FROM Hole
                       GROUP BY CourseID
                       HAVING COUNT(*) >= 9);
```

Why did we choose to ask for courses that are NOT IN the list of courses with 9 or more holes? Because we need to make sure courses with no holes get deleted. These courses don't appear in the Hole table at all.

Solutions – DML part 5

36.
```
SELECT TournamentName, COUNT(*) AS NumIneligible
FROM Tournament NATURAL JOIN PlayerCompetes NATURAL JOIN Player
WHERE StartDate >= DateOfBirth + (INTERVAL AgeLimit YEAR)
GROUP BY TournamentName
ORDER BY NumIneligible DESC;
```

37. *We need to use a unary join or self-join:*
```
SELECT Other.FirstName, Other.LastName, Other.Address
FROM Player AS Bella
  INNER JOIN Player AS Other ON Bella.Suburb = Other.Suburb
WHERE Bella.FirstName = 'Bella'
  AND Bella.LastName = 'Sandbury';
```

38.
```
SELECT DISTINCT FirstName, LastName
FROM Player NATURAL JOIN PlayerCompetes AS PC1
WHERE PC1.TournamentID IN (SELECT PC2.TournamentID
                           FROM Player NATURAL JOIN PlayerCompetes AS PC2
                           WHERE FirstName = 'Bella'
                             AND LastName = 'Sandbury')
  AND PlayerID NOT IN (SELECT PlayerID
                       FROM PlayerCompetes AS PC3
                       WHERE PC1.Rank > (SELECT PC4.Rank
                            FROM Player NATURAL JOIN PlayerCompetes AS PC4
                            WHERE FirstName = 'Bella'
                              AND LastName = 'Sandbury'
                              AND PC4.TournamentID = PC3.TournamentID));
```

39.
```
SELECT PC1.PlayerID, FirstName, LastName, TournamentName,
  PC1.Rank AS IncorrectRank, COUNT(PC2.PlayerID) + 1 AS CorrectRank
FROM Player NATURAL JOIN PlayerCompetes AS PC1 NATURAL JOIN Tournament
  LEFT JOIN PlayerCompetes AS PC2
  ON PC1.TournamentID = PC2.TournamentID
    AND PC2.TotalScore < PC1.TotalScore
GROUP BY PC1.PlayerID, PC1.TournamentID
HAVING IncorrectRank <> CorrectRank;
```
An outer join is used to make sure the first-placed player is not left out.
Think about what would happen if we had written `COUNT(PC2.PlayerID)` *instead of* `COUNT(PC2.PlayerID) + 1` *and* `PC2.TotalScore <= PC1.TotalScore` *instead of* `PC2.TotalScore < PC1.TotalScore`*. If two players had equal scores, what rank would be shown?*

40. *This type of query is surprisingly difficult to solve. Unfortunately, the use of a cross product is unavoidable (unless certain MySQL-specific SQL techniques are used). Here's one approach:*
```
SELECT DISTINCT P1.FirstName, P1.LastName
FROM Player AS P1, Player AS P2
WHERE P1.PlayerID <> P2.PlayerID
  AND 1 NOT IN (SELECT COUNT(*)
                FROM (SELECT TournamentID
                      FROM PlayerCompetes
                      WHERE PlayerID = P1.PlayerID
                      UNION ALL
                      SELECT TournamentID
                      FROM PlayerCompetes
                      WHERE PlayerID = P2.PlayerID) AS a
                GROUP BY TournamentID);
```
Alternatively, the "relational divide" technique can be adapted:
```
SELECT DISTINCT P1.FirstName, P1.LastName
FROM Player AS P1, Player AS P2
WHERE NOT EXISTS (SELECT *
                  FROM PlayerCompetes AS PC1
                  WHERE PlayerID = P1.PlayerID
                    AND NOT EXISTS (SELECT *
                                    FROM PlayerCompetes
                                    WHERE PlayerID = P2.PlayerID
                                      AND TournamentID = PC1.TournamentID));
```

41.
```
SELECT FirstName, LastName, PPH1.HoleNumber,
  SUM(PPH2.NumberOfShots - Par) AS CumulScore
FROM Player NATURAL JOIN PlayerPlaysHole AS PPH1
  INNER JOIN PlayerPlaysHole AS PPH2
  ON PPH1.PlayerID = PPH2.PlayerID
    AND PPH1.TournamentID = PPH2.TournamentID
    AND PPH1.HoleNumber >= PPH2.HoleNumber
  INNER JOIN Hole
  ON PPH2.CourseID = Hole.CourseID
    AND PPH2.HoleNumber = Hole.HoleNumber
WHERE PPH1.TournamentID = (SELECT TournamentID
                           FROM Tournament NATURAL JOIN Course
                           WHERE CourseName = 'Royal Park Golf Course'
                           ORDER BY StartDate DESC
                           LIMIT 1)
GROUP BY Player.PlayerID, PPH1.HoleNumber
HAVING CumulScore = (SELECT SUM(PPH3.NumberOfShots - Par) AS TopScore
                     FROM PlayerPlaysHole AS PPH3 NATURAL JOIN Hole
                     WHERE PPH1.TournamentID = PPH3.TournamentID
                       AND PPH1.HoleNumber >= PPH3.HoleNumber
                     GROUP BY PPH3.PlayerID
                     ORDER BY TopScore
                     LIMIT 1)
ORDER BY PPH1.HoleNumber;
```