SQL Questions and Answers

Database Systems, CSCI 4380-01
Sibel Adalı

October 3, 2002

Key to terms used to mark the difficulty of queries:

- **Basic**, single relation select-from-where query
- **Group By**, a simple select-from-where query with group by
- **Join**, requires a single select-from-where with multiple relations joined, including outer join
- **Scalar**, requires at least one scalar subquery
- **Nesting**, requires at least one nested subexpression, possibly involving NOT IN, NOT EXISTS
- **Complex**, requires a combination of multiple nesting and scalar queries
- **Update**, requires one of INSERT/UPDATE/DELETE
- **Advanced**, requires advanced features, possibly object-relational queries

Question 1 For the following questions, use the following schema.

Employee(SSN, Name, Salary, Dept_No, MGRSSN)
Customer(AccountNo, Name, LenderSSN)
Loan(LoanNumber, Amount, AccountNo)

Foreign Keys:

MGRSSN references Employee.SSN (Manager SSN)
Customer.LenderSSN references Employee.SSN
Loan.AccountNo references Customer.AccountNo

Write the following queries in SQL (you may not use Union, Intersect or Except):

1. **(Scalar)** Find the numbers (Dept_No) of all departments where the average salary of an employee in that department is strictly greater than the average salary of employees in the company.
2. **(Nesting)** Find the SSN of all employees whose managers do not have any customers (the relational algebra expression for this query is given below):

   $\Pi_{SSN} Employee - \Pi_{SSN}(Employee \bowtie_{MGRSSN=LenderSSN} Customer)$.

Question 2 Suppose you are given the following relational schema for storing information about students:

Student(SID , Student_name, Curriculum, EntryYear)
Course(CID , Course_name)
Faculty (FID, Faculty_name, Curriculum)
Take (SID, CID, FID, Semester, Year, Grade) /* semester is one of "Fall" or "Spring" */
Offer (FID,Semester,Year, CID)
Requirement (ReqID, CID, Curriculum)

Write down the following queries in SQL.

1. **(Nesting)** Find the names of all required courses for the "CS" curriculum that student "Smith" did not take yet.
2. **(Nesting)** Find the names of all students who never took a course offered by faculty named "Smith".
3. **(Group By)** Find the average grade of all "CS" curriculum students with respect to different semesters.
4. **(Nesting)** Find all courses that are offered at least once every year.
5. **(Nesting)** Find all faculty who taught a course in which the average grade for CS students was lower than the other students.

Question 3 School Database:

STUDENT(sid, lastname, firstname, major, entrydate, gpa, total_credits, probation_date)
COURSE(course_number, dept_id, course_name)
DEPARTMENT(dept_id, department_name, address)
INSTRUCTOR(instructor_id, firstname,lastname, dept_id) SECTION (section_id, section_number, course_number, dept_id, semester, instructor_id)
TRANSCRIPT(sid, section_id, credits, grade)
REQUIREMENT(req_id, major, course_number, dept_id)

PART A. Write following queries in SQL without using nesting (single select from where)
1. **(Basic)** Find the id and gpa of all students named "Kenny" (firstname).
2. **(Join)** Find the id of all courses offered by department "Computer Science".
3. **(Join)** Find the id of all sections of courses offered by department "Computer Science" department in the "Fall99" semester.
4. **(Join)** Find the id of all sections that a student named "Kenny" is taking in "Fall99" semester.
5. **(Join)** Find the last name and first name of all students who got an "A" at least once.

PART B. Write the following queries in SQL using a single nesting involving NOT EXISTS (i.e. exactly two select from where clauses nested). There is no simpler way to write these queries. Make sure you understand why.

1. **(Nesting)** Find the identifier of all students who never got an "F".
2. **(Nesting)** Find the identifier of all instructors who never taught a course.
3. **(Nesting)** Find the identifier of all students who never took the course 101 offered by department 11.
4. **(Nesting)** Find the name of all students who never took the course 101 offered by department 11.
5. **(Basic)** Find the lowest grade that a student got for section number 1216.

PART C. Write the following queries using no nesting and group by:
1. **(Group By)** Find the number of students in each major , output major name and the number of students.
2. **(Basic)** Find the number of students in section 1216 (you do not need a group by for this query).
3. **(Scalar)** Find the average number of students in each different course (identified by course_number and dept_id), average over different semesters. Find for each course, number of students for different semesters and then take the average.

PART D. Write the following queries using simple "UPDATE", "INSERT" or "DELETE" statements.
1. **(Update)** Set the gpa of the student named "Kenny" to 1.0.
2. **(Update)** Set the grade of student named "Kenny" for course number 111, dept_id 15 offered in "Fall99" to "F".
3. **(Update)** Insert a tuple into transcript indicating that "Kenny" is taking course number 111, dept_id 15 in "Fall99".
4. **(Update)** Delete all tuples from transcript about students named "Kenny" (firstname).

Question 4 Using the school database from the previous question, write the following queries in SQL. You may use any construct that was covered in class. (You may not use INTERSECT and EXCEPT).

1. **(Nesting)** Find the number and department identifier of all courses in which no student ever got an 'F'.
2. **(Complex)** Find the first name, last name and the department name for all instructors who are teaching at least three sections of a single course in semester 'Fall 99'.
3. **(Nesting)** Find the first name and last name of students who took all the required courses for their majors -even if they do not have a grade for some of the courses yet.
4. **(Group By)** For all different courses in the curriculum, find the total number of students in that class for each different semester (regardless of their sections).
5. **(Join)** Find the first name and last name of all instructors who taught a course offered by a department other than their own department.

Question 5 Any SQL expression containing the keyword 'ALL' for comparing a column with a sub-select expression can be converted to an equivalent expression that uses the keyword '[NOT] EXISTS' instead. In this problem, you are asked to write a conversion algorithm.
Suppose you are given a query of the form:

```
SELECT R1.a1, R1.a2, ..., R1.ak
FROM R1
WHERE R1.a1 ΘALL (SELECT R2.b2 FROM R2 WHERE Cond )
```

In this case, "Cond" is any condition involving the columns of relation R2, and "ΘALL" is one of ">ALL", "<ALL", "=ALL", "<>ALL", "<=ALL", or ">=ALL".
Write a parametric SQL query of the above form using "[NOT] EXISTS" instead of "ΘALL" that is equivalent to the above SQL expression. You may use "Θ" as a comparison predicate in your query.

Question 6 You are given the following database schema for an X-files database:

AGENTS(FBI-ID, StartDate, LastName, FirstName, Rank)
X-FILES(Fileno, CaseLocation, Description, Agent1, Agent2)
CONSPIRATORS(CodeName, Lastname, FirstName)
INVOLVED-IN(ConspiratorName, XfileNo)

Agent1 and Agent2 attributes in the X-FILES table are both foreign keys to the FBI-ID attribute of the AGENTS table. ConspiratorName attribute in the INVOLVED-IN table is a foreign key that refers to the CodeName attribute of the CONSPIRATORS table. XfileNo in the INVOLVED-IN table is a foreign key that refers to the Fileno attribute of the X-FILES table.

Given the following relational algebra query:
$T1 := \Pi_{XfileNo} INVOLVED - IN$
$T2 := (\Pi_{Fileno} X - FILES)[XfileNo] - T1$
$T3 := T2 \bowtie_{XfileNo=FileNo} X - FILES$
$T4 := \Pi_{Agent1} T3 \cup \Pi_{Agent2} T3$

a. Write down the meaning of the above query in English. Be concise and describe the query without referring to the particular operators. Describe what relations T1,T2, T3 and T4 correspond to separately.
b. **(Nesting)** Write your interpretation of the above query in SQL. You will get partial credit for this part even if your interpretation in part a is not correct as long as it is not overly simplistic. You may not use set difference in your SQL query.
c. **(Group By)** For all X-File cases that agent "Spender" worked on, return the Fileno and the total number of conspirators involved in that specific case.
d. **(Join)** Find the first name and the last name of all conspirators who have been involved in an X-File case located in "Baltimore".
e. **(Nesting)** Find the last and first name of all FBI agents who never worked on an X-File case.
f. **(Join)** Find all agents who worked on a case located in "North Pole".
g. **(Group By)** Find the number of conspirators involved in each X-files case. For each case, return the description and the total number of conspirators for that case.

Question 7 You are given the following set of relations for an Olympic database for events involving competition among individual athletes. All events in this database measure success by the shortest time (such as running).

ATHLETE(athlete_id, name, county, birthdate)
SPORT(sport_id, name)

PARTICIPATE(athlete_id, sport_id, personal_record)
RECORD(type, sport_id, athlete_id, record_time, date)
MEDAL(medal_id, event, year, sport_id, athlete_id, medal_type, time)

NOTES:

- athlete_id in all relations other than Athlete is a foreign key to Athlete.athlete_id. Similarly, sport_id in all relations other than Sport is a foreign key to Sport.sport_id.
- Participate.personal_record is the best time for an athlete for a given sport.
- Record.type is one of WR (world record) or OR (Olympic record).
- Medal.event is an event like 'Olympics', 'World Championship', etc.
- Medal.type is one of 'gold', 'silver', or 'bronze'.

1. **(Group By)** Write the following query in SQL:

   Find all athletes who have won at least three gold medals in the same event and the same year. Return the id and the name of the athlete, the name of the event, the year and the total number of gold medals the athlete has won at that event.

2. **(Complex)** Write the following query in SQL:

   Find all athletes who have never won a medal in a specific sport but have a personal_record that is better than the current world record for that sport. Return the name of the athlete, the personal record for the athlete, the name of the sport, and the total number of athletes that are faster than the current athlete for that sport. (Hint. To find the number of athletes that are faster, you can use a scalar subquery that finds all athletes with personal records better than the current athlete for that sport.)

3. Write the meaning of the following query in English. Your description should be in English as in question 1 and 2. It should not involve terms like SELECT, FROM, JOIN, etc. Explain your answer in detail by first describing the result of the WHERE clause in English, and then the result of the GROUP BY-HAVING clause in English.

```
SELECT A.name
FROM   Athlete A, Medal M1, Medal M2
WHERE  M1.medal_type = 'gold' AND M2.medal_type = 'gold' AND
       M1.year = M2.year + 1 AND M1.event = M2.event AND
       M1.athlete_id = M2.athlete_id AND M1.athlete_id = A.athlete_id
GROUP BY A.name
HAVING count(DISTINCT M1.event) > 1
```

4. **(Advanced)** Suppose in the above database, we replace the medal table with the following.

```
CREATE TYPE medal_t AS OBJECT
(
medal_type VARCHAR(6),
athlete_id   INT
```

```
)
/
CREATE TYPE allmedal_t AS TABLE OF medal_t
/
CREATE TABLE medal
(
medal_id INT PRIMARY KEY,
event VARCHAR(10),
year INT,
medallist allmedal_t,
sport_id INT FOREIGN KEY REFERENCES sport(sport_id)
) NESTED TABLE medallist STORE AS medallist_tab ;
```

Write the following query in SQL.

Find all athletes who have won at least three gold medals in some sport and also hold the world record for that sport. Return the name of the athlete and the name of the sport.

5. **(Complex)** Find all athletes that participate in more than one sport and has won a medal in each sport they participate and hold a (world or Olympic) record in at least one of them. Return the name of the athlete, the number of sports they participate in and the total number of gold medals they have won so far (all the sports combined) [Hint. You can use a scalar subquery to return the total number of medals].

6. Write the meaning of the following query in English. Your description should be in English and should not involve terms like SELECT, FROM, JOIN, etc. Explain your answer in detail by first describing the result of the WHERE clause in English, and then the result of the GROUP BY-HAVING clause in English.

```
SELECT A.athlete_id, A.name
FROM   Record R1, Record R2, Athlete A, Participate P
WHERE  R1.athlete_id = R2.athlete_id AND R1.sport_id = R2.sport_id AND
       A.athlete_id = R1.athlete_id AND R1.type = 'WR' AND R2.type = 'OR' AND
       A.personal_record < R1.record_time AND P.athlete_id = A.athlete_id
GROUP BY A.athlete_id, A.name
HAVING count(DISTINCT P.sport_id) > 3
```

Question 8 SHARPEN YOUR SQL SKILLS EXERCISE

Student(ssn, name, major, year) // lists all students
Course (cid, name, description, department, credits) // lists course information
Offered(cid, semester, year, section, ins_id, location) // lists all course offerings for different semesters
Instructor(ins_id, name, department)// lists all instructors
Took(ssn, cid, semester, year, section, grade) // lists when a student took a course
Required(major, cid)// lists which courses are required for a specific major

Took.cid, Required.cid Offered.cid are foreign keys, they refer to the course.cid.

Offered.ins_id is a foreign key, refers to Instructor.ins_id.

Took.ssn is a foreign key, refers to Student.ssn.

In Offered and Took, semester is one of fall or spring, year is given as a number.

Problem 1. First, you will write a query to return the name of all "generous" professors in the database. The aim is to find professor who have given "A" to more than %10 of the class in the past, together with the class, semester and year in which this happened.

We will construct this query in a number of steps.

Step 1. For all instructors, find the courses they have offered and the grades they have given. Return, instructor name, course id, semester, year and the grade in the output.

Hint: Use a single SFW statement, join Instructor, Offered and Took. No nested statements are needed.

Step 2. Once you are done with the query in Step 1, try to find the total number of students in each class. Return instructor id, course id, semester, year and the total number of students. For this exercise, disregards different sections.

Hint: Add to your previous query a simple group by for instructor id, course id, semester, and year.

Step 3. Now instead of returning the total number of students, return the total number of "A"s in each class. Return the same information as in Step 2.

Hint: Add a single condition to the "WHERE" clause in the query from Step 2.

Step 4. You will eliminate from the above query some instructor + class + semester + year combinations if the total number of "A"s divided by 100 is less than 0.10. We will replace the number 100 by the total number of students in the next step.

Hint: Add a having clause to your query and eliminate the undesired groups according to the above criteria.
Step 5. Finally, we will replace 100 with the total number of students for a given class. Make sure that you understand the following: for each group that you are considering in the having clause, the combination of "ins_id, cid, semester, year" attributes is unique.

On the side, write a query that returns a single scalar value corresponding to the total number of students for a given "ins_id, cid, semester, year". You will need to select count(*) from took, all students for this instructor and course. Make sure you are referring to the same attributes and aliases that you use in the group by.

Now, substitute this query for the number "100".

And, voila! You are returning all professors and courses, where the professor has given "A"s to more than %10 of the class.

Problem 2. Now, we want to use the query from Problem 1 to find "sparing professors", those who have never given "A"s to more than %10 of their students in any class.

Step 1. Write a simple SFW statements that returns all instructor names from the instructor relation.

Step 2. You will want to exclude from these all professors who have been generous at least once. Hence, for all instructors in step 1, check if they have ever been generous. If so, exclude their name from the result set.

Hint: This is a case where you need NOT IN, or NOT EXISTS. You cannot use NOT IN since in the above query, you have to return four attributes because of the group by operation. Make sure, you understand why.

Instead, for each instructor, check that there does not exist a tuple in the generous instructor relation. A tuple in the generous instructor means that there was a course, semester and year, in which this instructor was generous. All you have to do is to make sure you construct the generous instructor relation only for the current instructor from the outer relation.

Problem 3. We want to extend query from problem 2 to students. We want to return students who have received an "A" from a "sparing instructor".

Hint: All you need is to add two more relations to your (outer) query and add a join. No need to add additional nesting. Add students and took, and join to find for each student a course they took from that instructor and got an "A".

Problem 4. Finally, you want to find students who managed to get an "A" from all "sparing instructor" they have encountered. Return their names.

Hint: You should use a query similar to problem 3 as the inner query. Now, for each student, check that for that student there does not exist a course they took from a sparing professor and got a grade other than "A".

Problem 5. Now, let's do a for-all query just to prove that you have learnt SQL. Find all department that have offered a course in all years. We will try to solve this in two steps.

Step 1. First for each department find all years that no course was offered by that department. For this, you will need to use course and offered. Return pairs of semester from offered and department names from course such that there does not exist a tuple in offered where a course by this department was offered in that year.

Hint: Use course and offered in the outer relation, and course and offered in the nested not exists query. You are using the relations in the outer relation only to find the names of all departments and all years. Let's call this query (Y not exists Z)

Step 2. Now, write a new outer query that looks at the course department to find the names of all department and check for each department that there does not exist a year that the department

did not offer a course.

Hint: Let's call this query (X (not exists Y (not exists Z). The outer relation calls a nested query which is the one you constructed in step 1. Instead of finding all department and year pairs in the Y part of the query, send the name of the department from X to Y and check if there exists a year.

Question 9 You are given the below database for maintaining student records.

Student(ssn, name, major, year) // lists all students
Course (cid, name, description, department, credits) // lists course information
Offered(cid, semester, year, section, ins_id, location) // lists all course offerings for different semesters
Instructor(ins_id, name, department)// lists all instructors
Took(ssn, cid, semester, year, section, grade) // lists when a student took a course
Required(major, cid)// lists which courses are required for a specific major

Took.cid, Required.cid Offered.cid are foreign keys, they refer to the course.cid.

Offered.ins_id is a foreign key, refers to Instructor.ins_id.

Took.ssn is a foreign key, refers to Student.ssn.

In Offered and Took, semester is one of fall or spring, year is given as a number.

Problem 1. Write the following queries in SQL.

1. **(Nesting)** Find all instructors who never taught a course that was offered by some department other than their own (the department of an instructor is given by Instructor.department, the department of a course is given by Course.department).
2. **(Nesting)** Find students who have taken all the required courses for their major.
3. **(Nesting)** Find the name of all students whose gpa for a semester has never fallen below 2.5. Note that you need to compute the average grade taking into account the number of credits (course.credits) for each course.
4. **(Nesting)** Find all courses that are offered exactly once a year, either in spring or in fall semesters, regularly. Regularly means that no year was missed since the beginning of the database.
5. **(Complex)** For each instructor, return the name of the instructor, and for each semester, the total number of credit hours she has taught, and the total number of students in all her classes for that semester.

Problem 2. Suppose you created a table StudentStatus containing for each student the total number of credits they have taken so far and their current gpa.

StudentStatus(ssn, totalcredits, gpa)

Write the following queries in SQL:

1. **(Update)** Write a delete statement to delete all students who have never taken a course in both

Fall 2000 and Spring 2000 semesters.
2. **(Update)** Write an insert statement to insert all students who have enrolled in 2000 into this table. Note Student.year is an integer gives the graduating year for a student.
3. **(Update)** Write an update statement to update the gpa of a student. Assume the totalcredits field is correct. Find each student's grade from the Took relation, and multiply the grade with the number of credits for the course. If the student has taken the course more than once, the latest grade is the one counted in the GPA. (Hint. If you cannot solve this problem in one step, you can use two subsequent update statements.)

Note for Problems 1.3 and 2.3. Assume the existence of a function grade_value which returns the numerical value of a letter grade. In this case, both an "F" and an "I" (incomplete) returns 0. A null value for the current semester also returns 0, but this should not be counted in the GPA.

You can use this function as any other function:

```
SELECT * FROM Took WHERE grade_value(Took.grade) >= 3
```

Music Database Questions

You are given the below music database (loosely based on allmusic.com). Below is the complete list of relations. The meaning of each relation is explained in detail in the next section. The attributes making up the primary keys are underlined.

- Label(Name, Decades)
- Genre(Name, Summary)
- Style(Name, Summary, Genre, MainStyle)
- Artist(Name, YearFounded, Origin, Tones, Biography)
- Song(Id, Name, DateComposed)
- Album(Id, Title, Duration, Rating, Review, ReleaseDate, Tones, Type, Artist, Year, LabelName)
- Track(Album, TrackId, Recommended, Song)
- RepresentativeAlbum(Album, Style, Decade)
- SimilarArtist(ToA, ToY, FromA, FromY)
- MajorContributor(McA, McY, McOfA, McOfY)
- MemberOf(MA, MY, GA, GY)
- Influence(Artist, Year, InfluencedA, InfluencedY)
- AlbumStyle(Album, Style)
- RelatedStyle(FromStyle, ToStyle)

- MajorArtist(Artist, Year, Label)
- RecordsGenre(Label, Genre)
- AppearOn(Artist, Year, Album, TrackId)
- LyricsBy(Song, Artist, Year)
- MusicBy(Song, Artist, Year)

Foreign Keys: Notation: R(A,B) references S(C,D) means attributes A and B of relation R (together) is a foreign key referencing attributes C and D of S (respectively).

- Style(Genre) references Genre(Name)
- Album(Artist, Year) references Artist(Name, YearFounded)
- Album(LabelName) references Label(Name)
- Track(Album) references Album(Id)
- Track(Song) references Song(Id)
- RepresentativeAlbum(Album) references Album(Id)
- RepresentativeAlbum(Style) references Style(Name)
- SimilarArtist(ToA, ToY) references Artist(Name, YearFounded)
- SimilarArtist(FromA, FromY) references Artist(Name, YearFounded)
- MajorContributor(McA, McY) references Artist(Name, YearFounded)
- MajorContributor(McOfA, McOfY) references Artist(Name, YearFounded)
- MemberOf(MA, MY) references Artist(Name, YearFounded)
- MemberOf(GA, GY) references Artist(Name, YearFounded)
- Influence(A, Y) references Artist(Name, YearFounded)
- Influence(InfluencedA, InfluencedY) references Artist(Name, YearFounded)
- AlbumStyle(Album) references Album(Id)
- AlbumStyle(Style) references Style(Name)
- RelatedStyle(FromStyle) references Style(Name)
- RelatedStyle(ToStyle) references Style(Name)
- MajorArtist(Artist, Year) references Artist(Name, YearFounded)
- MajorArtist(Label) references Label(Name)
- RecordsGenre(Label) references Label(Name)

- RecordsGenre(Genre) references Genre(Name)
- AppearOn(Artist, Year) references Artist(Artist, YearFounded)
- AppearOn(Album, TrackId) references Track(Album, TrackId)
- LyricsBy(Song) references Song(Id)
- LyricsBy(Artist, Year) references Artist(Artist, YearFounded)
- MusicBy(Song) references Song(Id)
- MusicBy(Artist, Year) references Artist(Artist, YearFounded)

Details of the above relations:

- Label(Name, Decades) stores the name of recording label and the years it was active as a single string containing all the decades.
- Genre(Name, Summary) stores information about music genre (such as rock, jazz, pop, etc.)
- Style(Name, Summary, Genre, Main-style) stores information about various styles of a specific genre (such as folk-rock, heavy-metal, indie-rock, etc.). "Main-genre" is a Boolean field that refers to whether a specific style is considered a "main-style of its genre. Note that the E/R data model requires each genre to have 5 to 10 main styles. However, this constraint is not translated to the relational schema. This constraint has to be enforced in another way.
- Artist(Name, YearFounded, Origin, Tones, Biography) stores information about an artist or a music group. The list of group members can be found using the "MemberOf" relation. The attribute origin lists the country the artist is from and tones is a string containing all the tones that describe the music of this artist (such as Reflective, Intense, Plaintive, Wistful, Paranoid, Gloomy, etc.).
- Song(Id, Name, DateComposed) stores information about various songs.
- Album(Id, Title, Duration, Rating, Review, ReleaseDate, Tones, Type, Artist, Year, Label-Name) stores information about an album. If the album was released by a specific artist, then the "Artist, Year" fields point to the corresponding artist in the artist relation. If this is a compilation of tracks performed by various artists, then "Artist, Year" fields are left empty. The performer of each track is then stored in the "AppearOn" relation. If the album belongs to a single artist, then the "AppearOn" relation stores information about various guest artists for specific tracks.
- Track(Album, TrackId, Recommended, Song) information about specific tracks of an album. The Boolean field "Recommended" determines whether this is a recommended track or not. Song is a foreign key referring to the song that was features on that track.
- RepresentativeAlbum(Album, Style, Decade) for each style and decade (1910s, 1920s,...), the relation lists a number of albums are representative of this style at that given decade.
- SimilarArtist(ToA, ToY, FromA, FromY) asserts that Artist(ToA,ToY) is similar to Artist(FromA, FromY). Note that the reverse may not necessarily be true. Recall: you need to attributes to refer to a specific artist in the artist relation due to its primary key.

- MajorContributor(McA, McY, McOfA, McOfY) asserts that Artist(McA, McY) was a major contributor of the music of Artist(McOfA, McOfY).
- MemberOf(MA, MY, GA, GY) asserts that Artist(MA, MY) is a member of music group given by Artist(GA, GY).
- Influence(Artist, Year, InfluencedA, InfluencedY) asserts that Artist(Artist, Year) is a major influence for Artist(InfluencedA, InfluencedY).
- AlbumStyle(Album, Style) asserts that an Album belongs to a specific style.
- RelatedStyle(FromStyle, ToStyle) asserts that "FromStyle" is related to "ToStyle". The reverse may not necessarily be true. MajorArtist(Artist, Year, Label) Asserts that Artist(Artist, Year) is a major artist of a specific recording label.
- RecordsGenre(Label, Genre) asserts that a label records music of a specific genre.
- AppearOn(Artist, Year, Album, TrackId) asserts that a specific Artist(Artist, Year) appear on a specific track of an album. This table is used to store information about the guest artists and compilation CDs (featuring a different artist on each track).
- LyricsBy(Song, Artist, Year) asserts that the lyrics to a specific song was written by an Artist(Artist, Year).
- MusicBy(Song, Artist, Year) asserts that the music for a specific song was composed by an Artist(Artist, Year). Note that by the data model, a song may be written/composed by multiple artists.

Question 10 Write the following queries using SQL. Try to write your queries as succinctly as possible. Try to use nested expressions only when they are necessary.

1. **(Basic)** Find all albums with 20 or more tracks, return the title and the duration of the album.
2. **(Basic)** Find all artists that are similar to the artist named 'Prince', return the name and year founded of these artists.
3. **(Nesting)** Find all artists who have never contributed to a song (either with lyrics or music). Return the name of the artist.
4. **(Basic)** Find all albums that feature a song by 'Prince'.
5. **(Nesting)** Find artists who are considered major artists for all labels that they recorded with. Return name, origin and biography of the artist (Hint. Use Album and MajorArtist relations first).

Question 11 Write the following queries in SQL:

1. **(Join)** Find for all artists, the name of the artist, the number of albums they have recorded and the number of songs that they have composed (musicBy).
2. **(Complex)** Find all artists who never recorded a song by another artist, return their name and year founded.
3. **(Group By)** The ratings for albums are on a 5-point basis, a rating of 4 or 5 is considered a "good" album. Find all artists who have received a "good" rating for at least eight of their albums.

4. **(Basic)** Find all compilation albums, return all the information about the album. A compilation album will have associated artists that appear on every single track through the "AppearOn" relation.

Question 12 (Complex) Based on the music database given above, you are given the following relational algebra query. Write down the meaning of the query in English and write an SQL query that computes the same answer as this query. Show and explain your work for any partial credit.
$R := \Pi_{Song,Artist,Year}(Album \bowtie_{id=album} Track)$

$S := R - (LyricsBy \cup MusicBy)$

$Answer := (\Pi_{Name,YearFounded} Artist)[Artist, Year] - \Pi_{Artist,Year} S.$

Question 13 You are given the following SQL query. Write down the meaning of the query in English and write an equivalent query in relational algebra. Show and explain your work for any partial credit.

```
SELECT A.ArtistName, A.ArtistYear
FROM Track T, Album A, Lyricsby L
WHERE A.Id = T.Album AND T.Recommended='y' AND
      L.Artist=A.Name AND L.Year=A.YearFounded AND
      L.Song=T.Song
GROUP BY  A.Name, A.YearFounded
HAVING count(A.Id) < 3
```

Question 14 Write the following queries in SQL using only a single SQL statement (you may use union of course).

1. **(Group By)** For all major artists (MajorArtist relation), return the name, year of the artist, and label name, and the total number of albums they have recorded with this label with rating 4 or 5.
2. **(Nesting)** Find all artists who are only song writers, i.e. they have written lyrics (LyricsBy) or composed music (MusicBy) of songs, but they have never recorded any albums (Album). Return the name, year founded, and origin of the artist.
3. **(Complex)** For all albums, find the total number of songs by artist named 'Bob Dylan' (MusicBy or LyricsBy) with datecomposed after 1996. For each album, return the title of the album and the total number of songs.
4. **(Nesting)** For all artists, find all labels they are not a major artist for (you can use the "MajorArtist" relation to find all labels). Return the name, year of the artist and the name of the label.
5. **(Complex)** An album is considered an "all time favorite", if at least 4 or more tracks on this album are recommended (i.e. the "track.recommended" attribute has value 'y' for these tracks). Find all artists who have at least three "all time favorite" albums. Return the name, year of the artist and the total number of the all time favorite albums.

Question 15 You are given the following relational schema for family relations:

Person (PID, Person_Name, Age, Sex, Yearofbirth)
Parent (ParentID, ChildID)
/* both ParentID and ChildID are foreign keys referring to Person.PID */
Married (HusbandID, WifeID, YearofMarriage)

Write the following queries in SQL (you may use any keyword as long as it is SQL standard):

- **(Join)** Find the name of all children who were born after their father married their mother (i.e. their year of birth is bigger than the year of marriage of their parents).
- **(Group By)** Find the name of all husbands who were married only once according to the database.
- **(Group By)** Find the id of all parents such that the average age of their children is greater than 20.

Question 16 Use the two relations given below to write the following queries in SQL:

EMPLOYEE(ssn, firstname, lastname, title, salary, dept_id, mgr_id)
DEPARTMENT(dept_id, departmentname, totalbudget)

- **(Group By)** Find the name, total budget of departments in which the total salary of all people working in that department exceeds the total budget of the department.
- **(Join)** Find the first and last name of employees whose salary is greater than the salary of their manager.
- **(Basic)** Find the ssn, first and last name of employees who do not have a manager.
- **(Nesting)** Find name of departments that does not have any employees with title "Vice President" working in them.

Question 17 Use the three relations given below to write the following queries in SQL:

BOOKS(ISBN, Author, Title, Publisher, Publish_Date, Pages, Notes)
STORE(Store_Id, Store_Name, Street, State, City, Zip)
STOCK(ISBN, Store_Id, Price, Quantity)

- **(Complex)** Find the name and address of the bookstore(s) carrying the smallest total quantity of books by "J.D. Salinger."
- **(Basic)** Find the average price of books by "Morgan Kauffman" publishers in the bookstore with id 111.
- **(Nesting)** Find the Store_Id and Store_Name of all bookstores that do not carry any books by "Dr. Zeuss".

- **(Complex)** Find the Store_Id and Store_Name of all bookstores that are missing at least ten books from the BOOKS relation in their stock (for these books,either the Quantity=0 or there is no stock tuple).

Question 18 In this question, you are given the following simple database of employees that work in specific departments. Each department has an inventory of items with specific quantity.

EMPLOYEE(ssn, first-name, last-name, address, date-joined, supervisor-ssn)
DEPARTMENT(dept-no, name, manager-ssn)
WORKS-IN(employee-ssn, dept-no)
INVENTORY(dept-no, item-id, quantity)
ITEMS(item-id, item-name, type)

Foreign keys:
1. EMPLOYEE.supervisor-ssn and WORKS-IN.employee-ssn point to EMPLOYEE.ssn.

2. WORKS-IN.dept_no and INVENTORY.dept-no point to DEPARTMENT.dept_no.

3. INVENTORY.item-id points to ITEMS.item-id.

Write the following queries in SQL. Make sure you return what the query is telling you to return:

- **(Complex)** Find all departments that sell more than 5 items of type "Widget" and no items of type "Shoe", return the name of the department and the total number of employees that work in that department.
- **(Join)** Find for all employees, their grand-supervisor, i.e. the supervisor of their immediate supervisor. Return the name of the employee and the name of their grand-supervisor. If somebody does not have a grand-supervisor, you should return null next to their name.
- **(Complex)** Find all departments that have a larger inventory than the department "Joe Smith" is managing. The inventory is defined to be the sum of the quantity of items for this department. Return the name of the department and the total number of employees working in that department.
- **(Nesting)** Find all employees who work in all departments that have an item of type "Shoe" in their inventory. Return the name of employees.