University of California, Berkeley
College of Engineering
Computer Science Division – EECS

Fall 2004 Prof. Michael J. Franklin

MIDTERM II

CS 186 Introduction to Database Systems

NAME: D.B. Guru **STUDENT ID:**__________________

IMPORTANT: Circle the last two letters of your class account:

```
cs186 a b c d e f g h i j k l m n o p q r s t u v w x y z
      a b c d e f g h i j k l m n o p q r s t u v w x y z
```

DISCUSSION SECTION DAY & TIME:_____________ **TA NAME:** _____________

This is a **closed book** examination – but you are allowed one 8.5” x 11” sheet of notes (double sided). You should answer as many questions as possible. Partial credit will be given where appropriate. There are 100 points in all. You should read **all** of the questions before starting the exam, as some of the questions are substantially more time-consuming than others.

Write all of your answers directly on this paper. **Be sure to clearly indicate your final answer** for each question. Also, be sure to state any assumptions that you are making in your answers.

GOOD LUCK!!!

Problem	Possible	Score
1. SQL	27	
2. Join Cost Calculations	9	
3. Query Optimization	24	
4. ER Diagrams	25	
5. Functional Dependencies	15	
TOTAL	100	

SID:____________________

Question 1 –SQL [6 parts, 27 points total]

For parts a-d, consider the following schema (primary keys are underlined):

Student (sname, <u>sid</u>, gpa, level, deptno)
Course (<u>cno</u>, cname, deptno, units)
Dept (dname, <u>deptno</u>)
Takes (<u>sid</u>, <u>cno</u>)

a) [7 points] Write a SQL query that returns the names (i.e., snames) of students who have taken more courses outside their department than inside their department. For this question, you can assume that all students in the database have taken at least one course inside their department. **(note: you should do scratch work elsewhere and just put your final answer here!)**

```
SELECT S.sname from Student S
WHERE (SELECT COUNT (*)
            FROM Takes T, Course C
            WHERE S.sid = T.sid AND T.cno = C.cno AND C.deptno = S.deptno)
        < (SELECT COUNT(*)
            FROM Takes T2, Course C2
            WHERE S.sid = T2.sid AND T2.cno = C2.cno AND C2.deptno != S.deptno)
```

(Note: The above solution does not even require that each student takes atleast one course in her department)

Another variation seen in solutions:

```
SELECT S.sname
 FROM Student S, Takes T, Course C
WHERE S.sid = T.sid AND T.cno = C.cno AND S.deptno = C.deptno
GROUP BY S.sid, S.sname
HAVING count(*) <
            (SELECT COUNT (*)
                  FROM Takes T1, Course C1
                  WHERE S.sid = T1.sid AND T1.cno = C1.cno AND C1.deptno != S.deptno)
```

b) [3 points] Which of the following queries returns the department numbers of those departments for which there are no courses being offered? **More than one choice may be correct.**

A) SELECT D.deptno
   FROM Dept D, Course C
   WHERE D.deptno NOT EQUAL C.deptno;

B) SELECT C.deptno, COUNT(C.deptno)

```
    FROM Course C
    GROUP BY C.deptno
    HAVING COUNT (C.Deptno) = NULL;
```

C) SELECT C.deptno
FROM Course C
WHERE C.deptno NOT IN (SELECT * FROM Dept);

D) SELECT D.deptno
FROM Dept D
WHERE NOT EXISTS (SELECT * FROM Course C
WHERE C.deptno = D.deptno);

E) None of the above

c) [3 points] Which of the following queries returns the id of the student with the highest GPA? **More than one choice may be correct.**

A) SELECT S.sid
FROM Students S
WHERE S.gpa = MAX(S.gpa);

B) SELECT S.sid, MAX(S.gpa);
FROM Students S
GROUP by S.gpa

C) SELECT S.sid
FROM Student S
WHERE S.gpa > ALL (SELECT S.gpa FROM Student S);

D) SELECT S.sid
FROM Student S
Where S.gpa = (SELECT MAX(S.gpa)
FROM Student S);

E) None of the above

d) [3 points] Which of the following queries returns the sid of the students and the total units they are taking? **More than one choice may be correct**.

A) SELECT S.sid, sum(C.units)
FROM Student S, Takes T, Course C
GROUP BY S.sid
HAVING S.sid = T.sid AND T.cno = C.cno;

B) SELECT S.sid, sum(C.units)
FROM Student S, Takes T, Course C
Where S.sid = T.sid AND T.cno = C.cno;

GROUP BY S.sid

C) SELECT S.sid, Temp.Sum1
```
    FROM Student S, (SELECT sum( C.units) AS Sum1
                FROM Takes T, Course C
                WHERE T.sid = S.sid AND T.cno = C.cno) AS Temp;
```

D) SELECT S.sid, sum(C.units)
```
    FROM Student S, Takes T, Course C
    WHERE S.sid = T.sid AND T.cno = C.cno;
```

E) None of the above

e) [3 points] For the following schema: Athletes(<u>name</u>, country, sport, age, height, weight)

Which of the following SQL queries reflects the English query statement: "For each country, find the average height of weightlifters, qualifying only those countries that have weightlifters with minimum weight of 160 pounds." **More than one choice may be correct.**

```
A)  SELECT country, avg(height)
    FROM Atheletes
    WHERE sport = "weightlifting"
    GROUP BY country, height, weight  HAVING min(weight) >= 160;
```

B)
```
B)  SELECT country, avg(height)
    FROM Atheletes
    GROUP BY country, sport  HAVING min(weight) >= 160 AND sport =
    "weightlifting";
```

C)
```
C)  SELECT country, avg(height)
    FROM Atheletes
    WHERE sport ="weightlifting"
    GROUP BY country  HAVING min(weight) >= 160;
```

```
D)  SELECT country, avg(height)
    FROM Atheletes
    WHERE sport = "weightlifting" AND min(weight) >= 160
    GROUP BY country, weight;
```

```
E)  SELECT country, avg(height)
    FROM Atheletes
    WHERE sport ="weightlifting"
    GROUP BY country, height  HAVING min(weight) >= 160;
```

f) [8 points] For the schema in part (e), write a SQL query that returns for each sport, the name of the sport, the country that has the most athletes who play that sport, and the number of athletes of that country that play that sport. **(note: you should do scratch work elsewhere and just put your final answer here!)**

```
SELECT A.sport, A.country, count(*)
FROM Atheletes A
GROUP BY A.sport, A.country
HAVING count(*) >= ALL
                    (SELECT COUNT(*)
                    FROM Athletes A1
                    WHERE A1.sport = A.sport
                    GROUP BY A1.country)
```

Question 2 – Join Costs [3parts, 9 points total]

For this question, you will consider the I/O cost of operations on two tables of a database. The database has the following schema (note, this schema is slightly different than the schema used in question 1).

Students(sid, name, address, GPA)

EnrolledIn(sid, classid, semester, year)

Assume that tuples are of fixed size. There are 10 Students tuples per page and 200 EnrolledIn tuples per page. Also assume that there are 1000 pages in the Students relation, and 500 pages in the EnrolledIn relation. The data is unsorted, and tuples are distributed evenly throughout the database.

For the following join strategies, give the I/O cost. Assume 52 pages in the buffer, and that no pages are currently in the buffer when the join begins. If multiple variants of an algorithm have been discussed in class, section, or in the book, use the most efficient one unless otherwise noted. **Be sure to state any assumptions you are making and be sure to clearly indicate your final answer.**

a) [3 points] Hash Join (not hybrid):

*3(1000+500)=**4500***

b) [3 points] Sort Merge (note, both relations can be sorted in two passes):

sort(Enrolled)+sort(Student)+[Enrolled]+[Student]
= 4(500)+4(1000)+500+1000
*= **7500***
OR
= 2(500)+2(1000)+500+1000
*= **4500***

c) [3 points] Block Nested Loops:

Enrolled need to be the outer relation because it is smaller:
*[Enrolled] + [Enrolled/blocksize] * [Student] = 500 + (500/50)*1000*
*= **10500***

Question 3 – Query Optimization [7 parts, 24 points]

Consider a database containing information about all the car accidents between 1967 and 1975, including the cars involved and their owners. The database has the following tables:

```
Car(license, year, company, model);
Accident(license, accident_date, damage_amount, zipcode);
        // zipcode in Accident is the place where accident took place
    // assume that the same car does not get into an accident twice in a day
Owner(SSN, license, name, street, city, zipcode);
                        // assume each owner has only one licensed car
```

The statistics and other catalog information for the database are as follows:

- NTuples(Car) = 10,000, NTuples(Accident) = 10,000, NTuples(Owner) = 10,000
- NPages(Car) = 100, NPages(Accident) = 1000, NPages(Owner) = 500
- NDistinct(Car.company) = 50
- Min(Accident.damage_amount) = 1000, Max(Accident.damage_amount) = 16,000.
- Histogram on # of accidents per year (evenly distributed among the 12 months in each year:

Year	1967	1968	1969	1970	1971	1972	1973	1974	1975
N(Accidents)	500	1000	1500	1200	1500	1500	950	1250	500

- All indexes use Alternative #2 for the data entries.
- All BTrees have 100 keys per node, hash indexes have 100 keys per bucket,; assume BTrees are 3 levels deep, and the cost for a hash look up is 1.2 I/Os on average.
- An unclustered Extendible hash index exists on Car(company)
- A clustered B+Tree index on Accident(accident_date)
- An unclustered B+Tree index on Accident(damage_amount)

Consider the following query:

```
SELECT O.name, A.damage_amount
FROM Car C, Accident A, Owner O
WHERE C.license = A.license AND C.license = O.license
  AND A.zipcode = O.zipcode AND C.company = 'Volvo'
  AND A.accident_date < '07/01/1970' AND A.damage_amount > 10000;
```

For question parts **(a)** and **(b)**, begin the process of query optimization, by first pushing down all the non-join predicates. Compute the cardinality of the relations after these selections are applied.

a) [2 points] What is the expected cardinality of the Car relation after the initial selections are applied:

You can only push down the Car.company = 'Volvo' selection predicate.

Given NDistinct(Car.company) = 50. So, Selectivity(Car.company) = 1/50.

*NCardinality(Car.company = 'Volvo') = Selectivity * NTuples(Car)*

*= (1/50) * 10000*

= ***200***

Question 3 – Query Optimization (continued)

b) [4 points] What is the expected cardinality of the Accident relation after initial selections are applied:

You can push down both A.accident_date < '07/01/1970' AND A.damage_amount > 10000. The predicate on accident_date cuts the histogram right in the middle of the year 1970.
Cardinality(A.accident_date < '07/01/1970') = 500 + 1000 + 1500 + (1200/2) = ***3600***

Selectivity(A.accident_date < '07/01/1970') = 3600/10000 = ***(9/25)***

Selectivity(A.damage_amount > 10000) = (16000 – 10000)/(16000 – 1000) = 6000/15000 = ***(2/5)***

*Cardinality(A.damage_amount > 10000) = (2/5)*10000 =* ***4000***

Selectivity(A.accident_date < '07/01/1970' AND A.damage_amount > 10000) = (9/25)(2/5)*

Cardinality(A.accident_date < '07/01/1970' AND A.damage_amount > 10000)

= (9/25)(2/5)*10000*

= ***<u>1440</u>***

Next, estimate the I/O cost for the following access plans in Pass 1. **Be sure to list any assumptions you are making (for example if you are sorting RIDs before accessing data).**

c) [3points] Index scan on Car(company) for the relation Car:

We know that Cardinality(Car.company = 'Volvo') = 200.
Given:
There is a hash index on Car.company
Each hash bucket can hold 100 keys (need two buckets for 200 keys matching the predicate)
*Cost of hash bucket lookup is 1.2 I/Os. (To access 2 buckets, need 2*1.2 = 2.4 I/Os)*
The hash index is unclustered (To access 200 tuples, you may need 200 I/Os)
Hence, total I/O cost = 2.4 + 200 = ***<u>202.4 I/Os</u>****.*

If we assume that the rids will be sorted, and if we assume that the tuples are evenly distributed among all the 100 Car pages, then at the most we will read all the 100 pages once, with each page containing 2 tuples. Then, the total I/O cost will be = 2.4 + 100 = ***<u>102.4 I/Os</u>***

d) [3 points] Index scan on Accident(accident_date) for the relation Accident:

We know that Cardinality(A.accident_date < '07/01/1970') = 3600.
Given:
There is a clustered BTree index on Accident(accident_date)
Cost of traversing to first matching BTree leaf node = 2 or 3, depending upon height.
Each BTree node can hold 100 keys (need to traverse at least 36 BTree nodes to get 3600 keys)
Each data page can hold NTuples/NPages tuples i.e. 10000/1000 = 10 tuples per data page. To access 3600 tuples, you only need to read 3600/10 = 360 pages, since BTree is clustered.
Hence, total I/O cost = 2 (or 3) + 36 + 360 = ***<u>398 (or 399) I/Os</u>****.*

e) [3 points] Index scan on Accident(damage_amount) for the relation Accident:

We know that Cardinality(A.damage_amount > 10000) = 4000.
Given:
There is an unclustered BTree index on Accident(damage_amount)
Cost of traversing to first matching BTree leaf node = 2 or 3, depending upon height.
Each BTree node can hold 100 keys (need to traverse at least 40 BTree nodes to get 4000 keys)

Since the BTree is unclustered, the 4000 tuples could be anywhere in the table pages. If you do not sort the rids, to access 4000 tuples, you will need to read 4000 pages.
Hence, total I/O cost = 2 (or 3) + 40 + 4000 = ***4042 (or 4043) I/Os****.*

But, if you sort the rids, then you will need to read all the 1000 pages in Accident each once, since the rids could be evenly distributed among all the pages (4 tuples in each page).

Then, total I/O cost = 2 (or 3) + 40 + 1000 = ***1042 (or 1043) I/Os.***

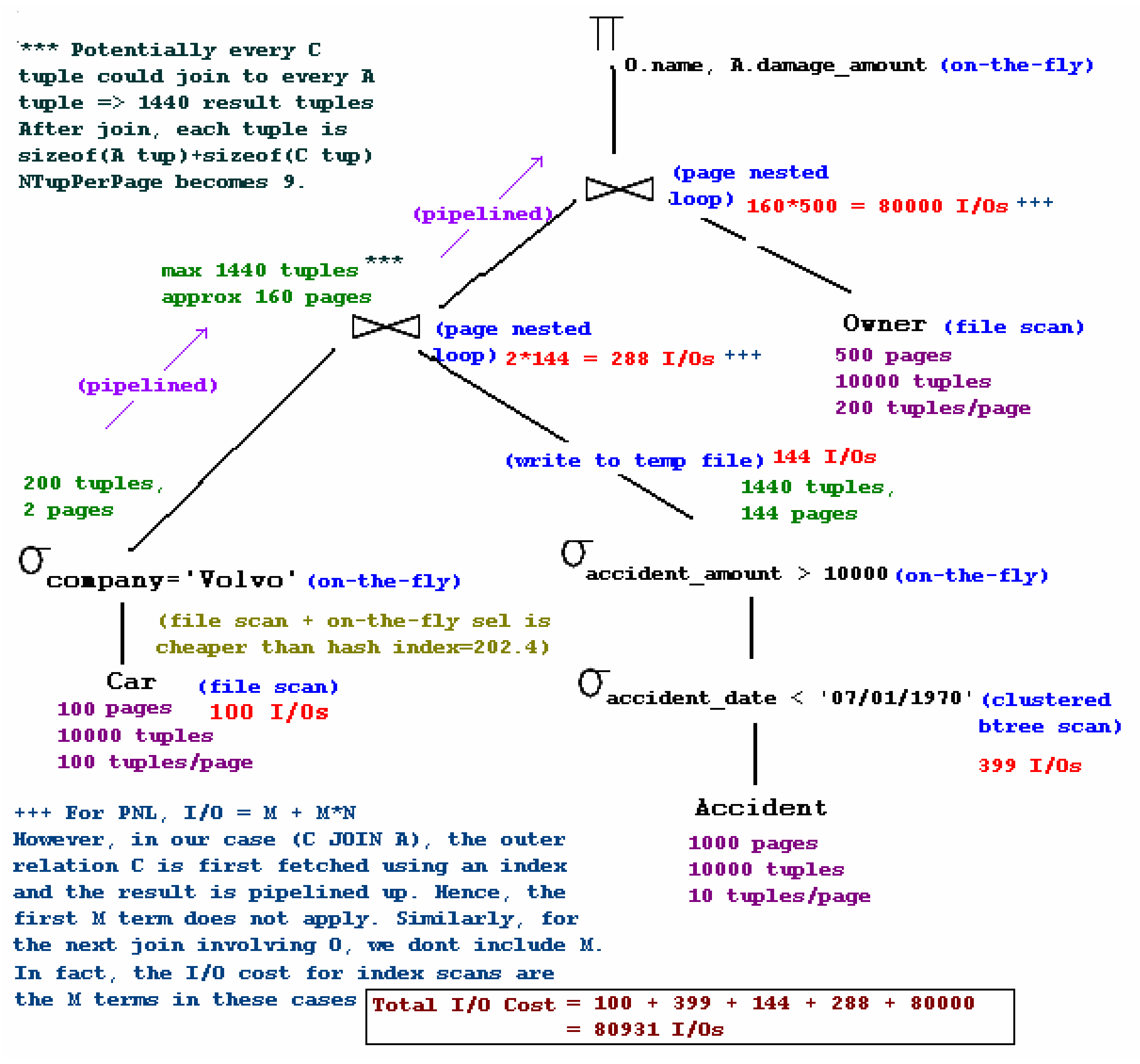
f) [3 points] List all *join orders* that will be considered in Pass 3 by the System R query optimizer. (ignore the specific join algorithm in this step).

If we do not consider a specific join algorithm for Pass 2, then all the left deep join orders will get considered in Pass 3. The answer is an enumeration of all the left deep join orders.

((C, A), O), ((A, C), O), ((O, A), C), ((A, O), C), ((O, C), A), ((C, O), A)

g) [6 points] Suppose page nested loop join algorithm is the only available join algorithm, what is the best join order and what is the total estimated cost?

Best join order is ((C, A), O), since PNL we get fewer I/O if outer relation has fewer pages.


∏ O.name, A.damage_amount (on-the-fly)
(page nested loop) 160*500 = 80000 I/Os +++
(pipelined)
max 1440 tuples ***
approx 160 pages
(page nested loop) 2*144 = 288 I/Os +++
Owner (file scan)
500 pages
10000 tuples
200 tuples/page
(pipelined)
(write to temp file) 144 I/Os
1440 tuples,
144 pages
200 tuples,
2 pages
σ company='Volvo' (on-the-fly)
σ accident_amount > 10000 (on-the-fly)
Car (file scan)
100 pages
10000 tuples
100 tuples/page
100 I/Os
σ accident_date < '07/01/1970' (clustered btree scan)
399 I/Os
Accident
1000 pages
10000 tuples
10 tuples/page

*** Potentially every C tuple could join to every A tuple => 1440 result tuples After join, each tuple is sizeof(A tup)+sizeof(C tup) NTupPerPage becomes 9.

(file scan + on-the-fly sel is cheaper than hash index=202.4)

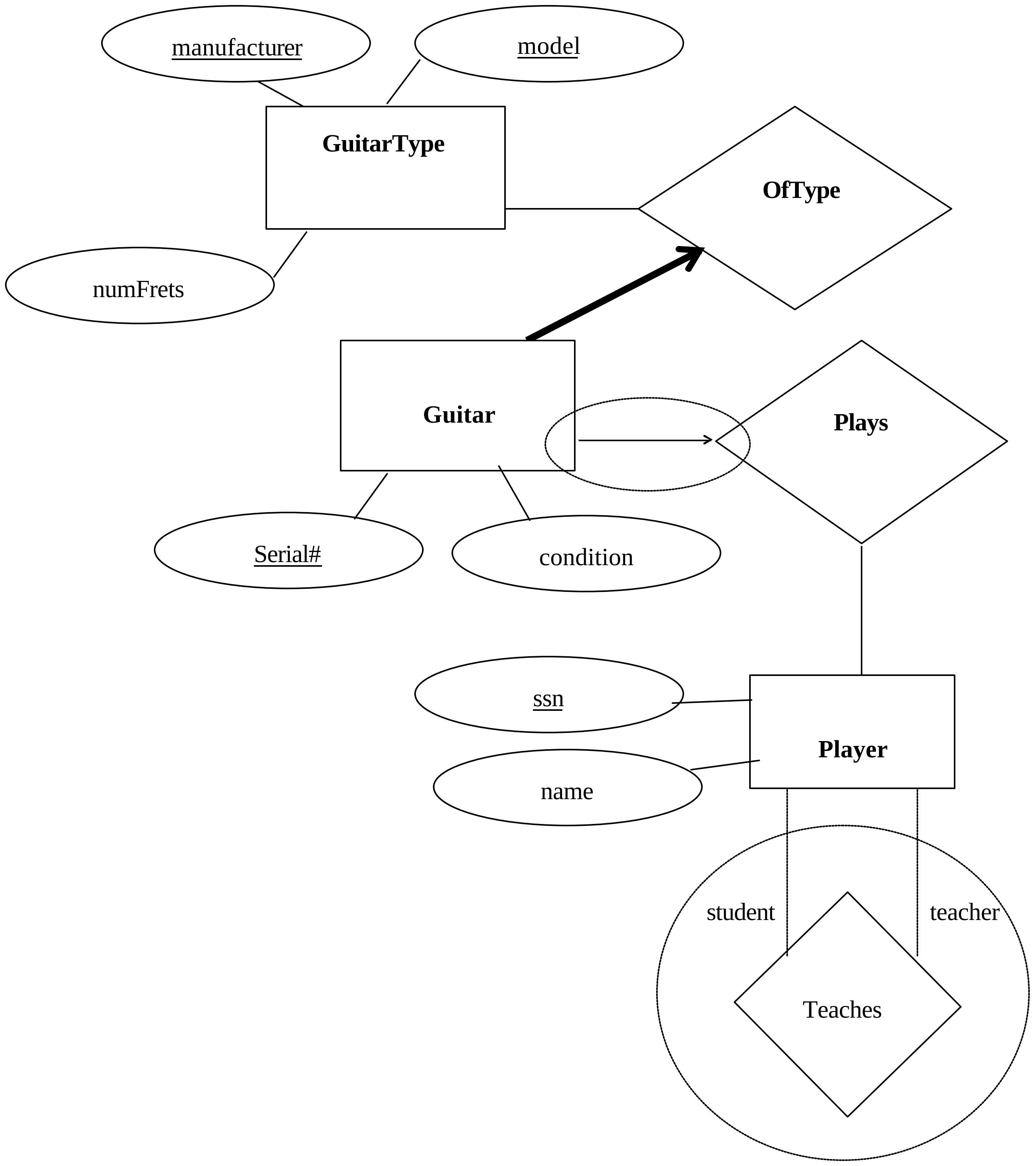
+++ For PNL, I/O = M + M*N However, in our case (C JOIN A), the outer relation C is first fetched using an index and the result is pipelined up. Hence, the first M term does not apply. Similarly, for the next join involving O, we dont include M. In fact, the I/O cost for index scans are the M terms in these cases

Total I/O Cost = 100 + 399 + 144 + 288 + 80000
= 80931 I/Os

SID:____________________

Question 4 – ER models [5 parts, 25 points]

Consider the following ER diagram



Using this diagram, answer the questions that appear on the following pages.

Question 4 – ER models (continued)

a) [10 points] Create a relational schema (with SQL CREATE TABLE statements) for this diagram. Be sure to label all primary and foreign key constraints. The types of the attributes are as follows:

INTEGER: GuitarType.numFrets, Guitar.serial#, Player.ssn

CHAR(20): all others

Note: Your schema for this part should have no more than four tables (solutions with more than four tables will not receive full credit). **NOTE: the solution for question 4 is a sample solution; some minor variations on this could still give you full credit.**

```
CREATE TABLE GuitarType(
   manufacturer CHAR(20),
   model CHAR(20),
   numFrets INTEGER,
   PRIMARY KEY(manufacturer,model));
CREATE TABLE Guitar(
   manufacturer CHAR(20) NOT NULL,
   model CHAR(20) NOT NULL,
   serial# INTEGER PRIMARY KEY,
   condition CHAR(20),
   FOREIGN KEY(manufacturer, model) REFERENCES GuitarType);
CREATE TABLE Player(
   ssn INTEGER PRIMARY KEY,
   name CHAR(20));
CREATE TABLE Plays(
   serial# INTEGER,
   ssn INTEGER,
   PRIMARY KEY(serial#, ssn),
   FOREIGN KEY(ssn) REFERENCES Player,
   FOREIGN KEY(serial#) REFERENCES Guitar);
```

Question 4 – ER models (continued)

b) [2 points] Say that we want to impose an additional constraint that a guitar can be played by at most one player. Indicate this new constraint **on the original diagram** for this question (**Be sure that the change is clearly indicated, there will be no regarding on this part).**

See arrow in dashed ellipse in diagram.

c) [4 points] For the change in part b, indicate (below) how your CREATE TABLE statements would have to be changed to reflect this.

Add a field ssn to Guitars, which is a foreign key that references Players, and drop the relation Plays

OR

Make the primary key of Plays the field serial# only.

d) [4 points] Now, say that we want to add the fact that some players are teachers that teach other players. Teachers can teach multiple players and players can have multiple teachers. Indicate this new information **on the original diagram** for this question (**Be sure that the change is clearly indicated, there will be no regarding on this part).**

See new relationship in dashed circle in diagram. You could also create a subclass for Teachers off of Students with ISA and use this and Students in the relationship, but this is not necessary for full credit.

e) [5 points] Write the CREATE TABLE statement(s) that capture this information. Note, this may cause you to change one or more of your original CREATE TABLE statements.

Add the following table:

```
CREATE TABLE Teaches(
    student_ssn INTEGER,
    teacher_ssn INTEGER,
    PRIMARY KEY(student_ssn, teacher_ssn),
    FOREIGN KEY student_ssn REFERENCES Player(ssn),
    FOREIGN KEY teacher_ssn REFERENCES Player(ssn));
```

Question 5 – Functional Dependencies [4 parts, 15 points]

Consider a database table T with attributes ABCDE and a set of functional dependencies

FD ={AE->BC, AC->D, CD->BE, D->E}

a) [6 points] Give three (3) *candidate* keys (if there are more than three, choose any three you want), and explain why they are *candidate* keys (i.e., in addition to being superkeys).

AE, AC and AD are candidate keys, as each of their attribute closures include all attributes and no subset of them is a super key by itself.

b) [3 points] Is table T already in BCNF? Why or why not?

No, because both CD -> BE and D -> E violate BCNF. In both cases, the left hand side is not a super key.

Now, consider the following table R with attributes ABCD and with the set of functional dependencies FD = {A->B, B->C, C->D}

c) [3 points] Say you decompose it into AB, CD, AC. Is this decomposition lossless? Explain why or why not.

Yes, a lossless decomposition would be: ABC CD which is lossless because C is a key for CD and then a further decomposition of ABC into AB and AC which is lossless because A is a key for AB.

d) [3 points] Give another BCNF decomposition of relation R, which is different from the one in part (c). Your decomposition should be lossless, but need not be dependency preserving.

One such composition is AB AD BC; another is AB BC CD.