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

Fall 2000 Prof. Michael J. Franklin

Midterm Exam - SOLUTIONS

October 18, 2000

CS 186 Introduction to Database Systems

NAME: J. Bond **STUDENT ID:** 007

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: Saturday 10pm TA NAME: E.F. Codd

General Information:

This is a **closed book** examination – but you are allowed one 8.5" x 11" sheet of notes (double sided). You have 1 hour and 30 minutes to answer as many questions as possible. Partial credit will be given. 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.

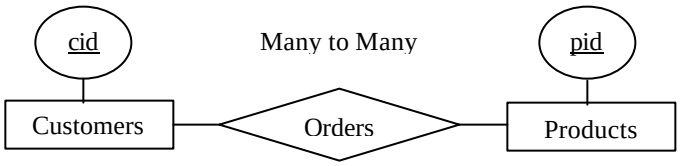
Please try to be as **concise as possible**.

GOOD LUCK!!!

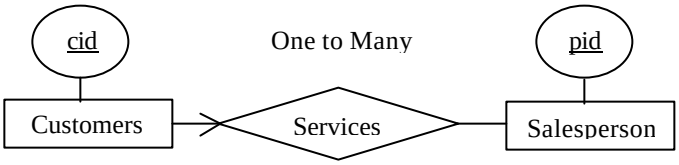
Problem	Possible	Score
1. Data Models (3 parts)	**20**	**20**
2. Formal Relational Languages (3 parts)	**15**	**15**
3. SQL (4 parts)	**25**	**25**
4. Disks, Pages, Buffer Mgmt (3 parts)	**15**	**15**
5. Indexes and File Organization (4 parts)	**25**	**25**
TOTAL	**100**	**100**

Question 1 [3 parts, 20 points total]: Data Models

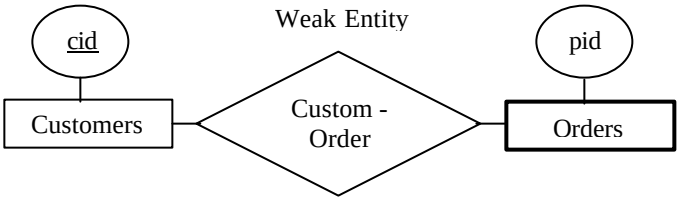
a) (10 points) Draw a (simple) E-R diagram that results in a primary key/foreign key constraint to be created between tables. Show the SQL statements that create the tables including the foreign key and primary key indications.




```
CREATE TABLE Customers (
    cid CHAR(10),
    primary key (cid))
CREATE TABLE Products (
    pid CHAR(10),
    primary key (pid))
CREATE TABLE Orders (
    cid CHAR(10),
    pid CHAR(10),
    PRIMARY KEY (cid, pid),
    FOREIGN KEY (cid) REFERENCES Customers,
    FOREIGN KEY (pid) REFERENCES Products)
```




```
CREATE TABLE Salespersons (
    sid CHAR(10),
    primary key (sid))
CREATE TABLE Customers(
    cid CHAR(10),
    sid CHAR(10),
    PRIMARY KEY (cid),
    FOREIGN KEY (sid) REFERENCES Salespersons)
```




```
CREATE TABLE Customers (
    cid CHAR(10),
    primary key (cid))
CREATE TABLE Orders (
    cid CHAR(10),
    pid CHAR(10),
    PRIMARY KEY (cid, pid),
    FOREIGN KEY (cid) REFERENCES Customers)
```

b) (5 points) For the relational tables you generated in question 1(a), Describe which **insert** and **delete** operations in this database must be checked to ensure that referential integrity is not violated for that foreign key. Please state specifically which operations on which relations can cause problems.

Many to Many:

On insert(Order) -> exists(Customers) and exist(Products);

On delete(Customers) -> delete(Orders) or not allowed;

On delete(Products) -> delete(Orders) or not allowed;

One to Many:

On insert(Customers) -> exists(Salespersons);

On delete(Salespersons) -> delete(Customers) or not allowed if the foreign key can not be null. Otherwise set_default(Customers).

Weak entity:

On insert(Order) -> exists(Customers);

On delete(Customers) -> delete(Orders) or not allowed;

c) (5 points) Consider a database of employees in which we need to record information about employees' addresses. Name one condition which would cause you to make "address" an **entity set** of its own rather than an **attribute** of the employee entity set.

There are several conditions.

- An employee may have more than one address and all of them are supposed to be stored in the database.
- Attribute address is composed of street, city, state, etc. Components of an address may also be interesting to some queries.

Address of an employee may be shared by another entity.

Question 2 [3 parts, 15 points total]: Pure Relational Languages

Consider the following schema for an airline database (primary key attributes are in **bold**):

FLIGHTS (**flight_num**, source_city, destination_city)

DEPARTURES (**flight_num, date,** plane_type)

PASSENGERS (**passenger_id**, passenger_name, passenger_address)

BOOKINGS(**passenger_id**, **flight_num, date**, seat_number)

Express the following queries in **one of** (your choice): relational algebra or relational calculus.

Feel free to use different languages for different querires and to abbreviate relation and attribute names:

a) (5 points) Find the cities that have direct (non-stop) flights to both Honolulu and Newark.

$\pi_{source\_city}(\sigma_{destination\_city = \text{“Honolulu”}}(FLIGHTS)) \cap \pi_{source\_city}(\sigma_{destination\_city = \text{“Newark”}}(FLIGHTS))$

$\{P \mid \exists F \in FLIGHTS, \exists G \in FLIGHT (P.source\_city = F.source\_city \wedge P.source\_city = G.source\_city \wedge F.destination\_city = \text{“Honolulu”} \wedge G.destination\_city = \text{“Newark”})\}$

b) (5 points) Find the passenger_name of all passengers who have a seat booked on at least one plane of **every** type.

$\pi_{passenger\_name, plane\_type}(PASSENGERS \infty BOOKINGS \infty DEPARTURES) / \pi_{plane\_type}(DEPARTURES)$

$\{P \mid \exists P1 \in PASSENGERS (\forall D1 \in DEPARTURES (\exists B \in BOOKINGS, \exists D2 \in DEPARTURES ($

$P.passenger\_name = P1.passenger\_name \wedge P1.passenger\_id = B.passenger\_id \wedge B.flight\_num = D2.flight\_num \wedge B.date = D2.date \wedge D1.plane\_type = D2.plane\_type)))\}$

c) (5 points) Find the flight_num and date of all flights for which there are no reservations.

$\pi_{flight\_num, date}(DEPARTURES) - \pi_{flight\_num, date}(BOOKINGS)$

$\{P \mid \exists D \in DEPARTURES (P.flight\_num = D.flight\_num \wedge P.date = D.date \wedge \forall B \in BOOKINGS (D.flight\_num <> B.flight\_num \vee D.date <> B.date))\}$

Question 3 [4 parts, 25 points total]: SQL

Consider the relational schema of question 2. Express the following queries in SQL (feel free to abbreviate relation and attribute names and to use INTERSECT and EXCEPT if you need to):

a) (5 points) Find the cities that have direct (non-stop) flights to both Honolulu and Newark

```
SELECT DISTINCT source_city
FROM Flights F
WHERE F.dest_city = "Honolulu"
 AND F.source_city IN
   (SELECT source_city
     FROM Flights F2
     WHERE dest_city = "Newark")
```

Could also be done with a self join on Flights, or with INTERSECT, or…, just can't use a simple selction with "AND" in the Where clause --- this would return no tuples

b) (5 points) Find the passenger_id of all passengers who have a seat booked on a plane of type ``747" from San Francisco to Washington. **Do not return any duplicate values.**

```
SELECT DISTINCT B.passenger_id
FROM Flights F, Departures D, Bookings B
WHERE B.flight_num = D.flight_num
 AND B.date = D.date
 AND F.flight_num = D.flight_num
 AND F.source_city = "San Francisco"
 AND F.destination_city = "Washington"
 AND D.plane_type = "747"
```

Since key of Departures is flight_num and date, you need both of these to do the join and find out what type of plane the passenger is booked on.

c) (7 points) Find the passenger_name of all passengers who have a seat booked on at least one plane of **every** type.

```
SELECT DISTINCT passenger_name
FROM Passengers P
WHERE
 (SELECT COUNT(DISTINCT D.plane_type)
   FROM Departures D, Bookings B
   WHERE D.flight_num = B.flight_num
   AND D.date = B.date
   AND B.passenger_id = P.passenger_id)
 =
  (SELECT COUNT(DISTINCT D.plane_type)
   FROM Departures D)
```

Alternatively:

```
SELECT DISTINCT P.passenger_name
FROM Passengers P
WHERE NOT EXISTS
 (SELECT D.plane_type
   FROM Departures D
   WHERE NOT EXITS
    (SELECT *
     FROM Departures D2, Bookings B
     WHERE D2.flight_num = B.flight_num
     AND D2.date = B.date
     AND B.passenger_id = P.passenger_id
     AND D.plane_type = D2.plane_type))
```

The first one is easier and does it by counting the number of plane types(we used a similar technique in class). The second is trickier but is similar to the techniq ue used in the book.

d) (8 points) Print an ordered list of all source cities and the number of distinct destination cities that they have direct (non-stop) flights to. The list should be ordered in decreasing number of destinations and should contain **only those source cities that have flights to 25 or more distinct destinations**.

For example, the output should look like:

Source_City	NumDestinations
Chicago	120
Atlanta	106
Boston	97
...	...
Austin	25

```
SELECT source_city,
       COUNT(DISTINCT destination_city) AS NumDestinations
FROM Flights F
GROUP BY source_city
HAVING NumDestinations >= 25
ORDER BY NumDestinations DESC
```

Question 4 [3 parts, 15 points total]: Disks and Buffer Management

a) (3 points) The main components of the cost of performing a disk read are **seek time, rotational delay**, and **transfer time**. For each of these three components state whether or not it is reduced by doing **sequential** reads rather than **random** reads*:*

Component	**Reduced by sequential? Yes or No**
Seek Time	Yes
Rotational Delay	Yes
Transfer Time	No

b) (2 points) For the question above, which of the three is likely to result in the largest savings when comparing sequential reads to random reads? (No explanation necessary)

Seek Time

c) (10 points) Consider a page reference pattern that performs **three** consecutive scans over a set of **five** pages. Assume you start with an empty buffer pool of **three** frames. 1) How many page faults will be incurred with an **LRU** page replacement policy, and 2) how many will be incurred with an **MRU** page replacement policy?

1) 15 page faults

Frames v	Read Page >>	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		1	1	1	4	4	4	2	2	2	5	5	5	3	3	3
2			2	2	2	5	5	5	3	3	3	1	1	1	4	4
3				3	3	3	1	1	1	4	4	4	2	2	2	5
Pagefault?		Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y

2) 9 Page faults

Frames v	Read Page >>	1	2	3	4	5	1	2	3	4	5	1	2	3	4	5
1		1	1	1	1	1	1	1	1	1	1	1	2	3	3	3
2			2	2	2	2	2	2	3	4	4	4	4	4	4	4
3				3	4	5	5	5	5	5	5	5	5	5	5	5
Pagefault?		Y	Y	Y	Y	Y	N	N	Y	Y	N	N	Y	Y	N	N

Question 5 [4 parts, 25 points total]: Indexes and File Organization

a) (5 points) Suppose that you have a file that is already **sorted** in key order and you want to construct a dense, clustered B+ tree index on this file using <key, RID> pairs for data entries. A simple way to accomplish this is to create a B+tree, and then sequentially scan the file, inserting an index entry for each record using the normal B+tree insertion routine. What **performance and storage utilization problems** are there with this approach?

Performance and storage utilization problem: Most leaf nodes are half full as a result of inserting sorted key values and splitting nodes at the leaf level in a way that each of the two nodes is half full. So utilization is roughly 50%. Also because the tree is large, the performance is not good.

Another performance problem is that to insert each key, the B+ tree is traversed from root to the leaf.

b) (4 points) Briefly describe a change to the B+tree insertion routine that would solve the problems you identified in part 5(a).

Two solutions:

- One is to change the splitting method. When a leaf node is split, do not move half key values to the newly created node. Just move the last key to the new node so that the left node is still full.
- Bulk loading. That is to build B+ tree bottom up from sorted key values.

c) (6 points) Circle the basic file organization (heap, sorted, or hash) that is best for a large file where the most frequent operations are as follows (answer each separately – no explanation needed):

1) Search for records based on a range of field values.

HEAP **SORTED** **HASH**

2) Perform inserts and scans where the order of records does not matter.

HEAP **SORTED** **HASH**

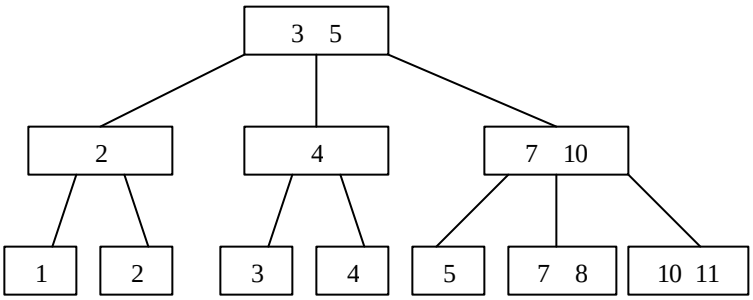
3) Search for a record based on a particular field value.

HEAP **SORTED** **HASH**

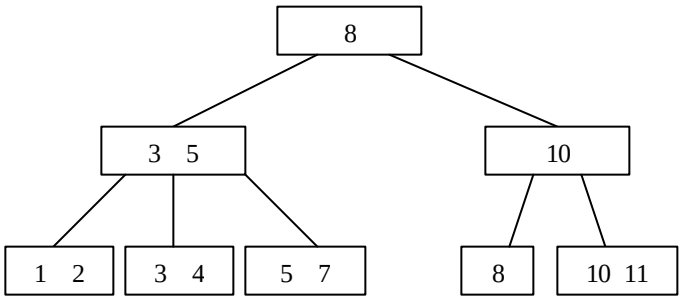
d) (10 points) Create a B+tree where each node can hold at most 3 pointers and 2 keys when the following keys are inserted in the following order:

1, 10, 2, 11, 3, 4, 8, 5, 7

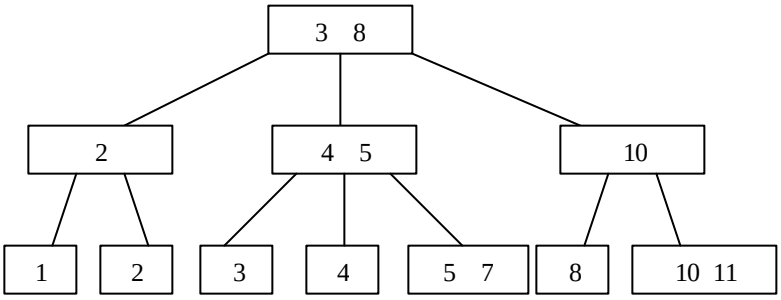
Tree 1:



Tree 2:



Tree 3:



Tree 4 (different search algorithm):