CS186 Midterm solutions
Spring 2003

1. Disks and Buffers

1a. 20000*2 = 40000 (-2 if answer 20000)

1b. (1024-20)/100 = 10 records/block
10000/10 = 1000 blocks

1c. B+1 or B+2 (B buffers for inner loop, 1 buffer for outer loop. If you consider output buffer, add 1 buffer.)

2. Hashing

2(a) The size of the Applicants relation is potentially infinite, and only limited by the number of distinct values (for amongst the columns projected in the query).

Partial credit given for a solution starting with a stronger assumption (eg. "assuming that there are no duplicates in the projected columns, N is limited by the amount of memory available for hashing")

2(b). 2N

Those who attempted to account for projection savings and came up with 2N*s where "s" is the projection size ratio were given full credit.

A small amount of partial credit was given for coming up with the 2N term provided it was properly explained. This is for people who came up with (2N + <some-gross-term>).

2(c).

Expected answer:

Symptom (3 points): Skewed partitioning leads to trouble while rehashing with too many _distinct_ tuples that can be accomodated in the hash table. (It's not enough to say that "a partition can grow too big" but a small amount of partial credit was given).

Compensation (3 points): recursive partitioning

Alternate answer for compensation (given full credit): switch strategies to sorting on the fly.

2(d).

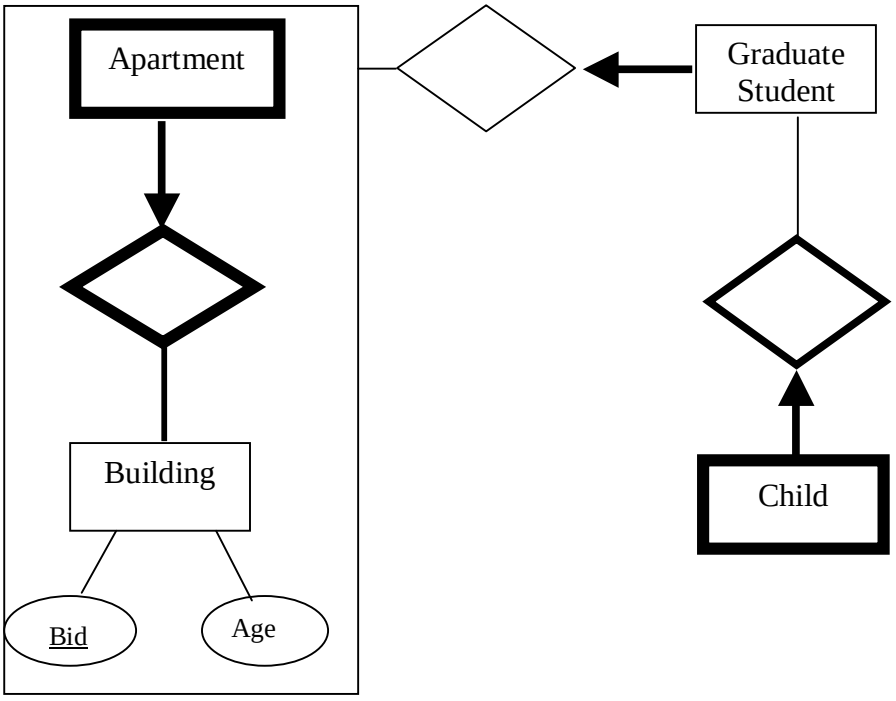
Will hashing work ? No (1 point irrespective of reason)
Is sorting better ? Yes (1 point irrespective of reason)
Why won't hashing work ? (4 points)

The hw2 implementation supports grouped aggregates. In the query Q2, however, we need to evaluate "count (distinct haircolor)" that is grouped by "homestate". This means that for each unique group in the hash table, we need to perform a "nested aggregate". This is not supported by the hw2 implementation. Sorting would be better as it's easier to remove tuples with identical "homestate,haircolor" pairs when sorting with "homestate,haircolor" as a key.

Full credit was also given for the following answer:
Hashing with the hw2 implementation will work provided the aggregate in question is capable of accumulating all distinct haircolors in its <transValue> since the default behaviour won't work (has to include explanation in the above paragraph).

3. ER

3a)



Apartment and Child are weak entities.

Due to space constraints, we have left out putting in the attributes. They are as follows:
Building: Bid, Age

Apartment: Aid, capacity
GraduateStudent: SID, name, age, sex, department
Child: Name, Age

Apartment.Aid and Dependent.Name are partial keys.

3b)

Table	Attributes	Primary Key	Foreign Keys
Building	Bid: integer, Age: integer	Bid	
Apartment	Aid: integer, Bid: integer, capacity: integer	Aid, Bid	Bid references Building
GraduateStudent	SID: integer, name: string, age: integer, sex: string, department, Aid: integer, Bid: integer	SID	Aid references Apartment, not null, Bid references Building not null
Child	Name: string, SID: integer, Age: integer	Name, SID	SID references GraduateStudent

3c)
Each building must have at least one apartment. (Participation constraint)
OR
Apartment can only have up to *capacity* number of occupants.

4. Relational Algebra and Calculus

4a. { P | P ∈ Players and !∃(P1) ∈ Players (P1.team = P.team and P1.height > P.height) }

4b.
{ P | P ∈ Players ∀(G)∈ Game (G.awayTeam = P.team => (∃(GS)∈ GameStats
(GS.playerID = P.playerID and GS.gameID = G.gameID)

4c. π ((playerID, gameID) (Player JOIN ($\sigma_{(assist>=10,\ rebounds>=10,\ points>=10)}$ GameStats)))

4d. π_{name} ($\sigma_{(team="ChicagoBulls")}$ Player JOIN ($\pi_{playerID,\ gameID}$ GameStats / π_{gameID} ($\sigma_{(hometeam='ChicagoBulls' \text{ or } awayteam='ChicagoBulls')}$ Game))
OR
π_{name} ((($\sigma_{(team="ChicagoBulls")}$ Player) JOIN ($\pi_{playerID,\ gameID}$ GameStats)) / π_{gameID} ($\sigma_{(hometeam='ChicagoBulls' \text{ or } awayteam='ChicagoBulls')}$ Game))

5. Indexing

a) 8

b) Many people lost 4 points on this question because their heap file did not end up matching the picture. This problem was intended to exercise your understanding of *clustered* indexes, and hence the heap file structure is significant. It should have no "holes", and be slightly out of order (Louis after Ornette).

One correct solution is:
Insert Charles
Insert Clifford
Insert Lester
Insert Ornette
Create (Bulk load) index
Insert Ferd
Delete Ferd
Insert Louis

c)