**14.3**

1. 输出时的有序段每页只包含tittle和ename信息，且每个属性都是等长的的字符串，故输出的每个有序段长度为20，共有10 000/2/20 = 250 个有序段。共进行了10 000 + 10 000/2 = 15 000次I/O。
2. 共需⌈log10 250⌉=3遍处理，每次处理5000个记录的输入输出，故总共为3\*2\*5 000 = 30 000次I/O。
3. 1. 有title的聚簇索引时将会只进行一遍扫描，总共需要10 000 + 2 500（B+树的读取） = 12 500次I/O。非聚簇的索引则需要2 500 + 10 000\*10（每页都需要读取至少十遍，因为title可能有重复）= 102 500次以上的I/O次数。索引是否为哈希索引没有影响，主要取决于索引是否为聚簇的。
   2. 跟a的情况类似，不过由于ename是键，没有重复，故非聚簇索引为102 500次I/O。
   3. 建立在<ename, title>上的索引可以使得遍历只在索引上进行，故无论是聚簇还是非聚簇都只需要5 000（10 000 / 2）次的I/O。
4. 这样的话就只需便利一遍元组，把不需要的信息剔除就可以了。实际上因为ename是键，所以本来也就不需要剔除重复。

**14.4**

关系R有10 000/10 = 1 000个数据页，关系S有2 000/100 = 200个数据页。

1. 总共需要的I/O次数为1 000\*200 + 200 = 200 200次。最少只需用3个数据页。
2. 总共需要的I/O次数为200\*(1 000 / (52 – 2)) + 200 = 4 200次。最少需用52个数据页。
3. 总共需要的I/O次数为3\*(1 000 + 2)) = 3 600次。最少需用1 000/50 + 200/50 + 1=25个数据页。(50是可用缓冲区个数)
4. 总共需要的I/O次数为3\*(1 000 + 2)) = 3 600次。最少需用满足B> 的B个数据页
5. 总共需要的I/O次数为1 000 + 2 = 1 200次。最少需用200+1+1 = 202个数据页
6. 由于S.b是键，即其无重复，故可生成的最多元组数为R的元组数，即10 000，当每个R.a都有对应的S.b时成立。合并后的元组大小可能翻倍，故每页可能存储的元组为5，此时共需要10 000/5 = 2 000页。
7. 排序归并和哈希归并无影响，对于简单嵌套循环连接其I/O次数变为：1 000+(1 000\*200/2) = 101 000.块嵌套循环连接算法变为1 000 + 200/2\*(1 000/(50 – 2)) = 3 000次。

**15.2**

1. 假设每页有2048/100 = 20 个元组
   1. B+索引总共需要10 000\*(20/100)\*0.1 + 10 000\*20\*0.1 = 22 000而直接遍历文件则只需10 000次，故直接遍历记录文件会比较好。
   2. 10 000\*20\*0.1 = 20 000次I/O。
   3. 2 + 10 00\*0.1 + 200 = 1202
   4. 1.2+1 = 2
   5. 和c一样，是1202
   6. 2+10 000\*0.1\*0.1 + 10 000\*0.4\*0.1\*0.1 = 142
   7. 文件遍历，10 000
   8. 2 + 10 000\*0.1 = 1002
2. 1. 只需遍历一遍索引sal，2 + 10 000\*0.1\*0.2 = 202
   2. 只需遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   3. 只需遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   4. 1.2+1 = 2
   5. 只需遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   6. 2+10 000\*0.1\*0.1 + 10 000\*0.4\*0.1\*0.1 = 42
   7. 文件遍历，10 000
   8. 2 + 10 000\*0.1 + 10 000\*0.1\*0.4 = 1402
   9. 遍历一遍索引sal，2 + 10 000\*0.4 = 4002
   10. 遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   11. 遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   12. 1.2+1 = 2
   13. 遍历一遍索引<age, sal>，2 + 10 000\*0.1\*0.4 = 402
   14. 2+10 000\*0.1\*0.1 + 10 000\*0.4\*0.1\*0.1 = 42
   15. 2 + 10 000\*0.4+10 000\*0.1 + 5 000\*0.1 = 24 002
   16. 2 + 10 000\*0.1 + 5 000\*0.1\*0.4 = 1502
   17. 16000 + 12 + 10000 \* 0.4 \* 0.1 + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.
   18. 16000 + 10000 \* 0.4 = 20000.1.2+1 = 2
   19. 2 + 10 000\*0.1\*0.4 = 402
   20. 1.2+1 = 2
   21. 16000 + 12 + 10000 \* 0.4 \* 0.1 + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.
   22. 16000 + 12 + 10000 \* 0.4 \* 0.1 + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.
   23. 16000 + 12 + 10000 \* 0.4 \* 0.1 + 2 = 16000 + 4000 + 12 + 400 + 2= 16414.
   24. 2 + 1000 \* (0.2 + 20) =2202
   25. 文件遍历
   26. 文件遍历
   27. 还是文件遍历

**15.5**

1. R有10\*3 = 30 个元组，S有100 \* 2 = 200个元组，又每个S只和一个R连接，故总共最多可以有200个元组。又每个元组为450bytes，每页只能存放2个元组，故200/2=100页会被写回磁盘
   1. 200\*[1og2 100] + (6+6\*30) = 1586
   2. (10 + 10\*100) + 200+100+2\*6\*100 = 2410
   3. 1010
   4. 将以上的log2换做log10

**15.6**

1. 用于模型化查找语句以便找到理想的查找计划

(a) σA=1(πABCD(R × S))

= πABCD(σA=1(R) × S)

(b) σA=1,C=2(πABCD(R × S))

= πABCD(σA=1,C=2(R × S))

= πABCD(σA=1(R) × σC=2(S))

(c) σC=5(πBC(R × S))

= σC=5(πC(πB(R) × S))

(d) σB=1,C=3(πBC(R × S))

= σB=1,C=3(πC(πB(R) × S))

= σB=1(σC=3(πC(πB(R) × S)))

= σB=1(πC(σC=3(πB(R) × S)))

(e) σB=1,C=3(πBC(R × S))

= σB=1,C=3(πC(πB(R) × S))

= σB=1(σC=3(πC(πB(R) × S)))

= σC=3(πC(πB(σB=1(R)) × S))

(f) σA=1,B=D(πBC(R × S))

= πBC(σB=D(σA=1(R) × S)) = πBC(σB=D(πBCD(σA=1(R) × S))

**15.7**

2. 如果只考虑左路归并的话，有两个归并顺序：(D,E,F)和(D,F,E)
3. 1. Emp: 50000 \* 1/50 \* 1/200 = 5

Dept: 2 500

Finance: 5 000



**20.1**

1. 在Emp上建立ename上的哈希索引可以满足要求三；

在Dept上建立floor的聚簇B+索引可以满足要求六

<age, sal>的非聚簇B+索引可以满足要求四、要求五和要求一

为满足要求二，首先在Dept上建立<dname, did>的非聚簇哈希索引，之后再Emp上建立deptid的非聚簇B+索引。

1. 在Emp上建立ename的B+索引，满座要求一

在Emp上建立sal的非聚簇B+索引以满足要求二

在Emp上建立<deptid, sal>的B+非聚簇索引以满足要求三

在Dept上建立<floor, budget>的非聚簇索引以满足要求四

**20.6**

1. 可以用堆文件进行存储，优化查询方面可以在Emp上对<did, sal>建立非聚簇B+索引，在eid上建立非聚簇哈希索引，在did上建立非聚簇哈希索引。
2. 可以考虑在Emp关系中添加location属性，并且在Emp关系中对eid建立哈希索引和一个<did, sal>的聚簇B+索引，在Dept关系中建立对did的哈希索引。
3. 可以尝试有序文件的方式进行存储，此时关系需要变为Emp(eid, did, location) 和 Dept(did, eid, sal, budgat)

**20.8**

1. 有可能是由于age使用了OR关键字，导致优化器不会使用age索引，可以将查询语句改为用UNION连接：

SELECT E.dno

FROM Employee E

WHERE E.age=20

UNION

SELECT E.dno

FROM Employee E

WHERE E.age=10

1. 可能是AND的使用导致查询优化器不会考虑age上的B+索引，可以改为用BETWEEN连接：

SELECT E.dno

FROM Employee E

WHERE E.age<20 BETWEEN E.age>10

1. 查询条件中数学表达式的使用使得查询优化器不会使用age中的索引，可以改为：

SELECT E.dno

FROM Employee E

WHERE E.age<10

1. DISTINCT关键字根本不必要：

SELECT \*

FROM Employee E

1. GROUP BY关键字导致查询器首先根据dno对数据进行排序，然后在组织，可以用WHERE优化：

SELECT \*

FROM Employee E

1. 可能优化器考虑S.sid可能为NULL或者指向多个R.rid,所以先进行了S和R的连接操作，可以改为：

SELECT R.sid

FROM Reserves R

WHERE R.sid NOT NULL

**19.2**

1. ACD, BCD, CDE
2. 是，因为B、E和A都是码或者码的一部分。
3. 不是，因为A、BC和ED都不是超码。

**19.4**

1. 计算一下R中是否有A相同而B不同的记录就好：

SELECT COUNT (\*)

FROM R AS R1, R AS R2

WHERE (R1.A = R2.A) AND (R1.B != R2.B)



CREATE ASSERTION ADeterminesB

CHECK ( (SELECT COUNT (\*)

FROM R AS R1, R AS R2

WHERE (R1.A = R2.A) AND (R1.B != R2.B))

=0)



SELECT COUNT (\*)

FROM R AS R1, R AS R2

WHERE ((R1.A = R2.A) AND((R1.B != R2.B)

OR ((R1.B is NULL) AND (R2.B is NOT NULL)))

CREATE ASSERTION ADeterminesBNull

CHECK ((SELECT COUNT (\*)

FROM R AS R1, R AS R2

WHERE ((R1.A = R2.A) AND((R1.B != R2.B)

OR ((R1.B is NULL) AND (R2.B is NOT NULL)))

=0)

**19.7**

1. B
2. 2NF
3. 可以分解为：AC, BC, CD
4. BD
5. 1NF
6. 可以分解为：AC, BC,BD
7. ABC, BCD
8. 3NF
9. 由于不可能同时保存依赖ABC→D，并且D→A而且A不是超键，所以对于R不可能分解成BCNF。
10. A
11. 2NF
12. 可以分解为：ABC, BCD
13. AB, BC, CD, AD
14. 3NF
15. 无法分解为BCNF。

**19.10**

1. BD. 不是好分解，因为BC和AD连接后形成的关系可能会大于原关系。
2. AB, BC 可能是个好分解，但加入关系ABC后就可以保持全部函数依赖而不需要连接操作。
3. A, C R本身已经是BCNF了，所以分解是没必要的。
4. A 分解出的ACD不是3NF，并且分解没有保持B→C的函数依赖。
5. A 没有保持B→C的函数依赖，一个更好的选择是 AB, BC, CD.