Relational Algebra:

Problem D. Potpourri: 30 points (5 Points per question):

Please answer the following questions. You must give a YES—NO answer to questions D1–D4 and if your answer is YES you must also write the equivalent RA expression.

D1 Can the intersection of relations R(A,B) and S(A,B) be expressed using only natu- ral joins?

Answer: Yes. $R \cap S = R \bowtie S$.

D2 Can the intersection of relations R(A,B) and S(A,B) be expressed using the set difference operator?

Answer: Yes. R∩S=R−(R−S).

D3 Can the intersection of relations R(A,B) and S(A,B) be expressed using the cartesian product and projection operator

Answer: No.

D4 Can the intersection of relations R(A,B) and S(A,B) be expressed using the carte- sian product, selection and projection operators?

Answer: Yes.

$$R \cap S = \pi_{R.A,R.B}(\sigma_{R.A=S.A \wedge R.B=S.B}(R \times S)).$$

Every condition is evaluated as True, False or Unknown.

σC (R) -> filters out rows in a relation

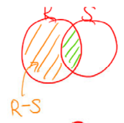
πA(R) -> filters out columns in a relation. Select column aka

R X S -> cross product, concatenates tuples from pairs of tuples

Natural Join -> shorthand for σ_student.sid = enroll.sid(R X S) for joining Student and Enroll

ρS(R) -> rename S to S ρS(A1,A2)(R) -> rename R to S(A1, A2) including attribute names

Intersect -> R and S = R – (R – S)



Core relational operators:

- σ, π, $\times$, $\cancel{\bowtie}$, ρ, $\cup$, $\cancel{\cap}$, $-$

A - B

- Q: which ones are "core" and which ones can be expressed with others?

Null handling:

How are NULL values handled for UNION ALL, INTERSECT ALL, and EXCEPT ALL?

Very much like regular values. {NULL } UNION ALL {NULL } = { NULL, NULL } for example.

Arithmetic operators with NULL input returns NULL

SQL only returns only True tuples except for aggregate functions

Aggregates are computed ignoring NULL values except COUNT(*)

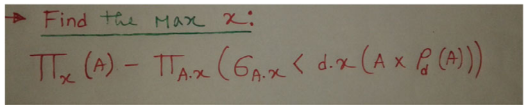
When the input to an aggregate is empty, count returns 0; all others return NULL

Coalesce() function:

Returns FIRST NON NULL value in the list (for example if I had a COALESCE(total – average, 0) and the first one was null, then it gets 0.

4. The relation **Company(company-name, valuation)** captures Company-valuation information, where **company-name** is the name of a company and **valuation** is its valuation. Write a relational algebra expression to find the name of the lowest valued companies. (Hint: When a query is difficult, think of its complement.)

$$\Pi_{\text{Company-name}}(\text{Company}) - \Pi_{\text{C1.Company-name}}(\sigma_{\text{C1.valuation}>\text{C2.valuation}}(\rho_{C1}(\text{Company}) \times \rho_{C2}(\text{Company})))$$



→ Find the Max x:

$\Pi_x(A) - \Pi_{A.x}(\sigma_{A.x < d.x}(A \times \rho_d(A)))$

ER Model

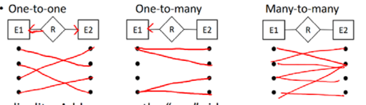
The graphical intuitive and informal representation of data, entities relations and attributes

Contains: Entity, entity set, key, relationship, relationship set.

Cardinality of relationships:

1. One to One each entity in E1 is related to at most 1 entity in E2 and vice versa.
2. Many to One: each entity in E1 is related to at most one entity in E2
3. Many to Many: each entity in E1 may be related to 0 or more entities in E2 and vice versa.
4. Total participation: an entity participates in the relationship at least once.

- Cardinality: how many times entities participate in a relationship?
  - One-to-one
  - One-to-many
  - Many-to-many



- Cardinality: Add arrow on the "one" side
- Total participation
  - every entity participates in the relationship ***at least once***
  - Double line in E/R model

3. This problem is based on an E/R design for a database used in a tech company shown in Figure 1. This database stores information about programmers. Each programmer has a name, which uniquely identifies the programmer. A programmer may in fact be a team leader who in turn leads a team of programmers. For example, Elaine leads a team consisting of Michael and Bryan. Bryan works on project C. Michael works on project A and in turn leads a team consisting of Jane and David who work on project A and B respectively. Each team leader is also associated with the name of the team he leads.

   Convert the E/R diagram to relations. For the translation of subclasses, assume that we generate one table per each subclass, instead of creating one gigantic table for the ISA relationship.

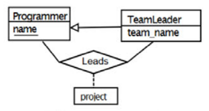
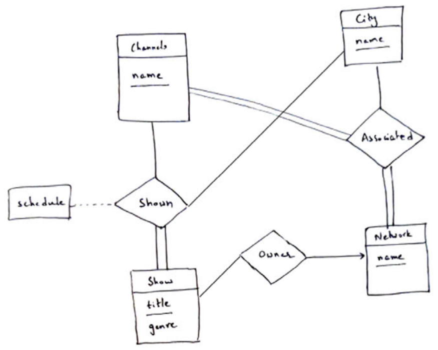


Figure 1: E/R diagram for a tech company

```
Programmer(name)
TeamLeader(name, team)
Leads(TeamLeader.name, Programmer.name, project)
```

Weak Entity Set: entity set without a unique key, double rectangle in ER model.



Many shows in one network, therefore error points to network (as that is the "one").

- A given channel in a given city is associated with one network.
- A given show is either owned by a network (and shown on a channel associated with that network) or is a local show and may be shown on any channel.
- Not all shows are shown in all cities, and the days and times for a given show may differ from city to city.
- You may ignore cable channels, which generally are not city-dependent.

Dashed lines in ER diagrams connect relationship sets to attributes.

SQL

Problem C. SQL — 20 Points

- Given the table: taken(StNo, CourseID, Year, Quarter, Sec, Grade, Remarks):
- write an SQL query to find the students who got a grade less than class average in 7 or more classes they took —a class is identified by (CourseID, Year, Quarter, Sec) and Grade in taken is of type numeric. In your answer, the depth of nesting of sub-queries should not exceed 2.
- ANSWER. We are seeking students who got a grade less than class average in 7 or more classes they took

```
select  StNO from taken as T1
  where  Grade  > (select avg(Grade) from taken as T2
                                        where T2.CourseID=T1.CourseID and
                                       T2.Year= T1.Year and
                                      T1.Quarter= T2.Quarter and
                                    T2.Sec= T1.Sec)
                                    )
group by T1.StNO having count(*) >=7
```

Bag Semantics: a set with duplicate elements and order does not matter. {a, a, b, c} = {a, c, b, a} != {a, b, c}

Under bag semantics, would R U S = S U R (Yes), R ∩ S = S ∩ R? (yes), R ∩ (S U T) = (R ∩ S) U (R ∩ T)? (no)

Joins:

You can have, left outerjoin, right outerjoin, full outerjoin(not available in mysql), R innerjoin s on r.a = s.a, and then r natural join s.

Diff between innerjoin and natural join, innerjoin will return more columns than natural join as it keeps both of the inner join elements (on r.a = s.a) keeps both! But natural join condenses it to just a.

We also do outerjoins if we want null values of a certain table when we combine.

(b) We want to find the movie stars who are not movie executives.

i. Write the query using EXCEPT operator.

ANSWER:
```
(SELECT name FROM MovieStar) EXCEPT (SELECT name FROM MovieExec)
```

ii. Write the query without using EXCEPT operator.

ANSWER:
```
SELECT name FROM MovieStar WHERE name NOT IN (SELECT name FROM MovieExec)
```

```
(SELECT name, address FROM MovieStar WHERE gender='F') INTERSECT (SELECT name,
address FROM MovieExec WHERE netWorth>1000000)
```

ii. Write the query without using INTERSECT operator.

ANSWER:
```
SELECT name, address FROM MovieStar WHERE gender='F' AND (name, address) IN (SELECT
name, address FROM MovieExec WHERE netWorth>1000000)
```

Expression	minimum #tuples	maximum #tuples
$R \cup \rho_{S(A,B)}(S)$		
$\pi_{A,C}(R \bowtie S)$		
$\pi_B(R) - (\pi_B(R) - \pi_B(S))$		
$(R \bowtie R) \bowtie R$		
$\sigma_{A>B}(R) \cup \sigma_{A<B}(R)$		

1. Min: r (when $S \subseteq R$), Max: $r + s$ (when $R \cap S =$)
2. Min: 0 (when all R.B values are different from S.B values), Max: $r \times s$ (when all R.B = S.B = b)
3. Min: 0, Max: s. This expression is equivalent to $\pi_B(R) \cap \pi_B(S)$
4. Min: r, Max: r. R ⋈ R is always R
5. Min: 0 (when A=B for every tuple in R), Max: r (when A≠B for every tuple in R)

(a) $\pi_{R1.B}(\sigma_{R1.B=R2.B \wedge R1.A \neq R2.A}(\rho_{R1}(R) \times \rho_{R2}(R)))$ True

(b) `SELECT B FROM R GROUP BY B HAVING COUNT(*) > 1`

False

(a)
```
SELECT B FROM R
  WHERE NOT EXISTS(SELECT * FROM S WHERE R.B = S.B)
```
(b) `(SELECT B FROM R) EXCEPT (SELECT B FROM S)`

Relational Design Theory

1. Suppose that we decompose the schema $R(A, B, C, D, E, F)$ into (A, B, C, F) and (A, D, E). When the following set of functional dependencies hold, is the decomposition lossless?
$A \to BC, CD \to E, B \to D, E \to A$
Explain your answer.

ANSWER:
(A, B, C, F) INTERSECT $(A, D, E) = A$, and A is a key for (A, D, E), so the decomposition is lossless.

Convert a non BCNF to a BCNF table:

For any R in the schema:

if (nontrivial X -> Y holds on R AND X does not contain a key), then

1. Compute X+ (X+: closure of X)
2. decompose R into R1(X+) and R2(X,Z) X become common attributes Z: all attributes in R except X+

Repeat until no more decomposition

Assume the following set of functional dependencies hold for the relation $R(A, B, C, D, E, F)$:
$A \to BC$, $C \to E$, $B \to D$
Is it in **BCNF**? Explain your answer. If it is not, normalize it into a set of relations in **BCNF**.

ANSWER:
It is not in BCNF.
The key is AF, so A → BC, C → E and B → D all violate BCNF.
$R(A, B, C, D, E, F) \Longrightarrow R1(A, B, C, D, F) and R2(C, E) using C \to E$
$R1(A, B, C, D, F) \Longrightarrow R3(A, B, C, F) and R4(B, D) using B \to D$
$R3(A, B, C, F) \Longrightarrow R5(A, F) and R6(A, B, C) using A \to BC$

The final BCNF tables are:
$R2(C, E)$
$R4(B, D)$
$R5(A, F)$
$R6(A, B, C)$

FD X -> Y leads to redundancy only if X does not contain a key.

Lossless-join decompositions:

decompositions R(X,Y,Z) -> R1(X,Y), R2(Y,Z) is lossless join if Y->X or Y-> Z shared attributes are the key of one of the decomposed tables

this condition can be checked using FDs

Trivial Functional Dependency

trivial FD: X -> Y is a trivial functional dependency when Y is a subset of X.

nontrivial FD: X -> Y when Y is not a subset of X

Completely non trivial FD: X -> Y where X ∩ Y = empty set

Cost of Join Algorithm

================ Join ================

Nested-Loop Join

- For each r ∈ R do
  For each s ∈ S do
  if r.C = s.C then output r,s pair
- Use smaller table for outer loop (R)
- Bulk block NLJ
  - $b_R + \lceil b_R/(M-2) \rceil \times b_S$

Hash Join

- Hashing stage (bucketizing)
  - Hash R tuples into G1,…,Gk buckets
  - Hash S tuples into H1,…,Hk buckets
- Join stage
  - For i = 1 to k do
  - match tuples in Gi, Hi buckets
- Number of buckets = M-1
- General cost ($b_R < b_S$)
  - $2(b_R + b_s)\left\lceil \log_{M-1}\left(\frac{b_R}{M-2}\right)\right\rceil + (b_R + b_S)$

Index Join

- Cost:
  - IO for R scanning
  - IO for index look up
  - IO for tuple read from S
- General cost:
  - $b_R + |R| \times (C + J)$
  - C average index lookup cost
  - J matching tuples in S for every R tuple

Sort-Merge Join

1. Sort stage: Sort R and S
2. Merge stage: Merge sorted R and S

- General cost:

$$2b_R\left(\left\lceil \log_{M-1}\left(\frac{b_R}{M}\right)\right\rceil + 1\right) + 2b_S\left(\left\lceil \log_{M-1}\left(\frac{b_S}{M}\right)\right\rceil + 1\right) + (b_R + b_S)$$

In general:

- Nested-loop join ok for "small" relations (relative to memory size)
- Hash join usually best for equi-join
  - if relations not sorted and no index
- Merge join for sorted relations
  - Sort merge join good for non-equi-join
- Consider index join if index exists

	Cost (M=22, b_R =100, b_S=1000)	Formula ($b_R < b_S$)
NLJ	5,100	$b_R + \left\lceil \frac{b_R}{M-2} \right\rceil b_S$
SMJ	7,500 (if unsorted) 1,100 (if sorted)	$2b_R\left(\left\lceil \log_{M-1}(\frac{b_R}{M})\right\rceil + 1\right)+$ $2b_S\left(\left\lceil \log_{M-1}(\frac{b_S}{M})\right\rceil + 1\right)+(b_R+b_S)$
HJ	3,300	$2(b_R+b_S)\left\lceil \log_{M-1}\frac{b_R}{M-2}\right\rceil + (b_R+b_S)$
IJ	1,115 – 10,640	$b_R + \|R\|(C + J)$ C: index lookup cost, J: # matching S tuples per R tuple

Disk

Access time = (average seek time) + (rotational delay) + (transfer time)

For 6000 RPM, **average rotational delay** is 5 ms because theres 60 seconds, 1 sec = 100 rotations, therefore 1 rotation = 10 ms, half of that is 5 ms.

Transfer time for a 6000 RPM 10,000 sectors/track is 10 ms / 10000 = 1 / 1000 ms

Transfer rate (bytes/sec) 6000 RPM, 10000 sectors/track, 1 KB/sector, what is transfer rate?

Burst transfer rate = (RPM / 60) * (sectors / track) * (bytes / sector)

Random IO is very expensive compared to sequential IO for MAGNETIC DISK, less expensive for SSD but still expensive.

Track is the whole circle

Index

Primary(clustering): can be dense*(record is created for every search key value) or sparse(only appears for some of the values, range of index columns are stored, deletion is slower and you may not know if you reach an actual search key)

and Secondary(non-clustering) Index

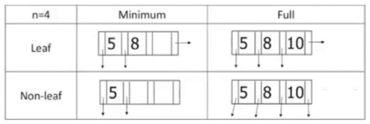
B+ Trees

B+ Tree nodes have at least:

Leaf (non-root) : ceiling((n+1)/2) pointers, ceiling((n+1)/2) – 1keys

Non-leaf nodes (non-root): ceiling((n/2)) pointers, ceiling(n/2) – 1 keys

Root node: 2 pointers, 1 key



Where does N come from?

N is determined by size of node, size of search key, size of an index pointer

Q: 1024 byte node, 10 byte search key, 8 byte pointer? What is N? 8(n) + 10(n-1) <= 1024

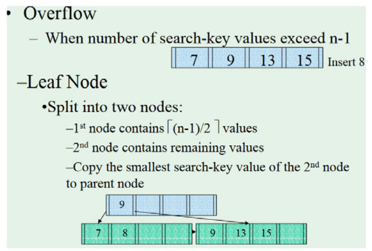
Deletion

for leaf node merging, we delete the mid key from the parent

for non-leaf nodes merging/redistribution, we pull down the mid key from their parent

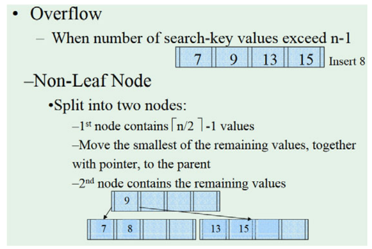
Leaf node overflow

The first key of the new node is copied to the parent


- Overflow
  – When number of search-key values exceed n-1
  – Leaf Node
    - Split into two nodes:
      – 1st node contains ⌈(n-1)/2⌉ values
      – 2nd node contains remaining values
      – Copy the smallest search-key value of the 2nd node to parent node

Non-leaf node overflow

The middle key is moved to the parent


- Overflow
  – When number of search-key values exceed n-1
  – Non-Leaf Node
    - Split into two nodes:
      – 1st node contains ⌈n/2⌉ -1 values
      – Move the smallest of the remaining values, together with pointer, to the parent
      – 2nd node contains the remaining values

Hash Index

No minimum space guarantee, also does not support range/inequality queries, only point/equality queries

Extendable hashing **insertion**

```
If no hbucket overflow
    insert the tuple into the hash bucket
if a hash bucket overflows:
    if the hash bucket I == directory I, then
        double directory size by copying existing pointers
        increase directory I value by 1
    split the overflowing hash bucket
        move tuples in the bucket to the new bucket based on their hash value
        update directory pointer
        increase the hash bucket I value by 1
```

Deletion/Merging

1. the mergeable buckets must have the same I value
2. first I - 1 bits should be the same
3. merge if satisfy condition 1 and 2

Transactions and ACID

- Atomicity: "all or nothing"
    - either all or none of the operations in a transaction is executed
    - if system crashes in the middle of a transaction, all changes are undone
- consistency
    - if the database was in a consistent state before transaction, it is still in a consistent state after the transaction
    - system will not corrupt database after a transaction
    - consistent - not corrupted state
- isolation
    - even if multiple transactions run concurrently, the result is the same as each transaction runs in isolation in a sequential order
- Durability
    - all changes made by committed transaction will remain even after system crash

SQL Isolation Levels

- strict isolation -> less concurrency
- lax isolation -> more concurrency

	Dirty Read	Non-repeatable Read(when you select rows and get different amount later)	Phantom
Read uncommitted	Y	Y	Y
read committed	N	Y	Y
repeatable read	N	N	Y
Serializable	N	N	N

Phantom – only for insertion and deletion

Dirt read – read before commit by a transaction

Non-repeatable read- when row retrieved twice and values within the row differ between reads

Remember: these isolation levels are for the current transaction, therefore for read uncommitted, the current transaction is allowed to read from another running transaction that has not yet committed.

MongoDB/NoSQL

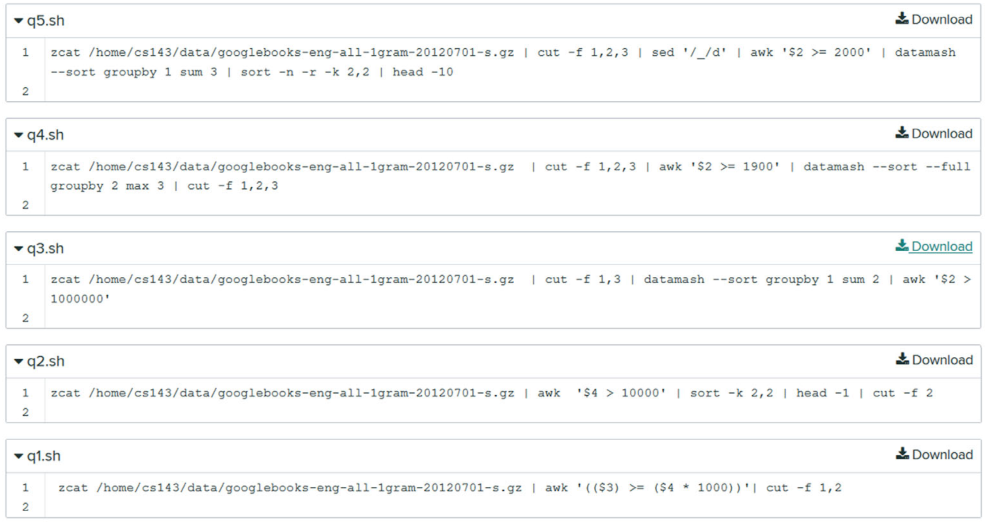
Unwind, match, group, project, sort

MapReduce

```python
from pyspark import SparkContext

from itertools import combinations
sc= SparkContext("local", "bookPairs")
lines = sc.textFile("/home/cs143/data/goodreads.user.books")
#lines = sc.textFile("/home/cs143/data/goodreads.1000")
#lines = sc.textFile("/home/cs143/data/goodreads.3000")
def function(line):
    #print(line)
    ID, books = line.split(":")
    books_individual = books.split(",")
    combos = combinations(books_individual,2)
    return combos
pair = lines.flatMap(lambda line: function(line))
pairs = pair.map(lambda word: (word, 1))
pairCounts = pairs.reduceByKey(lambda a, b: a+b)
#sortedpairCounts = pairCounts.sortBy(lambda a: -a[1])
#sortedpairCounts = sortedpairCounts.filter(lambda x: x[1] > 20)
sortedpairCounts = pairCounts.filter(lambda x: x[1] > 20)
sortedpairCounts.saveAsTextFile("output")
```

HW6



q5.sh

```
zcat /home/cs143/data/googlebooks-eng-all-1gram-20120701-s.gz | cut -f 1,2,3 | sed '/_/d' | awk '$2 >= 2000' | datamash
--sort groupby 1 sum 3 | sort -n -r -k 2,2 | head -10
```

q4.sh

```
zcat /home/cs143/data/googlebooks-eng-all-1gram-20120701-s.gz  | cut -f 1,2,3 | awk '$2 >= 1900' | datamash --sort --full
groupby 2 max 3 | cut -f 1,2,3
```

q3.sh

```
zcat /home/cs143/data/googlebooks-eng-all-1gram-20120701-s.gz  | cut -f 1,3 | datamash --sort groupby 1 sum 2 | awk '$2 >
1000000'
```

q2.sh

```
zcat /home/cs143/data/googlebooks-eng-all-1gram-20120701-s.gz | awk  '$4 > 10000' | sort -k 2,2 | head -1 | cut -f 2
```

q1.sh

```
 zcat /home/cs143/data/googlebooks-eng-all-1gram-20120701-s.gz | awk '(($3) >= ($4 * 1000))'| cut -f 1,2
```

Sample final

If no transactions modifies the table, you can run with the weakest isolation level, read uncommitted

If there is no insertion, we do not have to worry about phantoms, but we do need to worry about non repeatable reads (UPDATES)

Weakest isolation level needed for transaction if all other transactions were serializable is repeatable read if we do not perform any range queries.

Consider a B+ tree of order n

Minimum number of records that the tree can index, when it has 2 levels

2 * ceiling((n-1)/2)

Minimum k>=2 levels?

2 * (ceiling(n/2))^(k-2) * ceiling((n-1)/2)

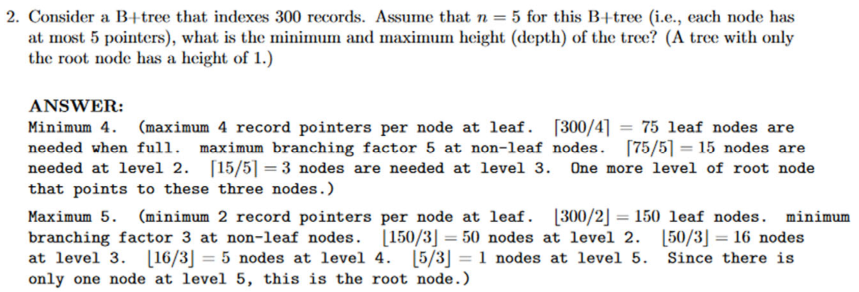
Maximum records that a tree can index, at 2 levels?

N * (n-1)

Max when k >= 2 levels

N^k * (n-1)

Suppose we are using n =99 with a table of 300,000, worst case, how many nodes do we index? 2 * 50 ^(k-2) * 49

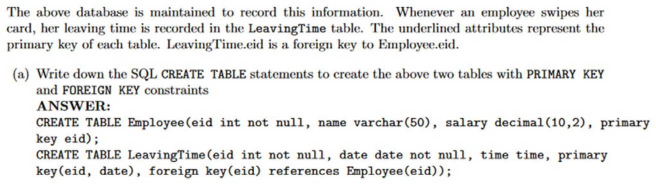


2. Consider a B+tree that indexes 300 records. Assume that $n = 5$ for this B+tree (i.e., each node has at most 5 pointers), what is the minimum and maximum height (depth) of the tree? (A tree with only the root node has a height of 1.)

ANSWER:
Minimum 4. (maximum 4 record pointers per node at leaf. $\lceil 300/4 \rceil = 75$ leaf nodes are needed when full. maximum branching factor 5 at non-leaf nodes. $\lceil 75/5 \rceil = 15$ nodes are needed at level 2. $\lceil 15/5 \rceil = 3$ nodes are needed at level 3. One more level of root node that points to these three nodes.)

Maximum 5. (minimum 2 record pointers per node at leaf. $\lfloor 300/2 \rfloor = 150$ leaf nodes. minimum branching factor 3 at non-leaf nodes. $\lfloor 150/3 \rfloor = 50$ nodes at level 2. $\lfloor 50/3 \rfloor = 16$ nodes at level 3. $\lfloor 16/3 \rfloor = 5$ nodes at level 4. $\lfloor 5/3 \rfloor = 1$ nodes at level 5. Since there is only one node at level 5, this is the root node.)

Integrity constraints

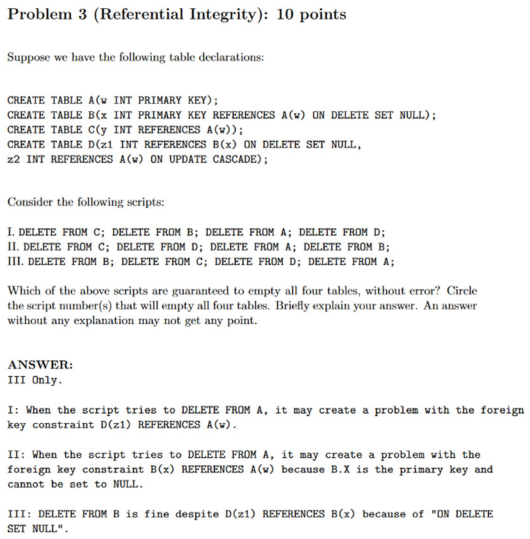


The above database is maintained to record this information. Whenever an employee swipes her card, her leaving time is recorded in the **LeavingTime** table. The underlined attributes represent the primary key of each table. LeavingTime.eid is a foreign key to Employee.eid.

(a) Write down the SQL **CREATE TABLE** statements to create the above two tables with **PRIMARY KEY** and **FOREIGN KEY** constraints
ANSWER:

```
CREATE TABLE Employee(eid int not null, name varchar(50), salary decimal(10,2), primary
key eid);
CREATE TABLE LeavingTime(eid int not null, date date not null, time time, primary
key(eid, date), foreign key(eid) references Employee(eid));
```

Referenced attributes must be PRIMARY KEY or UNIQUE



Problem 3 (Referential Integrity): 10 points

Suppose we have the following table declarations:

```
CREATE TABLE A(w INT PRIMARY KEY);
CREATE TABLE B(x INT PRIMARY KEY REFERENCES A(w) ON DELETE SET NULL);
CREATE TABLE C(y INT REFERENCES A(w));
CREATE TABLE D(z1 INT REFERENCES B(x) ON DELETE SET NULL,
z2 INT REFERENCES A(w) ON UPDATE CASCADE);
```

Consider the following scripts:

```
I. DELETE FROM C; DELETE FROM B; DELETE FROM A; DELETE FROM D;
II. DELETE FROM C; DELETE FROM D; DELETE FROM A; DELETE FROM B;
III. DELETE FROM B; DELETE FROM C; DELETE FROM D; DELETE FROM A;
```

Which of the above scripts are guaranteed to empty all four tables, without error? Circle the script number(s) that will empty all four tables. Briefly explain your answer. An answer without any explanation may not get any point.

ANSWER:
III Only.

I: When the script tries to DELETE FROM A, it may create a problem with the foreign key constraint D(z1) REFERENCES A(w).

II: When the script tries to DELETE FROM A, it may create a problem with the foreign key constraint B(x) REFERENCES A(w) because B.X is the primary key and cannot be set to NULL.

III: DELETE FROM B is fine despite D(z1) REFERENCES B(x) because of "ON DELETE SET NULL".

Suppose you have 2 relations, $R(\underline{A}, B)$ and $S(\underline{B}, C)$, with the following characteristics:

- The size of one disk block is 1000 bytes.
- Attributes A, B are of length 10 bytes. Attribute C is of length 180 bytes.
- The tuples are not spanned across disk blocks
- $|R| = 5,000$ (number of tuples of R)
- $|S| = 500$ (number of tuples of S)
- We have 30 blocks of memory buffer
- We use one disk block for one B+tree node
- Each pointer in a B+tree index (both a record pointer and a node pointer) uses 10 bytes.

1. (2 points) What is the minimum number of blocks needed to store R and S?

   R: ________________ S: ________________

   ANSWER:
   `R: 100, S: 100.`

2. (2 points) We want to construct a dense B+tree on attribute B of table S by scanning the S table. What is the maximum possible n for the B+tree given the above parameters?

   n: ________________

   ANSWER:
   $10n + 10(n-1) \leq 1000$. Therefore, $n = 50$.

3. Assume the numbers computed in the previous problems. Assume that each node in the B+tree contains the minimum number of keys and pointers (as long as it is allowed in our parameter setting).

   (a) (4 points) How many nodes does the constructed B+tree have?

   ____

   ANSWER:
   21. For leaf nodes, the minimum number of keys per node is 25. Therefore, we will have $500/25 = 20$ leaf nodes and one root node. Any answer with 500/(half of answer 2)+1 would be considered correct assuming that answer 2 is not too small.

   (b) (4 points) How many disk IOs would be incurred during the construction of the B+tree on `S.B`? Assume that you use the main memory buffer in the most efficient way to minimize the number of disk IOs. *In your answer, please include the cost of reading the tuples from S and writing the constructed B+tree to the disk.*

   ____

   ANSWER:
   121. For index construction, we need to scan the entire S table once. S is stored in 100 blocks, so 100 disk IOs for reading S. We also incur 21 disk IOs to write the B+tree. Any answer with S from answer 1 + answer 3(a) would be considered correct assuming that 3(a) is not too big.

4. (4 points) We want to execute the query "`SELECT R.A,R.B FROM R,S WHERE R.B=S.B`" using the B+tree constructed in the previous problems. Since we do not need to return `S.C`, we decide to use the following algorithm to avoid scanning the S table:

```
For each tuple r in R
  Lookup S.B index with r.B
  If r.B exists in the index, return (R.A, R.B)
```

   Compute the cost of this join. In computing your answer, please use the main memory buffers in the most efficient way to minimize the number of disk IOs. Do not include the cost for writing the final answer. Assume that none of the index nodes and the R blocks are cached in main memory in the beginning.

   ____

   ANSWER:
   121. We cache all B+tree node in the main memory, and scan the R table once. R is stored in 100 blocks. The B+tree index is 21 blocks. Any answer with answer 1S + 3(a) would be considered correct assuming that 3(a) is not too big.

5. (4 points) How many disk IOs would have been incurred if we used block nested loop join? Is it worthwhile to construct an index on `S.B` to perform the join using the above algorithm considering the index construction overhead?

   ____

   ANSWER:
   block nested loop join would have incurred $100 + 4 \times 100 = 500$ disk IOs. Even if we take the index construction overhead, the total number of disk IOs was 242. It was definitely worthwhile. Any answer with (1R + 4x1S) or (1R + 100x1S) would be considered correct.

For all questions, assume the relation $R(\underline{A}, B, C)$ and the relation $S(\underline{C}, \underline{D}, E)$.

1. (10 points) Suppose relation R occupies n blocks and relation S occupies m blocks. We assume that $n \leq m$. Initially, both relations are on disk and we want to perform a block-nested-loop join. If we want to read each block of R and S only once during the join, what is the minimum number of main-memory buffers (each the size of a disk block) required, including the buffer(s) for output?

   ANSWER:
   $n+2$. n to load the entire R, 1 for reading S, 1 for final output

2. (10 points) Assume that the table R has 100,000 tuples stored in 10,000 blocks. We construct a dense unclustered B+tree on the attribute A of the table R. Assume that the B+tree has 1,000 nodes at the leaf level, 10 nodes at the non-leaf level directly above leaf and 1 node at the root level. Each leaf node of the B+tree contains 100 key values. The values of $R.A$ are uniformly distributed between 1 and 100,000. Now consider the following query:

```
SELECT * from R WHERE A > 99000
```

   We use the B+tree index to identify the tuples that satisfy the condition $A > 99000$ and retrieve the tuples. How many disk IOs do we need to process this query? Assume that we have 10 main memory buffers to process this query.

   ANSWER:
   1,012 block. Traversing the B+ tree requires 3 node accesses (1 root, 1 non-leaf, and 1 leaf). 1% of the keys will satisfy the condition, so the matching keys will be contained in 10 leaf nodes at the right end of B+tree (1% of the 1,000 leaf nodes). Therefore, in order to retrieve these keys in the leaf nodes, we will have to follow 9 more succeeding leaf nodes.

   For each matching tuple (a total of 1,000), we then access one block to fetch the actual record, so the total disk IO is $3+9+1,000 = 1,012$ blocks.

   Is using the B+tree better than scanning the entire table of R to process the query? Ignore potential performance difference due to random IO vs sequential IO.

   ANSWER:
   Yes. Scanning R requires 10,000 disk IOs.

3. Now, assume that we are sorting the table R using the merge-sort algorithm. Again, R is stored in 10,000 blocks, and we have 10 main memory buffers. We just finished the first sorting stage of the merge-sort algorithm and generated multiple sorted runs.

   (a) (5 points) How many sorted runs do we have now?

   ANSWER:
   1,000 sorted runs

   (b) (5 points) We started the second "merging stage," where we merge sorted runs. So far, you have read a total of 5 disk blocks during this stage. What is the minimum and the maximum number of disk blocks may have been written to the disk in this stage? Count the disk writes during the merging stage only. Assume that we immediately flush an output buffer when it is full. Again, each block has 10 tuples and we have 10 main memory buffer blocks.

   ANSWER:
   min: 0, max: 0. When we have 10 main memory buffers, we can merge 9 sorted runs in each stage. In order to start ''merging,'' and producing an output, we need to read at least one block from each sorted run. Since we haven't read 9 blocks yet, we cannot start producing output.

   (c) (10 points) What if you have read a total of 15 disk blocks during the merging stage now (not 5 blocks as we previously assumed)? What is the minimum and the maximum number of disk blocks that you may have written to the disk so far?

   ANSWER:
   min: 6, max: 14. In one extreme case, all main memory buffers will be full except the output buffer (because when the output buffer is full, it is immediately flushed). Therefore we can hold slightly more than 9 blocks in main memory in this case, which means that we must have written 15-9=6 blocks. In the opposite extreme case, each buffer will have only one tuple, so we hold 10 tuples (= 1 block) in main memory buffers. So we have written 14 blocks to the disk.

The first plan requires an additional $(750,000 \cdot 5)$ random I/Os.

The new plan requires an additional $(750,000 \cdot \frac{5000}{100})$ sequential I/Os.

Thus the new plan requires approximately 10 times more I/O (5 vs. 50 blocks of S per tuple in R). If the cost of random access is more than 10 times the cost of sequential access, then the second plan will require less time to complete.

2. Consider two relations $R(A, B)$ and $S(A, C)$. Suppose relation R occupies 1,000 blocks and relation S occupies 100 blocks. When we performed $R \bowtie S$ using the hash join algorithm that we learned in the class, the algorithm incurred (approximately) 3,300 disk block I/Os (either read or write), excluding the I/Os for writing the final join result. Write the minimum number of main-memory buffers (each of the size of a disk block) that we must have had when we performed the join.

   ANSWER:
   12. The total cost of the join is 3 times the sizes of the input relations, indicating that the hash-join algorithm went through just one bucketizing step and one join step. This means that each bucket of the smaller table should fit in main memory during the join step (with two extra memory blocks left for reading the other table and for writing the output). Assuming we have M memory blocks, the size of each bucket of the smaller table S after one bucketization step is $\lceil \frac{100}{M-1} \rceil$. This number should be smaller than or equal to $M-2$. That is, That is, $\lceil \frac{100}{M-1} \rceil \leq M-2$. The minimum M that satisfies this inequality is 12.

1. Consider the relation $R(A, B, C)$ with the following properties:

   - R contains 8,000 tuples
   - The size of a disk block is 4,000 bytes
   - The size of each R tuple is 40 bytes
   - Each disk block holding R tuples is full

   (a) How many blocks are required to store the relation R?

   ANSWER:
   Each disk block holds $4000/40 = 100$ R tuples, so R is stored in $8000/100 = 80$ blocks.

   (b) We would like to sort the tuples of R on attribute A using the sort-merge algorithm in two passes. That is, after the initial sorting pass, we want to generate the completely sorted relation by one more merging pass. What is the minimum number of main memory blocks required?

   ANSWER:
   10. In order to finish sort-merge algorithm in two passes, the number of sorted runs created from the first sorting pass should be at most one less than the available memory blocks. Assuming that we have M memory blocks, the number of sorted runs created from the first sorting pass is $\lceil 80/M \rceil$. This should be less than or equal to $M-1$. That is, $\lceil 80/M \rceil \leq M-1$. The minimum such M is 10.