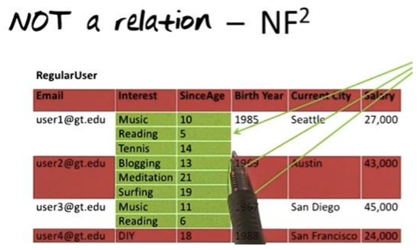
Log_base 2(N) = log_base10(N) / log_base10(2)

Log)base10(2) = 0.30102999566

Normalize a relation:

1) No redundancy of facts
2) No cluttering of facts
3) Preserve information
4) Preserve functional dependencies

Not a relation - Multi value



NOT a relation – NF²

RegularUser

Email	Interest	SinceAge	Birth Year	Current City	Salary
user1@gt.edu	Music	10	1985	Seattle	27,000
	Reading	5			
	Tennis	14			
user2@gt.edu	Blogging	13	1969	Austin	43,000
	Meditation	21			
	Surfing	19			
user3@gt.edu	Music	11	1967	San Diego	45,000
	Reading	6			
user4@gt.edu	DIY	18	1988	San Francisco	24,000

To make this a relation, avoid multi values…repeat uer1 for each interest/sinceAge relation

Problems:

1) Redundancy – leads to inconsistency.
2) Insertions anomalies –
3) Deletion anomalies – delete one fact inadvertently deletes other fact for another user.
4) Update anomalies – redundant information updated

Information loss:

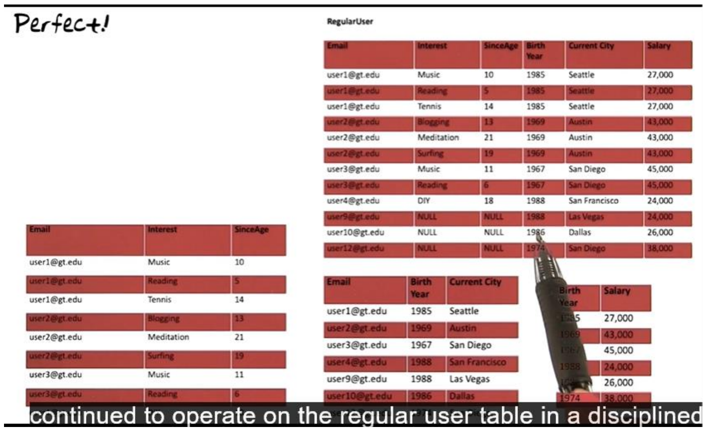
Decompose the regularUser table into two tables.

Table1: Email, Interest, SinceAge, Birth Yr, Current city. Table2: Current City, Salary

Information Loss: Joining the two tables causes more information (tuples) returned that are not accurate. Additional rows do not reflect reality.

Dependency Loss: cannot enforce the functional dependencies that are split between the two relations. Birth year and salary do not coexist in the table and cannot be enforced.

Solution: decompose into three tables



Perfect!

RegularUser

Email	Interest	SinceAge	Birth Year	Current City	Salary
user1@gt.edu	Music	10	1985	Seattle	27,000
user1@gt.edu	Reading	5	1985	Seattle	27,000
user1@gt.edu	Tennis	14	1985	Seattle	27,000
user2@gt.edu	Blogging	13	1969	Austin	43,000
user2@gt.edu	Meditation	21	1969	Austin	43,000
user2@gt.edu	Surfing	19	1969	Austin	43,000
user3@gt.edu	Music	11	1967	San Diego	45,000
user3@gt.edu	Reading	6	1967	San Diego	45,000
user4@gt.edu	DIY	18	1988	San Francisco	24,000
user9@gt.edu	NULL	NULL	1988	Las Vegas	24,000
user10@gt.edu	NULL	NULL	1986	Dallas	26,000
user12@gt.edu	NULL	NULL	1974	San Diego	38,000

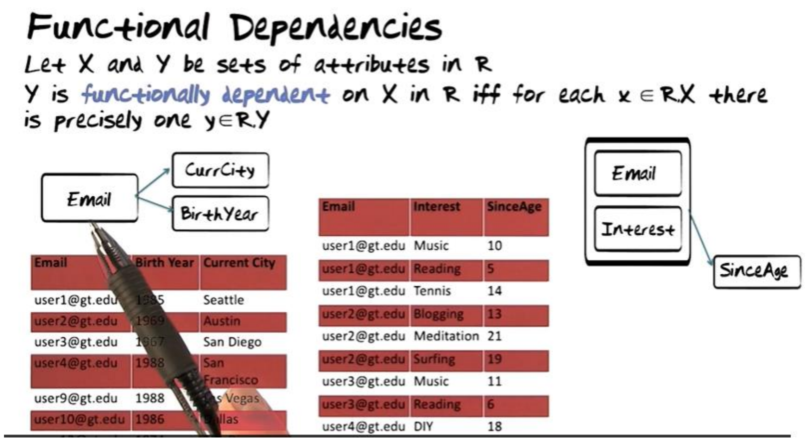
Email	Interest	SinceAge
user1@gt.edu	Music	10
user1@gt.edu	Reading	5
user1@gt.edu	Tennis	14
user2@gt.edu	Blogging	13
user2@gt.edu	Meditation	21
user2@gt.edu	Surfing	19
user3@gt.edu	Music	11
user3@gt.edu	Reading	6

Email	Birth Year	Current City
user1@gt.edu	1985	Seattle
user2@gt.edu	1969	Austin
user3@gt.edu	1967	San Diego
user4@gt.edu	1988	San Francisco
user9@gt.edu	1988	Las Vegas
user10@gt.edu	1986	Dallas

Birth Year	Salary
1985	27,000
1969	43,000
1967	45,000
1988	24,000
[illegible]	26,000
1974	38,000

continued to operate on the regular user table in a disciplined

How to decompose in the proper manner? --- Functional Dependencies



Functional Dependencies

Let X and Y be sets of attributes in R
Y is functionally dependent on X in R iff for each $x \in R.X$ there is precisely one $y \in R.Y$

Email	Birth Year	Current City
user1@gt.edu	1985	Seattle
user2@gt.edu	1969	Austin
user3@gt.edu	1967	San Diego
user4@gt.edu	1988	San Francisco
user9@gt.edu	1988	Las Vegas
user10@gt.edu	1986	Dallas

Email	Interest	SinceAge
user1@gt.edu	Music	10
user1@gt.edu	Reading	5
user1@gt.edu	Tennis	14
user2@gt.edu	Blogging	13
user2@gt.edu	Meditation	21
user2@gt.edu	Surfing	19
user3@gt.edu	Music	11
user3@gt.edu	Reading	6
user4@gt.edu	DIY	18

1) For each email, there is one currCity and BirthYr.
2) Email and interest determine SinceAge.

Full functional Dependency: Since Age is Fully Functional Dependent on Interest.

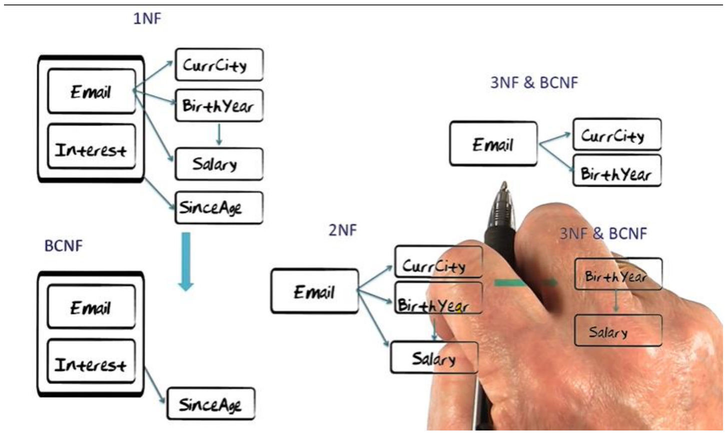
How to enforce? Use keys to enforce full functional dependencies. (Email and Interest are made to be keys and Email is the key to CurrCity and BirthYear)

- In a relation, the value of the key is unique and hence enforces a function

Overview of Normal Forms

1) The whole set of data structures is called non-first normal form: NF^2(NF squared)
2) 1NF: R is in 1NF iff (if and only if) all domain values are atomic
3) 2NF R is in 2NF iff R is in 1NF and every nonkey attribute is fully dependent on the key
4) 3NF: R is in 3NF iff R is 2NF and every nonkey attribute is non-transitively dependent on the key
5) BCNF (Boyce-Codd Noraml Form) R is in BNCF iff every determinant is a candidate key
6) Determinant: a set of attributes on which some other attribute is fully functionally dependent.

All attributes must depend on the key(1NF), the whole key(2NF), and nothing but the key (3NF), so help me Codd (BCNF)"



- 1NF: Email and Interest together determine sinceAge. Email alone determines CurrCity, BirthYR and Salary and salary is determined by birthYr.
- 2NF: CurrCity, BirthYr and Salary all depend on the key EMAIL. But there are transitive dependencies: Email determined birthyear which determines salary.
- 3NF & BNCF: Email is the key to enforce CurrCity and BirthYr
- 3NF & BNCF: BirthYr key determined salary
- BNCF: Email and Interest are candidate key and SinceAge is determined by them.

Armstrong's Rule

1) Reflexivity: If Y is a part of X, then X determins Y. If the right side is a subset of the left side, then the left side determines the right side.
2) Augmentation: If x determines Y, then augment left side, right side, or both sides. If email determines birthyear, then Email, Interest determines BirthYear, Interest
3) Transitivity: If x determines y and Y determines Z, then X determines Z.
    a. Email -> BirthYear and BirthYEar -> Salary, then Email → Salary

Guarantee lossless joins/lossless decomposition– The join field must be a key is at least one relation. No way of creating duplicates using the key to join

Guarantee preservation of functional dependencies – meaning implied by remaining functional dependencies must be the same as the meaning implied in the original set.

PatientVisit(doctorID, PatientID, Date, Diagnosis, PatientName, procedure, charge, BillingCode)

F:

{Patient id – PatientName

DoctorID, PatientID, Date – Diagnosis

DoctorID, PatientID, Date – Procedure

Procedure – charge

Procedure – billingCode}

KEY: DoctorID, PatientID, Data

Transitivity Rule = DoctorID, PatientID, Date → Charge

TRUE	FALSE
Two rows with same values for Dignosis and different value for procedure	
F is a minimal cover	

Non-prime attributes: PatientName, Procedure, Diagnosis, Charge

2NF Violation: PatientID -> PatientName

3NF Violation: PatientId -> PatientName; Procedure -> Charge, BillingCode

Highest Normal Form of schema F = 1NF

PV1(DoctorID, PatientID, Date, Diagnosis, Procedure, Charge, BillingCode)
PV2(PatientID, PatientName)
	• Only pv2 is 3NF
	• PV1 and 2 satify 2nd normal form

Is ^^^ lossless join decomp?	• Yes

Pv1(DoctorID, PatientID, Date, Diagnosis, Procedure)
PV2(PatientID, PatientName)
PV3(Procedure, Charge, BillingCode
	• All satisfy 3rd normal form
	• All satisfy BNCF

Fully functional dependent on (DoctorID, PatientID, Date) = Procedure, Diagnosis, BillingCode, Charge

Set of attributes represent the closure of the set (PatientID, Procedure) = PatientID, Procedure, PatientName, Charge, BillingCode

Based on F:

- PatientID, Diagnosis -> PatientName
- PatientID, Procedure -> BIllingCode
- PatientID, Procedure -> Charge

DoctorID	PatientID	Date	Diagnosis	PatientName	Procedure	Charge
1001	11	2011-03-25	Hypertension	Brown	20	400
1003	10	2011-03-25	High-ldl	Green	20	400

1001	11	2011-04-10	Obese	Brown	40	300
1003	10	2011-04-10 ---- 2011-04-12	Hyperthyroid	Green -------- Green	40	300 ------- 300

R(A,B,C,D,E,G) – which functional dependency has extra attributes on left {ABC->D, B->E, C->G,…..}

Computer Architecture

Main memory – RAM – volatile, fast, small, and expensive

Secondary Memory – DISH – permanent, slow and cheap

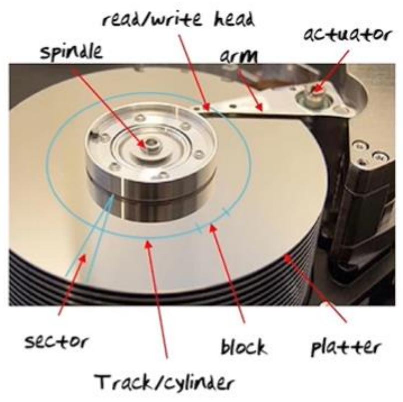
- Applications run by the cpu can only query and update data in main memory
- Data must be written back to secondary memory after updating
- Only a tiny fraction of a REAL database fits in main memory

Main memory access time = 3x10 ^ -7s econds

Disk access time = 1X10 ^ -2 seconds

Disk:

- 4 platters
- 8 heads
- ~150K tracks per surface
- ~1000KB/track
- 1200GB
- 512 byte/sector
- 4, 8, or 14KB/block
- 600MBs transfer rate
- 10,000 RPMs
- 304ms latency



Records, blocks, files:

RegularUser(Email varchar(50), Sex char(1), Birthdate datetime, currentCity varchar(50), hometown varchar(50) == 159 bytes (50 + 1 + 8 + 50 + 50)

Block size = 4K: assuming we only fill 80% will give 3,200 bytes. Records/block = ~20

- Spanned: Break up part of a block at the end and place on the next block is spanned representation
- Unspanned: ignore the waste at the end of block. Avoid breaking up records.

Files: blocks linked by pointers.

- Block pointer – 4 bytes
- # records = assume 4 million 'user' records and 20 records on a block is ~200,000 blocks
- Block size of 4K means file size is about 800 MBs

Compute time to transport from DISK to RAM back to DISK

Assumptions:

- Page Fault
  - Seek time: 3-8ms
  - Rotational delay: 2-3ms
  - Transfer Time: .5-1ms
  - Total = 5-12ms == 0.01sec /10 milliseconds
- Once we add a location to the disk, pickup 250 blocks
- BULK transfer
  - Save seek time and rotational delay
  - Incur transfer time only
  - Page fault of 10ms costs 2.5 seconds to transfer 250 blocks
  - By bulk transfer, cost is .260 seconds

Buffer Management:

- LRU – Least recently used strategy – replace the least recently used memory.
  - Excellent for merge joins, kills nested loops
- MRU – most recently used strategy – works well for nested loops

Heap Unsorted – files sit on a bunch of blocks with no sorting

- Block pointer: 4 bytes
- # records: 4mil
- # data blocks: 200,000

- File size ~800MB
- Lookup time:
  - N/2 where N = data blocks
  - 200,000/2 * 0.01s = 16.6min access time

Binary Search – sorted data. Split in half, not in that half, split in half again. Not there, split in half again.

- Sorted File
  - Block pointer: 4 bytes
  - # records: 4mil
  - # data blocks: 200,000
  - File size ~800MB
  - Lookup time:
    - Log_base2(n) for binary search – half the search, split left/right, half the search
      - 18*.01sec = access time .18sec

Primary index – reduce the size of where we search. Pick up key value of block and store key values.

- Block pointer: 4 bytes
- Filled ~80%
- Fanout ~60
- Records: 4mill
- Data blocks = 200,000
- Index blocks(sparse): 3334
- Index bocks(dense): 66,666
- File size: ~800MB

200,000 blocks of data and a block pointer is 4bytes. Filled 80%, we have 3,200 bytes to work with. Key value Email is varchar 50, plus pointer, is 54 bytes. 54 divided up into 3,200 is ~60. So each primary index will have ~60 key values. (fanout).

200,000/60 = number of index blocks(sprse)

Lookup time:

- Log_base2(N)+1 where N = # index bocks
- Sparse: log2(200,000/60)+1 * 0.01 = (12+1) * 0.01 = 0.13sec
- Dense: log2(4,000,000/60)+1 * 0.01s = (16+1) * 0.01s = 0.17s

Single level index(secondary index): building index on something other than key values and not sorted

- Block pointer: 4 bytes
- Filled ~80%
- Fanout ~60
- Records: 4mill

- Data blocks = 200,000
- Index blocks(sparse): 3334
- Index bocks(dense): 66,666
- File size: ~800MB
- Lookup time:
  - Log_base2(n) + 1
  - Dense: log2(4,000,000/60)+1 * 0.01s = (16+1) * 0.01s = 0.17s

Multi-level index: building an index on the index

- Block pointer: 4 bytes
- Filled ~80%
- Fanout ~60
- Records: 4mill
- Data blocks = 200,000
- Index blocks(sparse): 3334
- Index bocks(dense): 66,666
- File size: ~800MB
- Lookup time:
  - Log_base(fanout) (n) + 1
  - Sparse: log60(3334+1) * 0.01s = 2+1*0.01s = 0.03s

Multi level index: most popular is B+ Tree

- All data is at the bottom, and all above is index nodes above it.
- Insertion, deletion, and update operations are implemented to keep the tree balanced

Static Hashing: Bucket directory. Maps key to a bucket

- Much larger than address space
- Distribute values uniformly over the address space
- Fill buckets as much as possible
- Avoid collisions
- Lookup time:
  - Constant: (1-2)*0.01s = 0.01-0.02S if bucket directory is in main memory
  - Dynamic hashing expands bucket directory when needed.

Assume the COURSES file contains 132,000,000 records and is organized as a Hash File with the primary key as the hash key. We can assume that our Hash function uniformly distributes the records in the file and that there is no overflow for any bucket. We can also assume that each bucket is only

80% full. How many buckets would be needed to store the unspanned records if each bucket consists of 5 consecutive blocks?

- Steps to solve

1. Calculate the amount of records per block unspanned

   - int((8000 bytes/block) / (120 Bytes/Record)) = 66 records/block

2. Calculate the amount of blocks per bucket considering how full the buckets are

   (5 block/bucket) * (80% full) = 4 full blocks/bucket

3. Calculate the total amount of blocks needed

   - (132,000,000 records/file) / (66 records/block) = 2000000 blocks/file

4. Calculate the amount of buckets needed for the file

   (2000000 blocks/file) / (4 full blocks/bucket) = 500,000 buckets/file

How much time would it take, on the average, to read one track worth of data from disk to main memory (RAM) at one time?

   One track is 2000000/8000 = 250 blocks

   One block can be read in 0.1 ms , so 250 x 0.1 = 25 ms

Then it is necessary to place the reading heard at the start the track which will take 4 + 2 = 6 ms

   So total time is 31 ms

Consider a disk with 4 platter (8 recording surfaces), 100000 cylinders, a track size of 40,000 bytes and a block size of 4000 bytes. Also assume that block contains exactly 20 records. At most, how many records can be strored on this disk?

- 16

Spanned records are always needed when the size of a record is larger that the size of a block

----

Unordered Heap File: Searching for a particular record in the file, on average, requires reading half of the file

----

Disk with the following:

Avg seek time = 4ms

Avg rotational delay = 2ms

Transfer time per block = 0.05ms

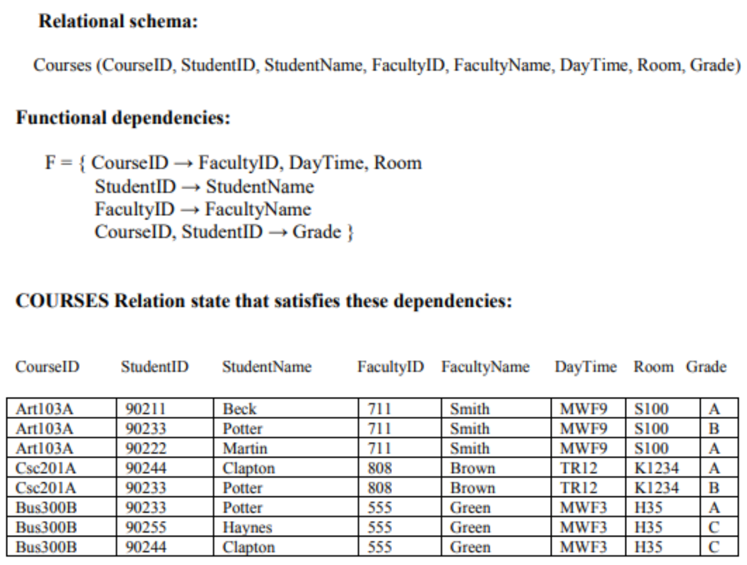
Consider a file containing 2048 blocks stored as an ordered file of this disk. On avg, how much time (in ms) would it take to find a particular record with a given key value in the file using binary search?

- 66.55

Which of the follow SQL queries can be answered entirely from a dense secondary index on the nonkey field Age for the table Employee(SSN, Name, City, Salary, Age, Dnum)

That is, the employee data field does not have to be accesses, just the index.

- Select Distinct Age from Employee
- Select MAX(Age) from Employee



Relational schema:

Courses (CourseID, StudentID, StudentName, FacultyID, FacultyName, DayTime, Room, Grade)

Functional dependencies:

F = { CourseID → FacultyID, DayTime, Room
StudentID → StudentName
FacultyID → FacultyName
CourseID, StudentID → Grade }

COURSES Relation state that satisfies these dependencies:

CourseID	StudentID	StudentName	FacultyID	FacultyName	DayTime	Room	Grade
Art103A	90211	Beck	711	Smith	MWF9	S100	A
Art103A	90233	Potter	711	Smith	MWF9	S100	B
Art103A	90222	Martin	711	Smith	MWF9	S100	A
Csc201A	90244	Clapton	808	Brown	TR12	K1234	A
Csc201A	90233	Potter	808	Brown	TR12	K1234	B
Bus300B	90233	Potter	555	Green	MWF3	H35	A
Bus300B	90255	Haynes	555	Green	MWF3	H35	C
Bus300B	90244	Clapton	555	Green	MWF3	H35	C

Single functional dependency: CourseID, StudentID -> Grade

- If any two tuples in Course have the same COurseID, StudentID value pair then those tuples must have the same grade value
- A given combination of values of CourseID and StundentID implies a unique grade value

Which of the following functional dependences can be derived from F using the REFLEXIVE inference rule

- StudentID, StudentName -> StudentID

- CourseID -> coursed
- CourseID, StudentID -> StudentID

Derived from F using transitive inference rule

- CourseID -> Faculty Name

StudentID -> StudentName violates 2nd normal form

- True

Key for courses with respect to F

- CourseID, StudentID

Allowable to have two tuples in Courses relation with different values for CourseID but the same value for DayTime

- Yes

Desirable properties of a relational decomposition

- Non0additive (lossless) join
- Dependency preservation
- Attribute preservation

Attributes that are fully fuctional dependent on the combo of attributes (CourseID, StudentID)

- 1

Would inserting [Csc201A | 90266 | Martin | 808 | White | TR12 | K1234 | A] violate any functional dependences

- Yes

Highest normal from that the Courses schema satifies

- 1st normal form

Decompose Course schema into:
Course1(CourseID, Student ID, Grade)
Course2(CourseID, FacultyID, FacultyName, DayTime, Room)
Course3(StudentID, StudentName)
Whats True

- The highest normal form for Courses2 is 2nd normal form
- The highest notal form for courses3 is BCNF

What is the closure of the set of attributes (CourseID, Grade) with respect to F

- (CourseID, Grade, FacultyID, DayTime, Room, FacultyName)

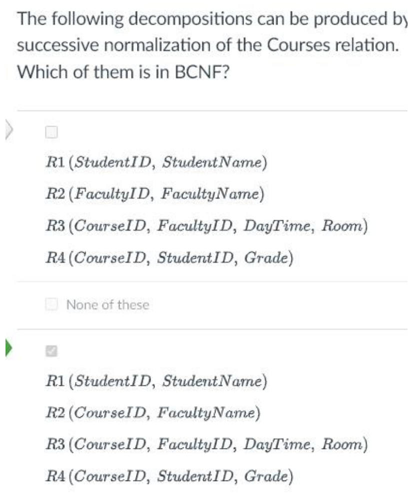
Decomposition into

Course1(CourseID, StudentID, Grade)

Course2(CourseID, StudentID, StudentName, FacultyID, FacultyName, DayTime, Room)

Is lossless?

- True



The following decompositions can be produced by successive normalization of the Courses relation. Which of them is in BCNF?

☐
$R1\,(StudentID,\ StudentName)$
$R2\,(FacultyID,\ FacultyName)$
$R3\,(CourseID,\ FacultyID,\ DayTime,\ Room)$
$R4\,(CourseID,\ StudentID,\ Grade)$

☐ None of these

☑
$R1\,(StudentID,\ StudentName)$
$R2\,(CourseID,\ FacultyName)$
$R3\,(CourseID,\ FacultyID,\ DayTime,\ Room)$
$R4\,(CourseID,\ StudentID,\ Grade)$

- Both True

Which are true

- One of the reasons for normalizing a relation is to reduce redundancy
- Repeated use of the inference rules on a superkey generates all the attributes of the relation

Highest normal form of R(A, B, C, D)

A->B, C, D ; A,B -> C,D ; D : A,B,C -> D

- BCNF

Relation R(A,B,C) with function dependencies A->B; B->C; C->A can be decomposed into a set of BCNF relations without information loss

- True

Savings when the DBMS read a number of consecutive blocks from the same track as one I/O operation as opposed to reading the same number of blocks randomly as multiple I/O operations

- There will be a savings in eliminating the seek time and rotational latency cost for all but the first block

SQL Insert statement, without a WHERE clause, is issues. An _____ file organization would likely be best to reduce processing time

- Unordered (Heap)

Each record = 120 bytes
We have a DISK with:
5 platters (10 recording surfaces)
8,000 cylinders
track size = 2,000,000 bytes
block size = 8000 bytes
average seek time = 4 ms
average rotational delay = 2 ms
transfer time per block = 0.1 ms

1) Blocking factor = block size/size of each record (round down)
   a. 8000/120 = 66
2) How many unspanned records can be stored in one track of this file?
   a. Each block can accommodate 66 unspanned records
   b. Track contains 2,000,000/8000 = 250 blocks
   c. 66*250 = 16,500
3) Assume the COURSES file contains 132,000,000 records and is organized as a Hash File with the primary key as the hash key. We can assume that our Hash function uniformly distributes the records in the file and that there is no overflow for any bucket. We can also assume that each bucket is only 80% full. How many buckets would be needed to store the unspanned records if each bucket consists of 5 consecutive blocks?
   a. Calculate the amount of records per block unspanned
      - int((8000 bytes/block) / (120 Bytes/Record)) = 66 records/block
      2. Calculate the amount of blocks per bucket considering how full the buckets are
      - (5 block/bucket) * (80% full) = 4 full blocks/bucket
      3. Calculate the total amount of blocks needed
      - (132,000,000 records/file) / (66 records/block) = 2000000 blocks/file
      4. Calculate the amount of buckets needed for the file
      - (2000000 blocks/file) / (4 full blocks/bucket) = 500,000 buckets/file

1. Is it possible to have a relational schema which is in 3normal form but NOT in 2normal form

* No

2. If you have a relation R(A,B) with Functional dependency {A→B;} how can you enforce this functional dependency?

* Make A the key of R

3. Can we tell that the join of the two relations R1(A,B) and R2(A,B,C,D) is lossless, even if we do not know the set of functional dependencies

* Yes

4. For the relational schema R(A,B,C,D,E,G) which of the following functional dependencies contain an extraneous (extra/unneeded) attribute on the LHS (left hand side): FD = { ABC→ D; B → E; C→ G; EG→ D; E → A }'

* A: ABC→ D

5. Given the set of FD's as follows: {ABC → D; B →E; C →G; EG →D; E→A} which of the following functional dependency can be generated by the reflexive inference rule?

* EG → G

6. Given the set of FD's as follows: {ABC → D; B →E; C →G; EG →D; E→A} which of the following functional dependency can be generated by the transitive inference rule

* B→ A

7. Given the set of FD's as follows: {ABC → D; B →E; C →G; EG →D; E→A} which of the following functional dependency can be generated by the augmentation inference rule?

* EG → AG

8. Given the following sets of functional dependencies:F1 = { A → BC; E → CD; AD→ E}F2 = { A → B; A→ C; E→ C; E→ D; AD→ E }are F1 and F2 equivalent?

A: Yes

9. Given the relation R(A,B,C,D) with FD's {A→B; A→C; C→D} what is the highest normal form this relation is in?

* C: 2NF

10. Given the relation R(A,B,C,D) with FD's {A→B; A→C; } what is the highest normal form this relation is in

* B: 1NF

Given the relation R(A,B,C,D) with FD's {AB → C; AB→D} what is the highest normal form this relation is in?

- 3NF

Given the relation R(A,B,C,D) with FD's {A→B; A→C; C→D} what is the closure of attribute {A} with respect to the functional dependencies?

- C: {A,B,C,D}

13. Consider the following relational schema R(A,B,C,D) and set of functional dependenciesFD = { A→ B; C→D; BC →A }Note, there are two candidate keys for R, they are AC and BCSuppose we decompose relation R into two relations R1(C,D) and R2(B,C,A).Is this a lossless join decomposition?

* A: Yes

14. Consider the following relational schema R(A,B,C,D) and set of functional dependenciesFD = { A→ B; C→D; BC →A }Note, there are two candidate keys for R, they are AC and BCSuppose we decompose relation R into two relations R1(C,D) and R2(B,C,A).What is the highest normal form for this decomposition?

* D: 3NF

15. Representation metadata contains both structural and semantic data. What is semantic metadata?A:
* The meaning of the data (such as cooking recipes)

16. Based upon the lecture videos, which of the following is needed in order to be successful in the creation and evolution of a metadata registry?

* initial creation of schemata

17. If the record size for a file is greater than the disk block size, then

* C: We must use a spanned record organization for the file.

18. Which of the following organization systems can be used for the physical disk blocks that are allocated to hold records in a file

* All of the above

19. Which of the following would be the most logical reason for choosing a heap/pile organization for afile?

* D: The file is not queried often, but record insertions occur very frequently

20. Consider the following disk pack information:number of disks (platters) = 4Number of read/write heads = 8number of cylinders = 1000track size = 500 bocksHow many blocks can be stored on this disk?

* A: 4,000,000

21. Consider an unordered fileof 330,000 records with a record size of 120 bytes stored on blocks of 4000 bytes with an unspanned record organization. (Assume the blocks are 80% filled.) On average, how many blocks would have to be accessed to find a particular record?

* 6,200

22. Suppose we have an ordered filewith 32,768 records with a records size of 100 bytes stored on blocks of 4000 bytes with an unspanned record organization. (Assume 80% fill on the blocks.) If we perform a binary search using the ordering field, then how many blocks would be accessed on average?

* 10

23. Consider an ordered filewith 330,000 records with a record size of 120 bytes stored on blocks of 4000 bytes with an unspanned record organization. (Assume the blocks are 80% full.) Suppose the ordering/key field is 4 bytes, and a block pointer is 4 bytes. How many index blocks are needed for a single-level, sparse primary index for the file? (Assume index blocks are also 80% full.

* 31

24. Consider an ordered filewith 150,000 records with a record size of 100 bytes stored on blocks of 4000 bytes with an unspanned record organization. (Assume 80% fill in the blocks). Suppose we create a dense secondary index on a unique field. The index field is 4 bytes and a block pointer is 4 bytes. How many index blocks will be needed for our single-level dense secondary index for our file. (Also assume 80% fill on the index blocks.)

* 375

25. Consider a static hash based file organization (hashed on SSN) which contains 1,000 buckets with no more than 2 overflow buckets each. Assuming that the bucket directory is kept in memory, what is the maximumtime it would take to read a record for a given SSN?

Assume the overflow blocks are not stored contiguously, and use the following times:

Avg Seek Time = 0.005 seconds

Avg Rotational Delay = 0.002 seconds

Avg bucket transfer time = 0.0001 second

- 0.0213