Contents

1 Important things to remember

1.1 What is a key?

A key is the *minimum* set of attributes that *all* other attributes depend on.

IF WE REMOVE AN ATTRIBUTE FROM THE KEY, THEN WE LOSE SOME ATTRIBUTE IN THE DEPENDENCY.

Ex: we have $A \to B, C, D$; $B \to E$; $F \to G$; $A, F \to H$

- A, F is the key. If we removed F, then we would lose G being dependent on F, and if we removed A, we'd lose everything else.

1.2 Superkeys

- If you apply the inference rules on them, we should get all the attributes of the relation, because there shouldn't be any attributes that don't eventually depend on a superkey.

2 Decomposition

- Important things we want:
  - Attribute Preservation
  - Dependency Preservation
  - Lossless (nonadditive) Join Property

2.1 Functional Dependency

- Y is functionally dependent on X in R if for every $x \in R.X$, there is exactly one $y \in R.Y$.

2.2 Full Functional Dependency

- Y is FULLY functionally dependent on X in R if it is functionally dependent on X and NOT functionally dependent on any proper subset of X.

2.3 Transitive Dependency

- Y is transitively dependent on X in R if there exists set of attributes Z that is a) NOT A CANDIDATE KEY and b) NOT A SUBSET OF ANY KEY OF R, such that $X \to Z$ and $Z \to Y$ hold.

2.4 Normal Form Rules:

2.4.1 1NF

All values are atomic (no sets, tuples, or relations in relations)

2.4.2 2NF

Relation is in 1NF AND every nonkey attribute is fully functionally dependent on the key

2.4.3 3NF

Relation is in 2NF and every nonkey attribute is nontransitively dependent on the key

2.4.4 BCNF

Every determinant is a candidate key

Practice exam vs. lecture difference:

Lecture has $Email \rightarrow CurrentCity, Salary$; $CurrentCity \rightarrow Salary$ as 2NF. This makes sense because $Salary$, while dependent on $Email$ by itself, also depends on it through $CurrentCity$, which is a nonkey.

Practice exam has the equivalent of $Email \rightarrow CurrentCity, Salary$; $Email, CurrentCity \rightarrow Salary$ as BCNF. This makes sense because $Salary$ now depends on both $Email, CurrentCity$ at once, but $Email, CurrentCity$ is a candidate key, given that it determines $Salary$, so this doesn't count as a transitive dependency.

2.5 Closure of Attributes w.r.t A Dependency Set

- This is all the attributes that depend on the given attributes, based on the dependency set.

3 Important Memory Things

- Blocking factor = number of records per block = `block size / record size`
- Copy time = `seek time + rotation delay + block copy time * numblocks`
- Files are bunches of blocks linked in a linked-list of pointers
- Unspanned = records don't get split across blocks
  - When there's a choice, we go with unspanned
  - If the object is too big for a block, then obviously we must span it
- Bulk transfer: when finding a block, find blocks after it too to save on seek costs

3.1 Heap

- No organization, records are just put into blocks in the order they are inserted.
- To find something, takes on average $N/2$ time

3.2 Sorted

- Records are sorted by key
- Binary search: $\log_2 N$ cost

3.3 Sorted w/ index

- Records are sorted by key
- We build a (sparse) index; entries are keys of the FIRST record in a block
- Binary search: $\log_2 N + 1$ cost, where N is the number of index blocks.
  - $+1$ is for accessing the data block itself.
- A DENSE index has all the keys for data, which will still live in fewer blocks.
- When considering index size and fanout, you must keep in mind the data type of the index, and the size of a block pointer. For example, if the data type is a `varchar28` and the block pointer is 4B, then each index entry costs 32B. If we're filling a block of size 4000B 80% of the way, we have 3200B to fill with index entries. 3200 / 32 = 100, so our fanout is 100.
  - Fanout refers to the number of index entries per index block.
- `number of data blocks / fanout = number of index blocks`
- Primary indices are good for point queries (single access) and range queries (facts about many records in sequence)
- Clustered index defines the order the table is laid out in

3.4 Secondary Index

- Values are not sorted on this field, so we need to grab all the values from the records, THEN sort them.
- If this isn't a key field, then we have to choose whether or not to keep multiple pointers to the block in question.

3.5 Multilevel Index

- I herd you liek indexes
- Lookup is now $\log_{fanout} n + 1$ where n is the number of index blocks

3.6 Hashing

- Each bucket can have a bunch of blocks, generally we'll assume that the hash function uniformly distributes blocks. After that it's just unit conversion.