Tutorial 6 : Normalisation(Contd…) and indexing.

4NF

To satisfy the Fourth Normal Form, a table should satisfy the following two conditions:

1. It should be in the Boyce-Codd Normal Form.
2. And, the table should not have any Multi-valued Dependency.

Multi-valued Dependency:

1. For a dependency A → B, if for a single value of A, multiple value of B exists, then the table may have multi-valued dependency.
2. Also, a table should have at-least 3 columns for it to have a multi-valued dependency.
3. And, for a relation `R(A,B,C)`, if there is a multi-valued dependency between, A and B, then B and C should be independent of each other.

Fifth Normal Form / Projected Normal Form (5NF):

A relation R is in 5NF iff every join dependency in R is implied by the candidate keys of R. A relation decomposed into two relations must have loss-less join Property, which ensures that no spurious or extra tuples are generated, when relations are reunited through a natural join.

Properties – A relation R is in 5NF if and only if it satisfies following conditions:

1. R should be already in 4NF.
2. It cannot be further non loss decomposed (join dependency)

Joint dependency – Join decomposition is a further generalization of Multivalued dependencies. If the join of R1 and R2 over C is equal to relation R then we can say that a join dependency (JD) exists, where R1 and R2 are the decomposition R1(A, B, C) and

R2(C, D) of a given relations R (A, B, C, D). Alternatively, R1 and R2 are a lossless decomposition of R. A JD ⋈ {R1, R2, …, Rn} is said to hold over a relation R if R1, R2, ….., Rn is a lossless-join decomposition. The *(A, B, C, D), (C, D) will be a JD of R if the join of join's attribute is equal to the relation R. Here, *(R1, R2, R3) is used to indicate that relation R1, R2, R3 and so on are a JD of R.

Questions:

1. Find the highest normal form in R (A, B, C, D, E) under following functional dependencies.

```
 ABC --> D
  CD --> AE
```

2. Let R (A, B, C, D, E, P, G) be a relational schema in which the following functional dependencies are known to hold: AB → CD, DE → P, C → E, P → C and B → G. The relational schema R is

(A) in BCNF

(B) in 3NF, but not in BCNF

(C) in 2NF, but not in 3NF

(D) not in 2NF

Answer: (D)

3. A table has fields Fl, F2, F3, F4, F5 with the following functional dependencies

F1 → F3   F2→ F4   (F1 . F2) → F5

In terms of Normalization, this table is in

(A) 1 NF

(B) 2 NF

(C) 3 NF

(D) none

Answer: (A)

4. Relation R has eight attributes ABCDEFGH. Fields of R contain only atomic values. F = {CH -> G, A -> BC, B -> CFH, E -> A, F -> EG} is a set of functional dependencies (FDs) so that F+ is exactly the set of FDs that hold for R.

How many candidate keys does the relation R have?

(A) 3

(B) 4

(C) 5

(D) 6

Answer: (B)

5. Given a Relation R(A, B, C) with functional dependencies-

AB -> C

C -> B

Identify the current normal form of the relation and convert it to BCNF

Answer:

Since, there are no partial dependencies and there are no transitive dependencies we can conclude that the relation is in 3NF.

For a relation to be in BCNF, in all functional dependencies X -> Y ,

X should be a superkey for the relation. We can see that the fd C -> B does not satisfy this rule so we decompose the relation and separate C as follows -

R1 (C, B)

R2 (A, C)

6.Given a relation R(A, B, C, D) with functional dependencies -

A -> BCD

BC -> AD

D -> B

Identify the current normal form and convert it to BCNF.

Answer:

As there are no partial dependencies or transitive dependencies the relation is in third normal form(3NF). To convert into BCNF we need to identify and the functional dependencies whose LHS does not contain a superkey

In this relation A and BC are the candidate keys.

Therefore, D -> B is the violating fd. On separating this fd to create a new relation we get-

R1 (A, D, C), and

R2 (D, B)

Indexing

Cluster indexing

What are the **benefits** of a clustered index?

The data in a clustered index is stored in order. That means:

Finding the data you need in your clustered index is a matter of knowing where to look in our alphabetical list of data. Computers are really good at doing this.
If your data needs to be outputted in the same order that it's stored in—presto!—SQL doesn't need to do any additional sorting.

Non clustered indexing

Using a nonclustered index to find an indexed column's value is fast (SQL is just going through the ordered index data to find the value it needs—once again, something computers are really good at doing). However, if you need other columns of data from the row that you just looked up, SQL is going to have to use those index pointers to go find the rest of that row data somewhere else on disk. This can really add up and slow down performance.

Hash indexing

Best for equality checks.

When updates to the attribute value is infrequent.

Divided into static and dynamic hashing

B - tree vs Hash index

https://www.enterprisedb.com/blog/hash-indexes-are-faster-btree-indexes

References:

https://www.geeksforgeeks.org/database-normalization-normal-forms/

http://agiledata.org/essays/dataNormalization.html

https://tutorialink.com/dbms/dependency-preserving-decomposition.dbms

https://www.youtube.com/playlist?list=PLeNFpOhruv2iM5EFv04SH4d84AO9WD2bA

http://fac.ksu.edu.sa/sites/default/files/E-%20Decomposition.pdf

https://stackoverflow.com/questions/28550601/functional-dependencies-in-databases

https://hackernoon.com/clustered-vs-nonclustered-what-index-is-right-for-my-data-717b329d042c

https://youtu.be/C_q5ccN84C8 (B - Tree)