CS 448/648 Assignment 3 Solutions

1.Consider the following relation scheme for compact discs

CD (Company, Date, CatalogNum, Composer, Track, **Group**, **Artist**, Title, **Instrument**, Duration) together with the following functional dependencies:

{Artist} -> {Instrument, Group}
{Composer, Title, Company} -> {CatalogNum, Track, Duration}
{CatalogNum} -> {Company, Date}
{CatalogNum, Title} -> {Composer}

(a) Use Armstrong's axioms to prove formally that this set of functional dependencies implies that {CatalogNum, Title} -> {Track}. Justify each step of your proof by listing the reason as given (i.e., the functional dependencies is one of the four given ones) or by naming which one of augmentation, transitivity, reflexivity, union, decomposition, or pseudotransitivity was applied.

Solution:
by FD3 {CatalogNum} -> {Company, Date} (1)
by FD4 {CatalogNum, Title} -> {Composer} (2)
augmentation on (1) {CatalogNum, Title} -> {Company, Date, Title} (3)
union on (2) and (3) {CatalogNum, Title} -> {Company, Date, Composer, Title} (4)
by decomposition on (4) {CatalogNum, Title} -> {Company, Composer, Title} (5)
by FD2 {Composer, Title, Company} -> {CatalogNum, Track, Duration} (6)
by transitivity on (5) and (6) {CatalogNum, Title} -> {CatalogNum, Track, Duration} (7)
by decomposition on (7) {CatalogNum, Title} -> {Track}

(b) Give an example of a relation instance for CD having two rows and showing that the dependency {Title, Company} -> {CatalogNum} need not hold when the four given functional dependencies do

Solution:
CD (Company1, Date1, Catalog1, Composer1, Track1, Group1, Art1, Title1, Instr1, Dur1)
CD (Company1, Date1, Catalog2, Composer2, Track1, Group1, Art2, Title1, Instr1, Dur1)

The two tuples agree on Title and Company, but don't agree on Catalog Num. As well FD1 holds because the two tuples don't agree on Author, FD2 because they don't agree on Composer, FD3 and FD4 because they don't agree on CatalogNum

(c) Calculate the closure of the attribute set {CatalogNum, Title} with respect to the given functional dependencies.

$\{CatalogNum, Title\}^+$ incl. {CatalogNum, Title}
(since X+ includes X)
$\{CatalogNum, Title\}^+$ incl. {CatalogNum, Title, Composer}
(from FD4)

{CatalogNum, Title}$^+$ incl. {CatalogNum, Title, Composer, Company, Date}
(from FD3)
{CatalogNum, Title}$^+$ incl. {CatalogNum, Title, Composer, Company, Date, Track, Duration} (from FD2)
{CatalogNum, Title}$^+$ = {CatalogNum, Title, Composer, Company, Date, Track, Duration}
(since no further FDs can be applied)

(d) Find two candidate keys for CD.

Solution: Note that {Artist, Title} must be part of any candidate key because they don't appear on the right side of any of the functional dependencies. If we add {CatalogNum}, we will construct a key, because from (c) {CatalogNum, Title}$^+$ = {CatalogNum, Title, Track, Duration, Company, Composer, Date} and from FD1 {Artist}$^+$ = {Instrument, Group}, i.e. {CatalogNum, Title, Artist} is a key. {Artist, CatalogNum, Title} is a candidate key because it can not be reduced. *Artist* and *Title* cannot be removed because they don't appear on the right side of any FD. On the other hand, {Artist, Title} is not a key because {Artist, Title}? {CatalogNum}.

Another candidate key is: {Artist, Title, Composer, Company}. It is a key because from FD2 {Composer, Title, Company} -> {CatalogNum, Track, Duration} and therefore {Artist, Title, Composer, Company}->{Artist, Title, CatalogNum}, and as shown {Artist,Title,CatalogNum} is a key. It is minimal because if *Composer* or *Company* are removed we will not be able to apply FD2, but *CatalogNum* appears only on the right hand side of FD2.

(e) In the presence of the given functional dependencies, determine whether the join of relations conforming to the schemes Recording (CatalogNum, Company, Composer, Title, Track, Duration, Group, Artist) and Performer (CatalogNum, Group, Artist, Instrument, Date) is lossless with respect to CD and explain why or why not.

Solution: By a theorem from the textbook the decomposition of R into R_1 and R_2 is lossless iff $R_1 \cap R_2$ -> R_1 or $R_1 \cap R_2$ -> R_2. In our case $R_1 \cap R_2$ = {CatalogNum, Artist, Group} and $R_1 \cap R_2$ -> R_2 (i.e. {CatalogNum, Artist, Group}-> {CatalogNum, Group, Artist, Instrument, Date}) because {CatalogNum, Artist, Group}+ = {CatalogNum, Artist, Group, Instrument, Company, Date}, which includes all attributes from R2). Therefore the join is lossless with respect to CD.

2. Consider the relation scheme R (A, B, C, D, E, F, G) and the set of functional dependencies {AB->C, BC->D, DEF->G, DG->EF, DG->AB}, where AB is a shorthand notation representing the set {A, B} etc.

(a) Find a lossless join decomposition into relation schemes, all of which are in Boyce-Codd Normal Form.

Solution:

The functional dependency BC->D violates the BCNF for R because BC is not a superkey in R. Decomposing R relative to BC->D, we get the decomposition R_1={B,C,D} R_2={A,B,C,E,F,G}. The functional dependency AB->C violates the BCNF for R_1 because AB is not a superkey for R_2. Decomposing R_2 relative to AB->C we get the relations: {A,B,C} and {A,B,E,F,G}. The set of relations {B,C,D},{A,B,C},{A,B,E,F,G} is the resulting decomposition. The relations are in BCNF because BC and AB are the candidate keys in the first two relations and ABEF and ABG are candidate keys in the 3rd relation).

If we started with AB->C, we would end up with the same three relations (if we first form {A,B,C,D} and {A,B,E,F,G}), or perhaps with { {A,B,C}, {A,B,D}, {A,B,E,F,G} }.

(b) Is the resulting set of relation schemes dependency preserving with respect to the given functional dependencies? Explain why or why not.

Solution:
It is not, the FDs DEF->G, DG->EF and DG-AB are not preserved.

3.Consider the relation scheme Cities (Name, Province, Population, KilometresFromWaterloo) and queries as follows:

```
Q1: select Name, Province
      from Cities
      where Population > 100000
Q2: select Name, Population, KilometresFromWaterloo
      from Cities
     where Province = ON. and KilometresFromWaterloo < 100
Q3: select Province, count(city), sum(Population)
     from Cities
     group by Province
```

(a) Briefly explain how a B+-tree index on Population could be used during processing of Q1.

Solution:
During query execution we can find first the first entry with key >100000 in the B^+ tree. Then we can follow the *next* pointers in the B^+ tree to traverse the rest of the B^+ leaf nodes comprising the result. The result itself is compiled by reporting the Name and Province values from the tuples referenced by the *data pointers* of the traversed leaf nodes in the B^+ tree.

(b) Briefly explain how a clustering index on the pair of attributes (Province, KilometresFromWaterloo) could be used during processing of Q2.

Solution:

We can look up in the clustered index the first record for which Province = ON and smallest value for KilometresFromWaterloo, and return the appropriate attribute values from the block of tuples pointed by this record if KilometresFromWaterloo < 100. We can then proceed scanning the records from the table in the order they are stored and return the referenced block of records until we reach a clustering value for which Province? ON or KilometresFromWaterloo >= 100.

(c) Briefly explain how a hash-table index could be used during processing of Q2.

Solution:
If we have a hash table on Province, then we can calculate the hash function for "ON" and look up the pointers to the Cities tuples from the ON entry in the hash table. (In principle, if we have a hash table on KilometresFromWaterloo, we can use it by calculating the hash function for the values 0 to 99, and retrieving the corresponding records; but this would likely be quite expensive.)

(d) If separate B+-tree indexes were available on Province and on KilometresFromWaterloo, explain which would likely be more selective for processing Q2.

Solution:
All cities <100 km from Waterloo are in Ontario. Therefore the condition Province=ON will give us all cities in Ontario and the condition KilometeresFromWaterloo <100 will give us only cities **in Ontario** which are less than 100 km from Waterloo. Therefore the condition KilometresFromWaterloo <100 is more selective.

(e) Briefly explain why using a clustering index on Province would be more efficient than either a hash-table or B+-tree index during processing of Q3.

Solution:
Note that a sequential traversal of a clustered index on Province will give us all records of the first province (alphabetically), followed by all records from the second province and so on, i.e. it will give us the Cities grouped by province, which is exactly the information we need to compute the query. On the other if a hash-table or B+ tree index is available, more processing has to be done. Note that, even though we find the *references* to the tuple for each province as a group, the records themselves are not physically co-located.