THE UNIVERSITY OF SYDNEY

Assignment 4

ISYS2120: Data and Information Management

Group R10_82

Question A: Relational Algebra

This question's parts all refer to a relational schema as follows:

- `Patient(AdmitNo, PName, Insurance)`
- `Nurse(NStaffNo, NName, NSalary)`
- `Doctor(DStaffNo, DName, DSalary)`
- `Locate(AdmitNo, Ward)`
- `Supervision(NStaffNo, SupervisorNStaffNo)`
- `Assign(NStaffNo, Ward)`
- `Illness(AdmitNo, Disease)`
- `Expertise(DStaffNo, Disease)`
- `Treatment(AdmitNo, DStaffNo, Disease)`

A1. Give a relational algebra expression to calculate the table containing the `AdmitNo`, `PName` for those patients who have the illness `'COVID'`.

A2. Give a relational algebra expression to calculate the table containing the `DStaffNo`, `DName` of those Doctors who treat a patient who has insurance `'NIB'`.

A3. Give a relational algebra expression to calculate the table containing the `NStaffNo` and `DStaffNo` of a pair of doctor and nurse where the doctor treats a patient who is located on the ward to which the nurse is assigned.

A4. Give a relational algebra expression to calculate the table containing the `AdmitNo` and `DStaffNo` of a pair of patient and doctor, where the patient has an illness in which the doctor has expertise.

A5. Give a relational algebra expression to calculate the table containing the `NStaffNo`, `NName` and `DStaffNo`, `DName` for pairs of Doctor and Nurse where the Nurse is assigned to a ward where there is a Patient who is treated by the Doctor for the disease `'Anthrax'`.

A1.

$$\Pi_{\text{AdmitNo,PName}}(\sigma_{\text{Disease = `COVID'}}(\text{Patient} \bowtie \text{Illness})) \tag{1}$$

A2.

$$\Pi_{\text{DStaffNo,DName}}(\sigma_{\text{Insurance = `NIB'}}(\text{Doctor} \bowtie \text{Treatment} \bowtie \text{Patient})) \tag{2}$$

A3.

$$\Pi_{\text{NStaffNo,DStaffNo}}(\text{Doctor} \bowtie \text{Treatment} \bowtie \text{Locate} \bowtie \text{Assign}) \tag{3}$$

A4.

$$\Pi_{\text{AdmitNo,DStaffNo}}(\text{Treatment} \bowtie \text{Expertise}) \tag{4}$$

A5.

$$R := \text{Doctor} \bowtie \text{Nurse} \bowtie \text{Assign} \bowtie \text{Locate} \bowtie \text{Treatment} \tag{5}$$

$$S := \sigma_{\text{Disease = `Anthrax'}}(R) \tag{6}$$

$$\text{Answer} = \Pi_{\text{NStaffNo,NName,DStaffNo,DName}}(S) \tag{7}$$

Question B: Relational Design Theory

Consider the following relational design, which collects data about the usage of drugs at several hospitals:

```
DrugUsage(HospitalName, HospitalNumber, DiseaseCode, DrugName,
    SizeOfDose, Cost, ManufacturerName, ManufacturerAddress)
```

The following functional dependencies are valid in this schema:

1. `HospitalNumber` $\longrightarrow$ `HospitalName`
2. `HospitalName` $\longrightarrow$ `HospitalNumber`
3. `DrugName` $\longrightarrow$ `DiseaseCode, ManufacturerName`
4. `DrugName, SizeOfDose` $\longrightarrow$ `Cost`
5. `ManufacturerName` $\longrightarrow$ `ManufacturerAddress`

B1. (i) Calculate the attribute closure `HospitalNumber+`. Show your working.
(ii) Calculate the attribute closure `HospitalName+`. Show your working.
(iii) Calculate the attribute closure `DrugName+`. Show your working.
(iv) Calculate the attribute closure (`DrugName, SizeOfDose`)+. Show your working.
(v) Calculate the attribute closure `ManufacturerName+`. Show your working.

B2. State whether the relation `DrugUsage` is in BCNF. Show your working.

B3. Give a decomposition of the relation `DrugUsage`, into two or more relations, with the properties that the decomposition is both lossless-join and dependency preserving. Explain how you know that the decomposition has these properties. For each relation in the decomposition, state whether or not that relation is itself in BCNF, and explain your working.

B1. Note in the following answers that the attribute closure $\{A_1, A_2, \ldots, A_n\}^+$ will always contain the attributes $\{A_1, A_2, \ldots, A_n\}$ as they are trivially dependent (per the reflexivity axiom). This is assumed and therefore is not explained in each part.

(i) From FD 1 (`HospitalNumber` $\longrightarrow$ `HospitalName`) we can retrieve `HospitalName` from `HospitalNumber`. Hence,

$$\{\text{HospitalNumber}\}^+ = \{\text{HospitalNumber, HospitalName}\} \tag{8}$$

(ii) From FD 2 (`HospitalName` $\longrightarrow$ `HospitalNumber`) we can retrieve `HospitalNumber` from `HospitalName`. Hence,

$$\{\text{HospitalName}\}^+ = \{\text{HospitalName, HospitalNumber}\} \tag{9}$$

As attribute closures are mathematical sets, this is an identical closure to part i).

(iii) We begin by retrieving `DiseaseCode` and `ManufacturerName` from FD 3 (`DrugName` $\longrightarrow$ `DiseaseCode`, `ManufacturerName`).

$$\{\text{DrugName}\}^+ = \{\text{DrugName, DiseaseCode, ManufacturerName}\} \tag{10}$$

Using the `ManufacturerName` attribute, we can now obtain the `ManufacturerAddress` using FD 5 (`ManufacturerName` $\longrightarrow$ `ManufacturerAddress`):

$$\begin{aligned}\{\text{DrugName}\}^+ = \{&\text{DrugName, DiseaseCode,}\\ &\text{ManufacturerName, ManufacturerAddress}\}\end{aligned} \tag{11}$$

As we do not have the `SizeOfDose` in the closure, we cannot obtain the `Cost` attribute. Equation (11) is the final attribute closure.

(iv) Here we are required to find the attribute closure of {`DrugName`, `SizeOfDose`}. The augmentation rule states that $X \longrightarrow Y \implies XZ \longrightarrow YZ$, and thus it makes sense for {`DrugName`, `SizeOfDose`}$^+$ to contain *at least* all the elements of {`DrugName`}$^+$. Hence our starting point for this question is equation (11) with the additional attribute of `SizeOfDose` through the reflexivity principle:

$$\begin{aligned}\{\text{DrugName, SizeOfDose}\}^+ = \{&\text{DrugName, SizeOfDose, DiseaseCode,}\\ &\text{ManufacturerName, ManufacturerAddress}\}\end{aligned} \tag{12}$$

Now that both the `DrugName` and `SizeOfDose` attributes are present in our closure, we can use FD 4 (`DrugName`, `SizeOfDose` $\longrightarrow$ `Cost`) to obtain the `Cost` attribute.

$$\begin{aligned}\{\text{DrugName, SizeOfDose}\}^+ = \{&\text{DrugName, SizeOfDose, DiseaseCode,}\\ &\text{ManufacturerName, ManufacturerAddress,}\\ &\text{Cost}\}\end{aligned} \tag{13}$$

(v) From functional dependency number 5 (`ManufacturerName` $\longrightarrow$ `ManufacturerAddress`) we can obtain `ManufacturerAddress` from `ManufacturerName`. Thus,

$$\{\text{ManufacturerName}\}^+ = \{\text{ManufacturerName, ManufacturerAddress}\} \tag{14}$$

B2. A relation R is in Boyce-Codd Normal Form (BCNF) if and only if for every non-trivial FD $A_1A_2\ldots A_n \longrightarrow B_1B_2\ldots B_m$ of R, it is the case that $\{A_1, A_2, \ldots, A_n\}$ is a superkey for R. In other words, the left side of every nontrivial FD must be a superkey of the relation it defines. Hence, in order to state whether the relation `DrugUsage` is in BCNF, it is necessary to first find out what a possible superkey of the relation is.

After completing part B1, it is an appealing ansatz for a superkey of the relation to be {`HospitalNumber`, `DrugName`, `SizeOfDose`}. Let us calculate the attribute closure of our conjecture to verify this.

This is made simple through using the augmentation rule once again – we can begin with the combination of our results from the attribute closure of {`HospitalNumber`} in equation (8) and the attribute closure of {`DrugName`, `SizeOfDose`} in equation (13) in order to calculate {`HospitalNumber`, `DrugName`, `SizeOfDose`}$^+$:

$$\begin{aligned}\{\text{HospitalNumber, DrugName, SizeOfDose}\}^+ = \{&\text{HospitalNumber, HospitalName,}\\ &\text{DrugName, SizeOfDose,}\\ &\text{DiseaseCode, ManufacturerName,}\\ &\text{ManufacturerAddress, Cost}\}\end{aligned} \tag{15}$$

Fortunately, there are no more attributes remaining in the relation for this closure to cover, and so our calculation stops with equation (15). Indeed, our ansatz was correct: {`HospitalNumber`, `DrugName`, `SizeOfDose`} is a superkey for the relation `DrugUsage`. In fact, these three attributes form a (minimal) key for the relation.

Now, as none of the given FD's for the relation contain the attributes of the superkey on their LHS, `DrugUsage` is not in BCNF.

B3. In order to convert the relation `DrugUsage` into BCNF, we must utilise the BCNF decomposition algorithm as described in Algorithm 1.

Algorithm 1 (BCNF decomposition theorem[a])**.** Given a relation R_0 with a set of functional dependencies S_0, this algorithm returns a decomposition of R_0 into a collection of relations, all of which are in BCNF. Apply the following steps recursively to all relations in the decomposition, starting with $R = R_0$ and $S = S_0$.

1. Check whether R is in BCNF. If so, the algorithm is finished. Return R.
2. If there are BCNF violations in the set S, let one be $X \longrightarrow Y$. Compute X^+.
3. Choose $R_1 = R_0 - X^+ + X$ and $R_2 = X^+$.
4. Compute the sets of FD's for R_1 and R_2; let these be S_1 and S_2 respectively.
5. Recursively decompose $\{R_1, S_1\}$ and $\{R_2, S_2\}$ with this algorithm. Return the union of the results of these decompositions.

[a]Based on Garcia-Molina, H., Ullman, J.D. and Widom, J. (2014) *Database systems: The complete book*, Algorithm 3.20

Lossless-join decompositions

It is also worthwhile noting why Algorithm 1 necessarily produces lossless-join decompositions relations R_1 and R_2 from R_0.

According to Theorem 1, a decomposition is lossless-join if and only if the set of common attributes between the decomposed relations are a superkey for at least one of the relations. Notice that in Algorithm 1, we chose to split R_0 based on the set X, one of its own functional dependencies. In fact, R_2 is entirely composed of the attributes in the closure of X, meaning that X is the superkey of the relation R_2. As R_1 is composed in such a way that $R_1 \cap R_2 = X$, we have satisfied the condition of Theorem 1 — the common attributes between the two decomposed relations (X) are a superkey for at least one of the relations (R_2). Therefore, Algorithm 1 always produces lossless-join decompositions of R_0.

Theorem 1 (Lossless-join decomposition theorem[a]). *The decomposition of R into X and Y is* lossless-join *w.r.t. a set of FDs F if and only if the closure of F contains*

$$X \cap Y \longrightarrow X \qquad \text{or} \tag{16}$$

$$X \cap Y \longrightarrow Y \qquad \text{or both.} \tag{17}$$

[a]Based on Fekete, A., (2022) *ISYS2120 Week 6 slides*, Slide 48

Dependency preserving decompositions

In each of the relations formed from decomposing R_0, we will note whether the decomposition is dependency preserving based on Definition 1.

Definition 1 (Dependency preserving decompositions[a]). Given a relation R with a set of FDs F, a decomposition of R into a set of relations R_i with FDs F_i is a *dependency preserving decomposition* if

$$(F_1 \cup F_2 \cup \ldots F_n)^+ = F+ \tag{18}$$

where F_X is the projection of F onto X.

[a]Based on Fekete, A., (2022) *ISYS2120 Week 6 slides*, Slide 50

In simple terms, if you take the closure of the set of functional dependencies for the smaller relations, you should retain the original set of functional dependencies.

Let us now begin decomposing the relation $R_0 =$ `DrugUsage`(`HospitalName`, `HospitalNumber`, `DiseaseCode`, `DrugName`, `SizeOfDose`, `Cost`, `ManufacturerName`, `ManufacturerAddress`) into Boyce-Codd normal form using Algorithm 1.

$\mathbf{R_0 \longmapsto R_1, R_2}$

We know from question B2 that a superkey for R_0 is {`HospitalNumber`, `DrugName`, `SizeOfDose`}. We also know that R_0 is satisfied by the following set of functional dependencies S_0:

1. `HospitalNumber` $\longrightarrow$ `HospitalName`
2. `HospitalName` $\longrightarrow$ `HospitalNumber`
3. `DrugName` $\longrightarrow$ `DiseaseCode`, `ManufacturerName`
4. `DrugName`, `SizeOfDose` $\longrightarrow$ `Cost`
5. `ManufacturerName` $\longrightarrow$ `ManufacturerAddress`

All of the above functional dependencies violate BCNF so we can begin the decomposition using any. Let's choose FD 3 to begin i.e. $X = \{\texttt{DrugName}\}$.

We already know the closure of X from equation (11):

$$\begin{aligned}\{\text{DrugName}\}^+ = \{&\text{DrugName, DiseaseCode,} \\ &\text{ManufacturerName, ManufacturerAddress}\}\end{aligned} \tag{11}$$

So let us now split up the relation R_0 into:

$$R_1 = R_0 - X^+ + X \tag{19}$$
$$= (\text{HospitalNumber, HospitalName, DrugName, SizeOfDose, Cost}) \tag{20}$$

and

$$R_2 = X^+ \tag{21}$$
$$= (\text{DrugName, DiseaseCode, ManufacturerName, ManufacturerAddress}) \tag{22}$$

with R_1 having the functional dependencies S_1:

1. `HospitalNumber` $\longrightarrow$ `HospitalName`
2. `HospitalName` $\longrightarrow$ `HospitalNumber`
3. `DrugName`, `SizeOfDose` $\longrightarrow$ `Cost`

and R_2 having the functional dependencies S_2:

1. `DrugName` $\longrightarrow$ `DiseaseCode`, `ManufacturerName`
2. `ManufacturerName` $\longrightarrow$ `ManufacturerAddress`

Note that here $S_1 \cup S_2 = S_0$. Therefore,

$$(S_1 \cup S_2)^+ = S_0^+ \tag{23}$$

and thus according to Definition 1 this decomposition is dependency preserving.

$\mathbf{R_1 \longmapsto R_3, R_4}$

We now repeat the algorithm with R_1 and it's functional dependencies S_1. It is clear that a superkey of the relation is $\{\texttt{HospitalName}, \texttt{DrugName}, \texttt{SizeOfDose}\}$. It is therefore also apparent that all FDs 1, 2 and 3 violate BCNF.

Here we choose to base our decomposition of R_1 on FD 3. We note from equation (13) that the closure of FD 3 is $\{\texttt{DrugName}, \texttt{SizeOfDose}, \texttt{Cost}\}$, and so the two new relations formed from R_1 are:

$$\begin{aligned} R_3 &= R_1 - \{\text{DrugName, SizeOfDose, Cost}\} + \{\text{DrugName, SizeOfDose}\} && (24)\\ &= (\text{HospitalNumber, HospitalName, DrugName, SizeOfDose}) && (25) \end{aligned}$$

and

$$R_4 = (\text{DrugName, SizeOfDose, Cost}) \tag{26}$$

with R_3 having the functional dependencies S_3:

1. $\texttt{HospitalNumber} \longrightarrow \texttt{HospitalName}$
2. $\texttt{HospitalName} \longrightarrow \texttt{HospitalNumber}$

and R_4 having the functional dependencies S_4:

1. $\texttt{DrugName}, \texttt{SizeOfDose} \longrightarrow \texttt{Cost}$

Note that R_4 is now in BCNF as the left-hand side of its only functional dependency is a superkey of the relation! Also note that this decomposition was dependency preserving for similar reasons as detailed for equation (23).

$\mathbf{R_3 \longmapsto R_5, R_6}$

We now repeat the algorithm with R_3 and it's functional dependencies S_3. As R_3 has superkey $\{\texttt{HospitalName}, \texttt{DrugName}, \texttt{SizeOfDose}\}$, both of its FDs violate BCNF. Choosing FD 1 to base our decomposition off, we know from equation (8) that $\{\texttt{HospitalNumber}\}^+ = \{\texttt{HospitalNumber}, \texttt{HospitalName}\}$. Thus R_3 is decomposed into:

$$\begin{aligned} R_5 &= R_3 - \{\text{HospitalNumber, HospitalName}\} + \{\text{HospitalNumber}\} && (27)\\ &= (\text{HospitalNumber, DrugName, SizeOfDose}) && (28) \end{aligned}$$

and

$$R_6 = (\text{HospitalNumber, HospitalName}) \tag{29}$$

with R_5 having only trivial dependencies in its dependency set S_5, and R_6 having the functional dependencies S_6:

1. $\texttt{HospitalNumber} \longrightarrow \texttt{HospitalName}$

2. `HospitalName` $\longrightarrow$ `HospitalNumber`

Note that both R_5 and R_6 are both in BCNF as the LHS of their functional dependencies are superkeys of their respective relations. Furthermore, this decomposition was dependency preserving for similar reasons as outlined for equation (1).

$\mathbf{R_2 \longmapsto R_7, R_8}$

We now repeat the algorithm with R_2 and it's functional dependencies S_2. R_2 has superkey {`DrugName`}. The closure of `DrugName` is once again seen in equation (11). For the first time in this schema, not all of the relation's FDs violate BCNF! The violator in this case is FD 2 with closure seen in equation (14) — $\{\texttt{ManufacturerName}\}^+ = \{\texttt{ManufacturerName}, \texttt{ManufacturerAddress}\}$. We will use this to decompose R_2 into:

$$\begin{aligned} R_7 &= R_2 - \{\text{ManufacturerName, ManufacturerAddress}\} + \{\text{ManufacturerName}\} && (30) \\ &= (\text{DrugName, DiseaseCode, ManufacturerName}) && (31) \end{aligned}$$

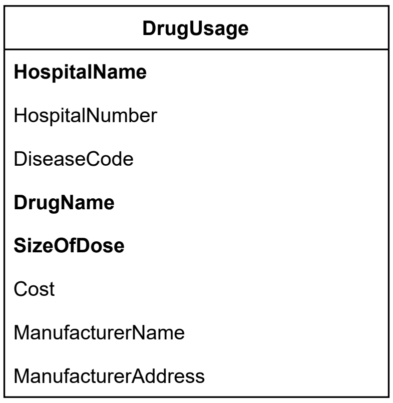
and

$$R_8 = (\text{ManufacturerName, ManufacturerAddress}) \qquad (32)$$

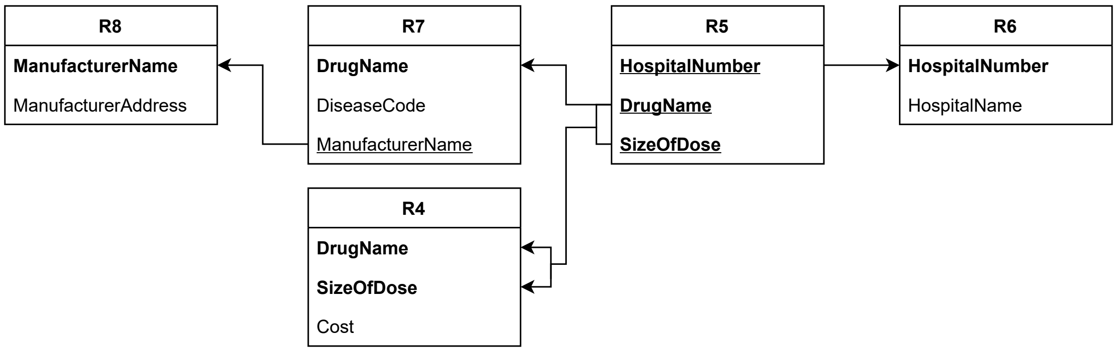
Both R_7 and R_8 are in BCNF, and once again this decomposition was dependency preserving.

The final result

Returning the union of all final results from the above decompositions, $R_0 \longmapsto R_4 \cup R_5 \cup R_6 \cup R_7 \cup R_8$. See Figure 1 for the comparison between the original relation `DrugUsage` and the normalised version.



(a) The original schema design of `DrugUsage`



(b) The normalised schema in Boyce-Codd normal form after decomposition

Figure 1: A comparison of the original and normalised schema design of `DrugUsage`

Question C: Index choices

Suppose a database has a table `T(a, b, c, d)` with primary key being the attribute `a`. For each query below, explain whether the query can be answered efficiently without creating any extra index; if the default structure does not allow efficient answering, you should write a `CREATE INDEX` statement that will improve performance for this query. You should also note whether or not the index you propose covers the query.

C1.
```
SELECT *
  FROM T
 WHERE T.b = 53;
```

C2.
```
SELECT T.c
  FROM T
 WHERE T.a = 'XYZ';
```

C3.
```
SELECT T.a, T.c
  FROM T
 WHERE T.c > 34 AND T.c < 47;
```

C4.
```
SELECT T.a
  FROM T
 WHERE T.c = 53 AND T.d = 81;
```

C5.
```
SELECT *
  FROM T
 WHERE T.d = 81 AND T.b BETWEEN 49 AND 61;
```

The following answers assume that the table `T(a, b, c, d)` contains a clustered primary index on `a`.

C1. Query C1 involves an equality lookup on the attribute `b` and thus it would be beneficial to create a secondary, unclustered hash-based index on the attribute `b`. This is not a covering index as the index does not cover all the attributes queried (`a, b, c, d`).

```
CREATE INDEX hash_index_b
ON T USING Hash (b);
```

C2. Query C2 involves an equality lookup on the attribute `a`, which is the primary key of the relation `T`. The query is already quite efficient as PostgreSQL automatically creates a clustered index on the primary key of any relation. However, it is possible to further speed up the query by creating a covering index on `a` and `c` so that all data can be extracted from just the index table. The DBMS therefore doesn't have to retrieve all the pages from disk containing tuples with `T.a = 'XYZ'`.

```
CREATE INDEX hash_index_a_c
ON T USING Hash (a) INCLUDE (c);
```

C3. Query C3 involves selecting columns `a` and `c` and filtering rows using a range query on `c`. Due to the presence of the range query, it is highly beneficial for query optimisation for a clustered B-tree index to be created on `c`. Furthermore, including `a` in the index will create a covering index, further improving performance of the query.

```
CREATE INDEX clustered_index_c
ON T USING B-tree (c ASC) INCLUDE (a);
CLUSTER T USING clustered_index_c;
```

C4. Query C4 involves equality lookups on attributes `c` and `d` and so it would be most beneficial to create an unclustered hash-index on attributes `(c, d)`. The reason we include `a` in our index is because we want all attributes in the query to be included in the index table as search keys, forming a covering index. This will ensure no readings into the actual data table, making the query more efficient.

```
CREATE INDEX hash_index_c_d
ON T USING Hash (c, d) INCLUDE (a);
```

C5. Query C5 involves an equality lookup on attribute `d` and a range query on attribute `b`. Because none of these attributes are the primary key, the query will strongly benefit from the creation of extra indexes. To speed up this type of query, we should create a clustered B-tree index on `(b, d)`. Notice that the order and sorting of the index here is important: we specifically choose to index on `b` first and *then* `d`. We also choose to cluster the relation `T` using `b` and `d` sorted in ascending order. These two choices this will drastically help improve the speed of the range-equality lookup in the query. The DBMS simply has to find the first entry in the index table with `T.b = 49` and then look in the second column for whether `T.d = 81`. The second column is in ascending order so all that is required is a vertical scan of the elements until `d = 81`. This is not a covering index as the index does not cover all the attributes queried (`a, b, c, d`).

```
CREATE INDEX clustered_index_b_d
ON T USING B-tree (b ASC, d ASC);
CLUSTER T USING clustered_index_bd;
```