INFO20003 Tutorial – Week 9 Solutions

(Tutorial: Normalisation)

Objectives:

This tutorial will cover:

I. Review of normalisation concepts – 15 mins
II. Normalisation exercises – 35 mins
III. *Break – 5 mins*
IV. Normalisation exercises, continued – 50 mins

Key Concepts:

NOTE for students: *This is a brief summary of some of the concepts taught in lecture 15. The lectures contain detailed content related to these and many more concepts. These notes should be considered quick revision instead of a sole resource for the course material.*

- Anomalies

  Consider the following instance of the relation Allocation (CourseNumber, Tutor, Room, Seats):

CourseNumber	Tutor	Room	Seats
INFO20003	Farah	Alice Hoy 109	30
COMP10001	Farah	EDS 6	25
INFO30005	Patrick	Sidney Myer G09	20
COMP20005	Alan	Sidney Myer G09	20

  - An **update** anomaly is a data inconsistency that results from data redundancy and partial update when one or more instances of duplicated data are updated but not all. For example, suppose the room Sidney Myer G09 has been improved, and there are now 30 seats. In this single entity, we will have to update all other rows where room = Sidney Myer G09.

  - A **deletion** anomaly is an unintentional loss of certain attribute values due to the deletion of other data for other attributes. If we remove COMP10001 from the above table, the details of room EDS 6 are also deleted.

  - An **insertion** anomaly is the inability to add certain attributes to a database due to absence of other attributes. For example, a new room "NewAlice109" has been built but has not yet been timetabled for any course or members of staff.

  If the relation is in denormalised form, it can lead to the above-mentioned anomalies. Normalisation helps us remove such anomalies and get rid of redundant data by iteratively improving relations. This requires understanding of the functional dependencies of the attributes of a given relation and resolving transitive and partial dependencies to achieve normal forms.

- Functional dependency

  For a particular relation R, a functional dependency occurs when a subset of R's attributes {A_1, A_2, ..., A_n} determine attributes {B_1, B_2, …, B_n}. If several tuples agree on the values of A_1, A_2, ..., A_n, they must agree on the values of B_1, B_2, …, B_n – that is, if two records have the same A_1, A_2, ..., A_n then they have the same B_1, B_2, …, B_n. Therefore A_1, A_2, ..., A_n "functionally determine" B_1, B_2, …, B_n. This is written as:

  $$\mathbf{A_1, A_2, \ldots, A_n \rightarrow B_1, B_2, \ldots, B_n}$$

  A relation R satisfies a functional dependency (FD) if and only if the FD is true for every instance of R.

- Determinants

  The attributes that determine the value of other attributes are called determinants. For example, consider the relation below:

  Person (ssn, name, birthdate, address, age)

  birthdate → age
  ssn → name, birthdate, age, address

  In this example, *birthdate* and *ssn* are determinants, as *birthdate* determines *age* and *ssn* determines the rest of the attributes.

- Key and non-key attributes

  A *key* is a set of attributes {A_1, A_2, …, A_n} for a relation R, such that {A_1, A_2, ..., A_n} functionally determines all other attributes of R and no subset of {A_1, A_2, …, A_n} functionally determines all other attributes of R. The key must be minimal.

  In the relation Person (ssn, name, birthdate, address, age), *ssn* is the minimal key of the Person relation, but {*ssn*, *name*} is not (it is a "super key").

- Partial functional dependency

  A partial functional dependency arises when one or more non-key attributes are functionally determined by a *subset* of the primary key. This is shown in Figure 1.

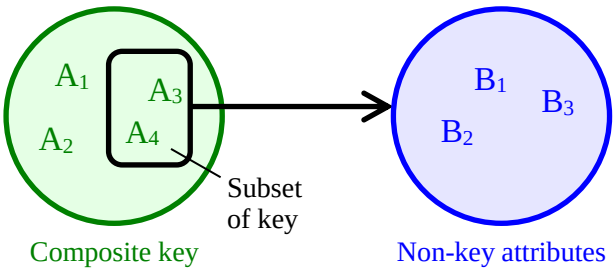

Figure 1: A partial functional dependency (subset of composite key determining some non-key attributes)

Consider a relation R (A, B, C, D) with a composite primary key of (A, D), satisfying the functional dependencies A → B, D → C. For this relation, AD determines BC (AD → BC: AD can uniquely identify BC). However, to identify B, attribute A is enough and similarly D can

identify C. Functional dependencies like A → B and D → C are called *partial functional dependencies*. For example, in the relation Order (Order#, Item#, Desc, Qty) from the lecture, Order# and Item# are the keys. Nonetheless, the item description, Desc, can be determined by Item# alone.

- Transitive functional dependency

  When a non-key attribute is determined by another non-key attribute (or by a subset of PK and non-key attributes), such a dependency is called a *transitive functional dependency*. This is shown in Figure 2.

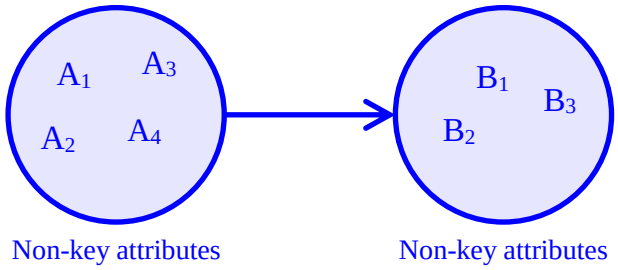

Figure 2: A transitive functional dependency (non-key attributes determining other non-key attributes)

- Armstrong's Axioms

  Given a relation and a set of functional dependencies (FDs), we can discover new functional dependencies using some rules generally known as Armstrong's Axioms. Three important axioms required to determine FDs are Reflexivity (also known as "trivial FDs"), Augmentation and Transitivity.

  Reflexivity: Suppose $A = \{A_1, A_2, ..., A_n\}$ is a subset of attributes of R, and $B = \{B_1, B_2, \ldots, B_n\}$ is also a subset of attributes of R such that B is a subset of A. Reflexivity implies that

  $$B \subseteq A \Longrightarrow A \rightarrow B$$

  For example, in the case of Person (ssn, name, birthdate, address, age),

  $$\text{ssn, name} \rightarrow \text{name}$$

  Augmentation: Consider another subset of attributes $C = \{C_1, C_2, \ldots, C_n\}$. Augmentation implies that

  $$A \rightarrow B \Longrightarrow AC \rightarrow BC$$

  In the case of Person (ssn, name, birthdate, address, age), we can take the above FD and write

  $$\text{ssn, name, age} \rightarrow \text{name, age}$$

  Transitivity:

  $$A \rightarrow B \text{ and } B \rightarrow C \Longrightarrow A \rightarrow C$$

  For our relation Person, we can say ssn → birthdate, birthdate → age ⟹ ssn → age.

- Normalisation and normal forms

  Normalisation is a technique used to iteratively improve relations to remove undesired redundancy by decomposing relations and eliminating anomalies. The process is iterative and can be performed in stages generally referred to as Normal Forms. In First Normal Form (1NF), the relation is analysed and all repeating groups are identified to be decomposed into new relations. In Second Normal Form (2NF), all the partial dependencies are resolved/removed. The next stage is Third Normal Form (3NF) where all the transitive dependencies are removed.

Exercises:

1. Consider the relation Diagnosis with the schema Diagnosis (DoctorID, DocName, PatientID, DiagnosisClass) and the following functional dependencies:

   DoctorID → DocName
   DoctorID, PatientID → DiagnosisClass

   Consider the following instance of Diagnosis:

DoctorID	DocName	PatientID	DiagnosisClass
D001	Alicia	P888	Flu
D002	John	P999	Lactose intolerance
D003	Jennifer	P000	Flu
D002	John	P111	Fever

   Identify different anomalies that can arise from this schema using the above instance.

   From the two given FDs, we can infer that the key for Diagnosis is (DoctorID, PatientID) since together they are sufficient to uniquely identify each record. The following anomalies can arise from the given schema:

   Insertion anomaly: If we require inserting data for a new doctor like DoctorID and DocName, we must insert data of at least one patient associated with the doctor. This inability to insert records of particular fields is insertion anomaly.

   Deletion anomaly: Deleting patient's data can result in the loss of doctor's data as well resulting in deletion anomaly. For example, if we delete P888 data from the table we lose record for the doctor named Alicia as well.

   Update anomaly: One doctor may be associated with more than one patient. In such case, an update anomaly may result if a doctor's name is changed for only one patient. For example, if we want to change the doctor's name from "John" to "John Miller", we have to do it for two records. Failing to do so will result in an update anomaly.

2. Consider a relation R (A, B, C, D) with the following FDs:

   $$AB \rightarrow C,\ AC \rightarrow B,\ BC \rightarrow A,\ B \rightarrow D$$

   The possible candidate keys of R are AB, AC, and BC, since each of those combinations is sufficient to uniquely identify each record. Let's consider AB for instance. From AB → C we see that AB uniquely identifies C, and since B alone uniquely identifies D, AB together have covered CD, i.e. the entire set of attributes.

List all the functional dependencies that violate 3NF. If any, decompose R accordingly. After decomposition, check if the resulting relations are in 3NF, if not decompose further.

To be in 3NF, a relation should be in 2NF and all the transitive functional dependencies should be removed. The relation is not in 2NF as there is a partial functional dependency B → D. B is part of a composite candidate key (AB and BC) and does not require any other key attribute to determine D. Hence, we need to decompose R into following relations: R1 (A, B, C) and R2 (B, D).

R1 and R2 don't have any partial or transitive functional dependencies hence both are in 3NF.

3. Consider the following relation StaffPropertyInspection:

StaffPropertyInspection (propertyNo, pAddress, iDate, iTime, comments, staffNo, sName)

The FDs stated below hold for this relation:

propertyNo, iDate → iTime, comments, staffNo, sName
propertyNo → pAddress
staffNo → sName

From these FDs, it is safe to assume that propertyNo and iDate can serve as a primary key. Your task is to normalise this relation to 3NF. Remember in order to achieve 3NF, you first need to achieve 1NF and 2NF.

As the relation is already in 1NF, because there are no repeating groups, we can directly check if it is in 2NF or not.

Second Normal Form: The functional dependency propertyNo → pAddress shows that pAddress is determined by part of the key. Hence, the relation is not in 2NF. It can be decomposed as:

Property (propertyNo, pAddress)

PropertyInspection (FK propertyNo, iDate, iTime, comments, staffNo, sName)

Third Normal Form: Now out of the two relations, Property is already in 3NF since there is no transitive dependency in this relation. Regarding PropertyInspection, we can observe that the FD staffNo → sName violates 3NF, as it is a transitive dependency. We further decompose this relation to:

Property (propertyNo, pAddress)

Staff (staffNo, sName)

PropertyInspection (FK propertyNo, iDate, iTime, comments, FK staffNo)

Break – 5 mins

Exercises continued:

4. The following Report table is used by a publishing house to keep track of the editing and design of books by a number of authors:

report_no	editor	dept_no	dept_name	dept_addr	author_id	auth_name	auth_addr
4216	woolf	15	design	argus1	53	mantel	cs-tor
4216	woolf	15	design	argus1	44	bolton	mathrev
4216	woolf	15	design	argus1	71	koenig	mathrev
5789	koenig	27	analysis	argus2	26	fry	folkstone
5789	koenig	27	analysis	argus2	38	umar	prise
5789	koenig	27	analysis	argus2	71	koenig	mathrev

By looking at the data, we see that functional dependencies in the Report table are the following:

report_no → editor, dept_no
dept_no → dept_name, dept_addr
author_id → auth_name, author_addr

The candidate key for this relation is (report_no, author_id) since we need these two attributes to uniquely identify each record. Thus we have:

Report (report_no, editor, dept_no, dept_name, dept_addr, author_id, auth_name, auth_addr)

a. Is the Report table in 2NF? If not, put the table in 2NF.

It is in 1NF since there are no repeating values within a cell. However, this relation is not in 2NF, since every non-key attribute is not fully dependent on the entire primary key (e.g. auth_name depends only on auth_id).

Hence we need to normalize and make relation attributes be dependent on the entire key.

Author (author_id, auth_name, auth_addr)

Report (report_no, dept_no, dept_name, dept_addr, editor)

ReportAuthor (report_no [FK], author_id [FK])

Note that we got Report and ReportAuthor in two steps. After splitting off the author information from Report into its own Author relation, Report now looks like this:

Report (report_no, dept_no, dept_name, dept_addr, author_id [FK], editor)

This relation is still not in 2NF, since dept_no, dept_name, dept_addr and editor depend only on report_no. Thus we need to split this relation again into Report and ReportAuthor (as above).

b. Are there any insert, update or delete anomalies with these 2NF relations?

Yes there are, because we still have the transitive dependency where:

dept_no → dept_name, dept_addr

If we delete a record from the report table we might delete the information about a department. We can't insert a new department until we have a report for it, etc.

To solve this we will need to normalize to 3NF.

Author (author_id, auth_name, auth_addr)

Department (dept_no, dept_name, dept_addr)

Report (report_no, dept_no (FK), editor)

ReportAuthor (report_no (FK), author_id (FK))

It may help to draw the final relations with the data from the original table:

Report

report_no	**editor**	**dept_no**
4216	woolf	15
5789	koenig	27

Department

dept_no	**dept_name**	**dept_addr**
15	design	argus1
27	analysis	argus2

Author

author_id	**auth_name**	**auth_addr**
53	mantel	cs_tor
44	bolton	mathrev
71	koenig	mathrev
26	fry	folkstone
38	umar	prise

ReportAuthor

report_no	**author_id**
4216	53
4216	44
4216	71
5789	26
5789	38
5789	71

Now anomalies are not present any more.

Continued over page…

5. Consider the following relation:

   Class (courseNumber, roomNumber, instructorName, studentNumber,
   workshopNumber, grade, tutor)

   The following functional dependencies hold for this relation:

   workshopNumber → tutor
   studentNumber, courseNumber → grade, workshopNumber
   courseNumber → roomNumber, instructorName

   Normalise this relation into 3NF.

   The relation is already in 1NF. We need to see whether we have a violation of 2NF.

   Since courseNumber determines roomNumber, and instructorName, this is a partial dependency that needs to be resolved. Thus, we decompose Class into:

   Course (courseNumber, roomNumber, instructorName)

   Class (courseNumber (FK), studentNumber, workshopNumber, grade, tutor)

   Now the relations are in 2NF, but Class is not in 3NF, since we have a transitive dependency where workshopNumber determines tutor.

   Thus, we need to further decompose Class into:

   Workshop (workshopNumber, tutor)

   Class (courseNumber (FK), studentNumber, workshopNumber (FK), grade)

   All three relations are now in 3NF, since they are in 2NF and there are no transitive dependencies.

END OF TUTORIAL