Study Unit 1
Database Design with Normalisation

Learning Outcomes

By the end of this unit, you should be able to:

1. Appraise a relation to determine whether modification anomalies exist
2. Differentiate functional dependency and multivalued dependency
3. Distinguish the various types of keys: candidate, primary, foreign and surrogate keys
4. Identify the normal forms for relations
5. Apply normalisation to place relations in BCNF and 4NF
6. Discuss table structures according to normalisation principles

Overview

In this study unit, we look into designing databases given existing data. The premise is data must be placed into the correct relations to avoid modification anomalies.

A relation can be placed in various normal forms. These normal forms have been proposed to solve the problem of modification anomalies arising from various sources. Before discussing the normal forms, the necessary terminologies in Relational Model are first explained.

Next, we look at a procedure introduced in the course text, for normalising a relation. We consider two main sources of modification anomalies: functional dependency and multivalued dependency. A procedure identifies which relation the data that should go into. The result of this procedure is that the relations produced are in Boyce Codd Normal Form (BCNF) and in Fourth Normal Form (4NF), thus removing modification anomalies arising from functional dependencies and multivalued dependencies.

This study unit covers chapter 3 of the course text. It is estimated that the student will spend about 5 hours to read the course text chapter 3 in conjunction with the study notes, to work out the activities, and self-assessment questions in the study notes included at suitable junctures to test understanding of the contents covered. It is advisable to use the study notes to guide the reading of the chapter in the textbook, attempt the questions, and then check the text and/or other sources for the accuracy and completeness of your answers.

Chapter 1 Importance of Database Design

1.1 Motivation

Applications with databases are ubiquitous; it is hard to find an application that is not connected to a database. The design of a database has consequence on data redundancy which then affects performance, data consistency and data integrity.

In relational databases, a good design means placing the data in the right relation (or, a table when implemented on a database). Deciding whether a relation is right is based on the dependency relationships amongst the data.

A definition for relation is given in section 1.2.1.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 171-172.

The course text uses the terms:

- table to mean relation,
- column to mean attribute and
- row to mean tuple.

This study unit uses the relational terms instead. That is, relation, attribute and tuple are used instead of table, column and row. We will use the terms table, column and row in Study Unit 3 when we discuss the implemented relations on a database.

1.1.1 Performance

Poorly designed database impacts how queries are processed and how data is inserted, updated and deleted.

For example, if two pieces of data about a single occurrence are placed in two different relations, then a query or an insert, an update or a delete operation will take up more processing time as there are two data accesses per operation, instead of one.

1.1.2 Data Consistency and Integrity

Poorly designed database contains unnecessary data redundancy. It is hard to implement business rules which include policies for insert, update and delete operations on poorly designed database, to ensure data consistency and integrity.

For example, a student can have multiple email addresses. In a relational database, as the different email addresses of a student must be recorded in different tuples, then if the student data is recorded in a single relation, non-repeating attributes such as the student id and home address will be repeated, as shown in Table 1.1.

Student Id	Home Address	Email Address
S1234	12 Nelson Road	eddie@mail.com
S1234	12 Nelson Road	eddie@gmail.com
S1234	12 Nelson Road	eddie@outlook.com
S2341	10 Angson Drive	john@gmail.com

Table 1.1 Student(student id, home address, email address)

Multiple tuples of same data, such as student id and address, result in data redundancy. This increases the overhead of maintaining data consistency and integrity, causing modification anomalies when data is inserted, deleted or updated.

1.1.3 Data Redundancy and Modification Anomalies

Data redundancy can result in modification anomalies.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 180-181.

Consider the example relation in Table 1.1 in which a student has one home address but multiple email addresses.

- Update Anomaly

  Update anomaly occurs when updating an attribute in one tuple causes data inconsistency with other tuples in the same relation.

  Suppose, student S1234 has a change in home address. If only one tuple is updated, then there is data inconsistency. To ensure data consistency, three tuples instead of one, must be updated.

- Insertion Anomaly

  Insert anomaly occurs when inserting a new tuple will require other data to be inserted as well.

  Suppose student S2341 has a new email address. Inserting the new email address will require home address to be inserted as well.

- Deletion Anomaly

Deletion anomaly occurs when deleting a tuple causes other data to be deleted as well.

Suppose student S2341 deleted his only email account, and this delete must be reflected on the data. The only tuple for S2341 will be deleted and so, the home address for student S2341 will be lost.

Activity 1

Reproduced from Question 3.32 of the course text.

Illustrate deletion, modification, and insertion anomalies on the STUDENT_ACTIVITY relation.

StudentID	StudentName	Activity	ActivityFee	AmounttPaid
100	Jones	Golf	65.00	65.00
100	Jones	Skiing	200.00	0.00
200	Davis	Skiing	200.00	0.00
200	Davis	Swimming	50.00	50.00
300	Garrett	Skiing	200.00	100.00
300	Garrett	Swimming	50.00	50.00
400	Jones	Golf	65.00	65.00
400	Jones	Swimming	50.00	50.00

1.2 Relational Model Terminology

One way to improve the structure of a relational database is to apply normalisation on the relations. A relation can be normalised to varying degree or, in other words, to different normal forms.

Different normal forms result in different sets of relations. Each normal form is able to eliminate modification anomalies from specific sources such as functional dependencies and multivalued dependencies, the two sources being the focus of this study unit. Functional dependencies and multivalued dependencies are discussed in Section 1.2.2 and Section 1.2.3 respectively.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 15*. Pearson, 166-168.

1.2.1 Relation

A relation is a table that has these properties:

1. The name of a relation is unique within a database.
2. The name of an attribute is unique within a relation.
3. Each tuple is unique in a relation.
4. The values for each attribute are from one domain
5. Each value for a attribute is atomic.
6. The order of the tuples in a relation is not significant.
7. The order of the attributes in a relation is not significant.

1.2.2 Functional Dependency

A function dependency A –› B describes the relationship between the attributes (or columns), A and B, of a relation.

- A is the determinant that determines B.
- For each value of the determinant, A, there is only one value for the determined attribute, B, as shown in Figure 1.1.

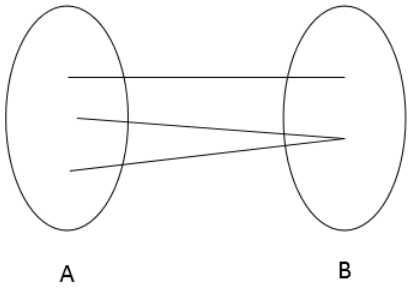


Figure 1.1 Functional dependency A –› B

- In general, A and B can each be a set of attributes.
  - A single attribute can be a determinant of a set of attributes.

    For example, each student id is related to one home address and one date of birth.

    student id –› home address, date of birth

    The above functional dependency is actually made up of two functional dependencies:

    student id –› home address

    student id –› date of birth

    However, it is easier to apply normalization when we combine the two functional dependencies as one: student id –› home address, date of birth

- Several attributes can be a determinant of an attribute.

  For example, student id, module code for a module that the student is enrolled in a particular year and semester determine the tutorial group the student is assigned to.

  student id, module code, year, semester –› tutorial group

Consider the functional dependency,

student id –› home address, date of birth

Each value of the determinant, student id has only one value for each of the attributes, home address and date of birth, it determines. For example, for a given value of student id S1234, only one home address, 12 Nelson Road , is associated to it.

However, note that functional dependency does not preclude that both home addresses cannot be the same as shown in Figure 1.1. In fact, S1234 and S2341 can be student ids of two siblings, and so, it is possible that both home addresses are the same. Likewise, the same can be said about date of birth; two different student ids may be associated with the same date of birth.

Student id does not determine email address as one student can have many email addresses. Whether or not an email address can determine a student id depends on the business domain.

- An email address may also not determine the student id if email addresses can be shared. For example, one of the email addresses of a student may belong to the student's parent, and that email is shared among students who are siblings.

- An email address can determine the student id if email addresses are not shared.

A consequent of attributes in a functional dependency not placed in the correct relation is modification anomaly. Modification anomaly occurs because the determinant and the determined attribute appear in multiple tuples, and each tuple must show data consistency.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 168-177.

Activity 2

Reproduced from Question 3.13 of the course text.

Explain the meaning of the expression:

(FirstName, LastName) –›Phone

1.2.3 Multivalued Dependency

A multivalued dependency A –›–› B describes the relationship between the attributes, A and B, of a relation.

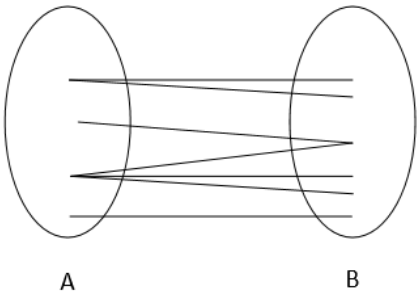


Figure 1.2 Multivalued dependency A –›–› B

- A is the determinant that determines B.

- For each value of the determinant, A, there is one set of values for the determined attribute, B as shown in Figure 1.2.
- In general, A and B can each be a set of attributes.

  For example,

  - Student id multivalued-determines a set of email addresses.

    student id –›–› email address

  - student id, module code, year, semester determine a set of assignment marks that a student obtains for the set of assignments for a module in a particular year and semester.

    student id, module code, year, semester –›–› assignment, mark

  Unlike functional dependencies, this multivalued dependency cannot be split into two multivalued dependencies as they are not the same. That is,

  student id, module code, year, semester –›–› assignment, mark

  is not the same as :

  student id, module code, year, semester –›–› assignment

  student id, module code, year, semester –›–› mark

Like functional dependency, multivalued dependency also imposes that a determinant can have only one set of values for each attribute it determines in the whole database. Thus, if a value of a determinant appear multiple times, then the set of values for the determined attribute must also show data consistency.

Activity 3

Reproduced from Question 3.51 of the course text.

How does a multivalued dependency differ from a functional dependency?

1.2.4 Keys of a Relation

To avoid modification anomalies due to functional dependencies and multivalued dependencies, all determinants must be a key in their relations, that is, the values of the determinants must be unique in their relations.

- Candidate key

  A determinant that determines all other attributes in a relation R1, is a candidate key of the relation R1.

  There may be more than one determinant that determines all other attributes in a relation. Each of these determinants is a candidate key.

- Composite key

  A candidate key that is made up of several attributes is called a composite key.

- Primary key

  One of the candidate keys is chosen as the official key of the relation. The chosen key is the primary key.

  A primary key is chosen based on whether it is short, and preferably, its values are numeric and do not change.

- Surrogate key

  When none of the candidate keys is suitable, an artificial attribute may be introduced into the relation to be its primary key.

  The DBMS auto-generates values for the surrogate key.

- Foreign key

  A primary key of a relation R1 when placed also in another relation R2, is a foreign key in relation R2.

  Foreign keys are used to associate or link two relations when one relation has been split to two relations to reduce data duplication.

However, foreign keys are themselves data duplication, but are a necessary form of data duplication. The foreign keys help identify related tuples in the various relations.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 177-180.

Activity 4

Reproduced from Question 3.24 of the course text.

What is a primary key? Explain the significance of the entity integrity constraint to a primary key.

1.3 Normal Forms

The aim of normalisation is to eliminate modification anomalies caused by data redundancy. Relations in BCNF (Boyce-Codd Normal form) and 4NF (Fourth Normal Form) do not suffer from modification anomalies arising from functional dependency and from multivalued dependency respectively.

We will not consider other normal forms such as 5NF and DKNF (Domain Key Normal form) in this study unit. 5NF and DKNF deal with data constraints to completely eliminate all forms of modification anomalies. Data objects such as triggers and stored procedures can be used to encode these data constraints. Triggers and stored procedures are covered in Study Unit 3.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 181-187.

Various normal forms from 2NF have been introduced to eliminate modification anomalies.

1.3.1 1NF

A table of data must be first normal form for it to be a relation, that is, for it to be in a relation database. A relation in 1NF must have all seven properties listed in section 1.2.1.

A relation in 1NF has no other restrictions on the data except for the seven properties. This, a relation in 1NF allows data duplication for some attributes, due to functional dependencies and to multivalued dependencies.

Consider the relation in Table 1.2. Observe that the property of having unique tuples does not prevent data duplication for some attributes, such as those in a functional dependency. Thus, a relation in 1NF has modification anomalies.

Student Id	Date of birth	Home address	Email address
S1234	1/1/1990	12 Nelson Road	eddie@mail.com
S1234	1/1/1990	12 Nelson Road	eddie@gmail.com
S1234	1/1/1990	12 Nelson Road	eddie@outlook.com

Table 1.2 Student(student id, date of birth, home address, email address)

1.3.2 2NF

In an attempt to overcome the data duplication problem in 1NF, the second normal form or 2NF is introduced.

A relation in 2NF is also in 1NF and moreover, all its non-key attributes are determined by the entire key. This means that the non-key attributes are not partially functionally dependent but rather fully functionally dependent on the key of the relation.

Assume that there is only this functional dependency for the relation in Table 1.2, in Section 1.3.1:

student id –› home address, date of birth

Figure 1.3 shows the dependency graph for this relation.

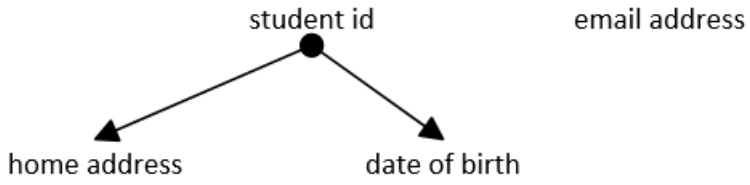


Figure 1.3 Dependency graph for relation Student

The key of the relation is (student id, email) .

The relation is in 1NF but not in 2NF because both home address and date of birth are partially dependent on the key (student id, email) as student id alone can determine both these attributes.

For the relation Student to be in 2NF, we split the relation into 2 relations. The attributes student id and email go into one relation, and the attributes student id, date of birth and home address go into another relation.

In the first relation as shown in Table 1.3, student id and email address is a composite primary key. In the second relation as shown in Table 1.4, student id is the primary key. Notice that now, both date of birth and home address are fully functionally dependent on the key of the relation, student id.

Notice also that student id appears in both relations. This allows us to identify related tuples in the two relations.

Student Id	Email address
S1234	eddie@mail.com
S1234	eddie@gmail.com
S1234	eddie@outlook.com

Table 1.3 StudentEmail(student id, email address)

Student Id	Date of birth	Home address
S1234	1/1/1990	12 Nelson Road

Table 1.4 StudentDetail(student id, date of birth, home address)

Both relations are now in 2NF.

1.3.3 3NF

A relation in 2NF still suffers from modification anomalies if there are transitive functional dependencies in the relation. Thus, the third normal form or 3NF is introduced.

A relation in 3NF is also in 2NF and moreover, no non-key attributes can be determined by another non-key attribute. 3NF is about removing transitive functional dependency in a relation.

Consider the example in the course text where within a relation with three attributes student id, building and housing fees, there are two functional dependencies:

student id –› building

building –› housing fees

The key of this relation is student id because student id –› building, housing fees. However, the relation is in 2NF but is not in 3NF as Student id –› housing fees is a transitive dependency.

Having a transitive dependency gives rise to modification anomalies. For example, if the housing fees for a building is changed, then the tuples for students in the same building will need to have the housing fees updated.

For the relation to be in 3NF, we remove the transitive dependency and put student id and housing in one relation, and housing and housing fees in another relation.

1.3.4 BCNF

A relation in 3NF still suffers from modification anomalies if there are overlapping functional dependencies in the relation. Thus, the Boyce Codd normal form or BCNF is introduced.

A relation in BCNF is also in 3NF and moreover, every determinant in the relation must be a key. BCNF is about removing overlapping functional dependency in a relation.

Consider the example in the course text where within a relation with three attributes student id, major and advisor, there are three functional dependencies:

student id, major –› advisor

studentid, advisor –› major

advisor –› major

The relation (student id, major, advisor) is in 3NF because there is no transitive dependency, and there is no non-key attribute that is not dependent on the entire key.

However, the relation is not in BCNF as there is a determinant advisor that is not a key of the relation. Advisor cannot determine student id.

Having a overlapping functional dependency gives rise to modification anomalies. For example, if a student drops one of his majors, then a tuple for that student must be deleted. If the advisor for the tuple to be deleted has only one student, then deleting the tuple will cause the data about the advisor and his major to be lost.

For the relation to be in BCNF, we remove the overlapping functional dependency and put advisor and major in one relation with advisor as key. The remaining attributes student id and advisor are placed in another relation with both attributes as composite key as they do not determine each other.

1.3.5 4NF

A relation in BCNF still suffers from modification anomalies if there are multivalued dependencies in the relation. Thus, the fourth normal form or 4NF is introduced.

A relation in 4NF is also in BCNF and moreover, the attributes of the relation are only those attributes participating in a multivalued dependency.

For example, the relation Student has these four attributes: student id, date of birth, home address and email address. The dependencies of this relation are:

student id –› date of birth, home address

student id –›–› email address

For the relation to be in 4NF, the attributes of the multivalued dependency student id –›–› email address are placed in a separate table. The remaining attributes including student id are placed into another relation.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 196-199.

Activity 5

Reproduced from Question 3.35 of the course text.

Which normal forms are concerned with functional dependencies?

Chapter 2 Normalisation

2.1 When to Apply Normalisation

When data is not modified, that is, there is no insert, delete and update operations on the data, modification anomaly does not result. There is no need for normalisation as normalization causes relations to be split, increasing the overhead of data retrieval.

Normalisation is required only if there is possibility of data modification. We apply can nomalisation only after the attributes of relations and the relationships amongst them have been identified.

The attributes are identified in three different sources, giving three approaches to database design:

- Data already exists in some form
- Data requirements for a new application, modelled in entity-relationship diagram. Entity relationship is covered in Study Unit 2.
- An existing database that needs to be redesigned

2.2 How to Apply Normalisation

One way to normalise relations is to put them first into 1NF, then into 2NF, then into 3NF, then into BCNF and finally into 4NF, progressively. In each transformation, distribute the attributes into relations to ensure the resulting relations meet the conditions of the normal forms.

Another simpler way is to apply the procedure as described in the course text, and place the relations straight into BCNF and 4NF. The steps of the procedure are modified from the course text version, to incorporate multivalued dependencies.

In addition, drawing a dependency graph is most helpful to identify the candidate keys as well as to help determine which dependencies should be processed first.

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 187.

The normalisation steps for the procedure from the course text version, modified as follow:

1. Identify the functional dependencies and multivalued dependencies.
2. Draw the dependency graph and identify all possible keys, that is, the candidate keys.
3. Highlight subsumed multivalued dependencies.
4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.
    a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.
    b. Make a copy of the determinant in the original relation and place the copy into the new relation.
    c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.
    d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.
    e. Create a referential integrity constraint between the original relation and the new relation.
5. Repeat step 4 until every determinant of every relation is a candidate key.

Activity 6

Reproduced from Question 3.44 of the course text.

What is a referential integrity constraint? Define the term, and give an example of its use. Are null values allowed in foreign key columns with a referential integrity constrain? How does the referential integrity constraint contribute to database integrity?

2.3 An Example on Normalisation

Read

Kroenke, D and D. Auer. (2016). *Database Processing: Fundamentals, Design and Implementation Edition 14*. Pearson, 189-196.

Scenario (Modified from ICT 321 Past TMA)

A law firm engages several solicitors and provides services for specific case types (e.g., intellectual property enforcement, copyright and trademark, risk management and litigation).

Each solicitor works on one or more cases. A simple case requires one solicitor but complex ones are handled by a team. Each case is for one client. On any day the case is being serviced, there may be several services provided (e.g., filing suit, court representation and documentation). The cost of each service depends on the service performed on the day of service and on the case .

In some cases, a solicitor may act for the client against one or more parties (e.g., several companies may be sued in a case of infringement of intellectual property).

Note that a party may be involved in a few cases (e.g., a company may infringe the IPs of several clients represented by the law firm).

The **Case** relation:

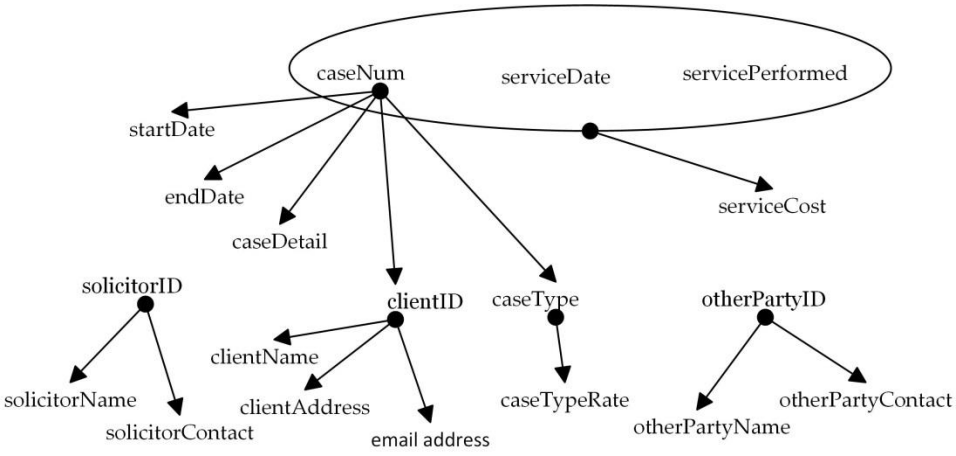
Case(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID, solicitorName, solicitorContact, clientID, clientName, clientAddress, clientContact, serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact)

Apply the nomalisation procedure:

1. Identify the functional dependencies and multivalued dependencies.

   solicitorID –› solicitorName, solicitorContact
   clientID –› clientName, clientAddress, clientContact
   otherPartyID –› otherPartyName, otherPartyContact
   caseType –› caseTypeRate
   caseNum –› startDate, endDate, caseDetail, caseType, clientID
   caseNum , serviceDate , servicePerformed –› serviceCost
   caseNum –›–› solicitorID
   caseNum –›–› otherPartyID
   caseNum –›–› serviceDate, servicePerformed serviceCost

2. Draw the dependency graph and identify all possible keys, that is, the candidate keys.



   Candidate key:

   (caseNum, serviceDate, servicePerformed, solicitorID, otherPartyID)

3. Remove multivalued dependencies that are subsumed by functional dependencies. A multivalued dependency is subsumed by some functional dependencies if all tuples for the relation for multivalued dependency are in some relation(s) for the functional dependencies.

   caseNum –›–› serviceDate, servicePerformed, serviceCost is subsumed by caseNum , serviceDate , servicePerformed –› serviceCost

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

   In this case, we may handle any of these functional dependencies:

   solicitorID –› solicitorName, solicitorContact
   clientID –› clientName, clientAddress, clientContact
   otherPartyID –› otherPartyName, otherPartyContact
   caseType –› caseTypeRate

   Consider solicitorID –› solicitorName, solicitorContact

   a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.

      Solicitor(solicitorID, solicitorName, solicitorContact)

   b. Make a copy of the determinant in the original relation and place the copy into the new relation.

      Case2(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID, ~~solicitorName, solicitorContact~~, clientID, clientName, clientAddress, clientContact, serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact)

   c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

      Solicitor(solicitorID(PK), solicitorName, solicitorContact)

      Case2(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID, clientName, clientAddress,

clientContact, serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact)

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

e. Create a referential integrity constraint between the original relation and the new relation.

Case2. solicitorID must exist in Solicitor. solicitorID

5. Repeat step 4 until every determinant of every relation is a candidate key.

Every determinant in Solicitor(solicitorID(PK), solicitorName, solicitorContact) is a candidate key and so, Solicitor is in BCNF.

However, not every every determinant of Case2(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID, clientName, clientAddress, clientContact, serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

In this case, we may handle any of these functional dependencies:

clientID –› clientName, clientAddress, clientContact
otherPartyID –› otherPartyName, otherPartyContact
caseType –› caseTypeRate

We can also handle the multivalued dependency:
caseNum –›–› solicitorID

Consider clientID –› clientName, clientAddress, clientContact

a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.
Client(clientID, clientName, clientAddress, clientContact)

b. Make a copy of the determinant in the original relation and place the copy into the new relation.

Case3(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID~~, clientName, clientAddress, clientContact~~, serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact)

c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

Client(clientID(PK), clientName, clientAddress, clientContact)
Case3(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact)

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

e. Create a referential integrity constraint between the original relation and the new relation.

Case3. clientID must exist in Client. clientID

Every determinant in Client(clientID(PK), clientName, clientAddress, clientContact) is a candidate key and so, Solicitor is in BCNF.

However, not every every determinant of Case3(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID, otherPartyName, otherPartyContact) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

In this case, we may handle any of these functional dependencies:

otherPartyID –› otherPartyName, otherPartyContact
caseType –› caseTypeRate

We can also handle the multivalued dependency:

caseNum –›–› solicitorID

Consider otherPartyID –› otherPartyName, otherPartyContact

a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.
OtherParty(otherPartyID , otherPartyName, otherPartyContact)

b. Make a copy of the determinant in the original relation and place the copy into the new relation.

Case4(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID, ~~otherPartyName, otherPartyContact~~)

c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

OtherParty(otherPartyID (PK), otherPartyName, otherPartyContact)
Case4(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK))

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

e. Create a referential integrity constraint between the original relation and the new relation.

Case4. otherPartyID must exist in OtherParty. otherPartyID

Every determinant in OtherParty(otherPartyID (PK), otherPartyName, otherPartyContact) is a candidate key and so, Solicitor is in BCNF.

However, not every every determinant of Case4(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, solicitorID (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK)) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

   In this case, we may handle this functional dependency :

   caseType –› caseTypeRate

   We can also handle the multivalued dependencies:
   caseNum –›–› solicitorID
   caseNum –›–› otherPartyID

   Consider caseNum –›–› solicitorID

   a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.
      CaseSolicitor(caseNum, solicitorID)

   b. Make a copy of the determinant in the original relation and place the copy into the new relation.

      Case5(caseNum, startDate, endDate, caseDetail, caseType, caseTypeRate, ~~solicitorID (FK),~~ clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK))

   c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

   d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

      CaseSolicitor(caseNum(FK), solicitorID(FK))
      Case5(caseNum(PK), startDate, endDate, caseDetail, caseType, caseTypeRate, clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK))

   e. Create a referential integrity constraint between the original relation and the new relation.

CaseSolicitor.caseNum must exist in Case5. caseNum and CaseSolicitor. solicitorID must exist in Solicitor. solicitorID

Every determinant in CaseSolicitor(caseNum(FK), solicitorID(FK)) is a candidate key and so, Solicitor is in BCNF. The only attributes in CaseSolicitor are also those in the multivaued dependency and so CaseSolicitor is in 4NF.

However, not every every determinant of Case5(caseNum(PK), startDate, endDate, caseDetail, caseType, caseTypeRate, clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK)) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

   In this case, we may handle this functional dependency :

   caseType –› caseTypeRate

   We can also handle the multivalued dependencies:
   caseNum –›–› otherPartyID

   Consider caseType –› caseTypeRate

   a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.

      CaseType(caseType, caseTypeRate)

   b. Make a copy of the determinant in the original relation and place the copy into the new relation.

      Case6(caseNum, startDate, endDate, caseDetail, caseType, ~~caseTypeRate,~~ clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK))

   c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

      CaseType(caseType(PK), caseTypeRate)

Case6(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK))

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

e. Create a referential integrity constraint between the original relation and the new relation.

Case6.caseType must exist in CaseType.caseType

Every determinant in CaseType(caseType(PK), caseTypeRate) is a candidate key and so, CaseType is in BCNF.

However, not every every determinant of Case6(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, otherPartyID(FK)) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

In this case, we may handle this functional dependency :

caseNum –› startDate, endDate, caseDetail, caseType, clientID
caseNum , serviceDate , servicePerformed –› serviceCost

We can also handle the multivalued dependencies:
caseNum –›–› otherPartyID

Consider caseNum –›–› otherPartyID

a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.
CaseOtherParty(caseNum, otherPartyID)

b. Make a copy of the determinant in the original relation and place the copy into the new relation.

Case7(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK), serviceDate, servicePerformed, serviceCost, ~~otherPartyID(FK)~~)

c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

CaseOtherParty(caseNum(PK), otherPartyID (FK))
Case7(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK), serviceDate, servicePerformed, serviceCost)

e. Create a referential integrity constraint between the original relation and the new relation.

CaseOtherParty.caseNum must exist in Case7.caseNum and CaseOtherParty. otherPartyID must exist in OtherParty. otherPartyID

Every determinant in CaseOtherParty(caseNum(PK), otherPartyID (FK))is a candidate key and so, CaseOtherParty is in BCNF. The only attributes in CaseOtherParty are also those in the multivalued dependency and so CaseOtherParty is in 4NF.

However, not every every determinant of Case7(caseNum(PK), startDate, endDate, caseDetail, caseType(FK), clientID(FK), serviceDate, servicePerformed, serviceCost) is a candidate key.

Repeat step 4.

4. If there are more than one functional dependencies or un-subsumed multivalued dependencies, choose to handle the dependencies that are at or are nearer the leaves of the dependency graph first.

In this case, we may handle this functional dependency :

caseNum –› startDate, endDate, caseDetail, caseType, clientID
caseNum , serviceDate , servicePerformed –› serviceCost

Consider caseNum –› startDate, endDate, caseDetail, caseType, clientID

a. Move the attributes of dependency, that is, the attributes that are determined into a new relation.

Case(caseNum, startDate, endDate, caseDetail, caseType, clientID)

b. Make a copy of the determinant in the original relation and place the copy into the new relation.

Case8(caseNum(PK), ~~startDate, endDate, caseDetail, caseType(FK), clientID(FK)~~, serviceDate, servicePerformed, serviceCost)

c. If the dependency is a function dependency, make the copy of the determinant a primary key in the new relation, and make the determinant in the original relation a foreign key.

Case(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK))

Case8(caseNum(FK), serviceDate, servicePerformed, serviceCost)

d. If the dependency is a multivalued dependency, make the copy of the determinant a foreign key in the new relation, and make the determinant in the original relation a primary key.

e. Create a referential integrity constraint between the original relation and the new relation.

Case8. caseNum must exist in Case. caseNum

Every determinant in Case(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK)) is a candidate key and so, Case is in BCNF.

Moreover, every every determinant of Case8(caseNum(FK), serviceDate, servicePerformed, serviceCost) is a candidate key.

Final Set of relations and referential integrity constraints:

Solicitor(solicitorID(PK), solicitorName, solicitorContact)

Client(clientID(PK), clientName, clientAddress, clientContact)

OtherParty(otherPartyID (PK), otherPartyName, otherPartyContact)

CaseType(caseType(PK), caseTypeRate)

Case(caseNum(PK), startDate, endDate, caseDetail, caseType (FK), clientID(FK))
where Case.caseType must exist in CaseType.caseType and Case. clientID must exist in Client.clientID

CaseSolicitor(caseNum(PK) (FK), solicitorID(PK) (FK))
where CaseSolicitor. caseNum must exist in Case.caseNum and CaseSolicitor. solicitorID must exist in Solicitor.solicitorID

CaseOtherParty(caseNum(PK), otherPartyID (FK))
where CaseOtherParty.caseNum must exist in Case.caseNum and CaseOtherParty. otherPartyID must exist in OtherParty. otherPartyID

Case8(caseNum(PK)(FK), serviceDate(PK), servicePerformed(PK), serviceCost)
where Case8. caseNum must exist in Case. caseNum

Activity 7

Reproduced from Question 3.58 of the course text.

Consider the relation:

STAFF_MEETING (EmployeeName, ProjectName, Date)

The tuples of this relation record the fact that an employee from a particular project attended a meeting on the given date. Assume that a project meets at most once per day. Also, assume that only one employee represents a given project, but that employees can be assigned to multiple projects.

a. State the functional dependencies.

b. Transform this relation into one or more relations in BCNF. State the primary keys, candidate keys, foreign keys, and referential integrity constraints.

Activity 8

Reproduced from Question 3.59 of the course text.

Consider the relation:

STUDENT (StudentNumber, StudentName, Dorm, RoomType, DormCost, Club, ClubCost, Sibling, Nickname)

Assume that students pay different dorm costs, depending on the type of room they have, but that all members of a club pay the same cost. Assume that students can have multiple nicknames.

a. State any multivalued dependencies.

b. State the functional dependencies.

c. Transform this relation into two or more relations such that each relation is in BCNF and in 4NF. State the primary keys, candidate keys, foreign keys, and referential integrity constraints.

Summary

In this study unit, we started with existing data. The intention is to place the data into the correct relations to avoid modification anomalies. Modification anomalies are caused by data duplication. The three forms of modification anomalies are insertion, deletion and update anomalies.

The key to placing data in relation is handling the functional dependencies and multivalued dependencies. Handling the dependencies to different extent gives rise to various normal forms.

The course text introduces q normalization procedure to put each relation into Boyce Codd Normal Form (BCNF) and into Fourth Normal Form (4NF). The procedure places every functional dependency and multivalued dependency in its own relation. Referential integrity constraints then associate two or more relations.

References

Book

Author(s)	Year	Book Title	Edition	Publisher
Kroenke, D., & Auer, D. J.	2016	*Database Processing – Fundamentals, Design, and Implementation*	14	Pearson

Quiz

1. Which statement about relations is true?

   *a. All relations are tables, but not all tables are relations.

   b. A relation that has special restrictions on it is a table.

   c. The attributes of the relation hold the same value.

   d. The tuples of a relation may be identical.

2. Which statement about functional dependency is true?

   a. If by knowing the value of A we can find the value of B, then we would say that A is functionally dependent on B.

   *b. A functional dependency is a relationship between attributes such that if we know the value of one attribute, we can determine the value of the other attribute.

   c. In functional dependencies, the attribute whose value is unknown is referred to as the determinant.

   d. Given the functional dependencies A –› B and C –› B, then it is true that A –› C.

3. Which statement about keys is false?

   a. A surrogate key is an artificial column that is added to a relation to be its primary key.

   b. A candidate key is one of a group of keys that may serve as the primary key in a relation.

   *c. It is possible to have a relation that does not have a key.

   d. A foreign key is one or more columns in one relation that also is the primary key in another table.

4. Having to enter facts about two entities when we want to enter facts about only one is an example of _____.

   *a. insertion anomaly

b. update anomaly

c. deletion anomaly

d. normalization anomaly

5. A relation is in Boyce-Codd normal form (BCNF) if and only if it is in 3NF and _____.

a. all non-key attributes are determined by the entire primary key

b. there are no non-key attributes determined by another non-key attribute

c. every attribute is a candidate key

*d. every determinant is a candidate key

Formative Assessment

1. Which property makes a table a relation?

   a. No columns with missing value

   Incorrect. A relation can have null values. Refer to textbook pages 169-171.

   b. Every determinant is a candidate key

   Incorrect. The requirement that every determinant is a candidate key makes the table in BCNF. Tables need not be in BCNF to be a relation as there are several forms of normalization. Refer to textbook page 187.

   c. No multi-valued dependency

   Incorrect. If a table contains multi-valued dependency, then it is not in 4NF but that table can be a relation. Refer to textbook page 196.

   d. No column containing more than one values.

   Correct! Every column should have only one value for a table to be a relation. Refer to textbook page 169.

2. What can be concluded about a table with three columns A, B and C, and with A –› B, C as the only functional dependency?

   a. (B, C) –› A

   Incorrect. Functional dependency is not commutative. Refer to textbook pages 172-174

   b. There is at most one row for each value of A

   Correct! A determines every other column in the table. Refer to textbook page 177

   c. There is at most one row for each value of B

   Incorrect. B is not a determinant. Refer to textbook page 172.

   d. There is at most one row for each combination of values of B and C

   Incorrect. (B, C) is not a determinant. Refer to textbook page 172.

3. What is the term to describe the anomaly that arises when the removal of facts about one entity results in the unintentional loss of data about another entity?

   a. Deletion

Correct! Deletion anomaly occurs when the delete causes more data than intended to be deleted. Refer to textbook page 180.

b. Insertion

Incorrect. Insertion anomaly occurs when we are required to enter more facts than intended. Refer to textbook page 180.

c. Update

Incorrect. Update anomaly occurs when some rows that must be updated are not. Refer to textbook page 180.

d. Modification

Incorrect. This general term refers to the three different types of anomalies: deletion, insertion and update. Refer to textbook page 180.

4. Given the table

CustomerOrder (CustNo, name, address, contact, OrderNo, OrderDate, ItemNo, ItemQty, ProductNo, description, price)

Given the functional dependencies for this table:

CustNo –› name, address, contact
OrderNo –› CustNo, OrderDate
OrderNo, ItemNo –› ProductNo, ItemQty
ProductNo –› description, price

Refer to the procedure in the textbook, page 187, Figure 3.19. Which table is not in the final set of tables derived?

a. Product(ProductNo , description, price)

Incorrect. Since ProductNo –› description, price and ProductNo is not a key for the table CustomerOrder, the dependency must be moved to this separate table, Product. Refer to textbook pages 188-195.

b. OrderDetails(OrderNo, ItemNo, ProductNo, ItemQty)

Incorrect. Since OrderNo, ItemNo –› ProductNo, ItemQty and OrderNo is not a key for the table CustomerOrder, the dependency must be moved to this separate table, OrderDetails. Refer to textbook pages 188-196.

c. Order(OrderNo, CustNo, OrderDate)

Incorrect. Since OrderNo –› CustNo, OrderDate and OrderNo is not a key for the table CustomerOrder, the dependency must be moved to this separate table, Order. Refer to textbook pages 188-195.

d. OrderDetails(OrderNo, ItemNo, CustNo, OrderDate, ProductNo, ItemQty)

Correct! Since OrderNo –› CustNo, OrderDate and OrderNo, ItemNo –› ProductNo, ItemQty, this table is not in BCNF yet and should not be in the final set of tables. Refer to textbook pages 188-195.

5. Given the table

CustomerOrder (CustNo, name, address, contact, OrderNo, OrderDate, OrderStatus)

The functional dependencies

CustNo –› name, address, contact
OrderNo –› CustNo, OrderDate, OrderStatus
The multivalued dependency
CustNo –› –› OrderNo

Refer to the procedure in the textbook, page 187, Figure 3.19, which table is not in the final set of tables derived?

a. Customer(CustNo, name, address, contact)

Incorrect. Since CustNo –› name, address, contact and CustNo is not a key for the table CustomerOrder, the dependency must be moved to this separate table, Customer. Thus, Customer is in the final set of tables. Refer to textbook pages 188-195.

b. Order(OrderNo, CustNo, OrderDate, OrderStatus)

Incorrect. Since OrderNo –› CustNo, OrderDate, OrderStatus and OrderNo is not a key for the table CustomerOrder, the dependency must be moved to this separate table, Order. Thus, Order is in the final set of tables. Refer to textbook pages 188-195.

c. CustomerOrder(OrderNo, CustNo)

Correct! The MVD CustNo –› –› OrderNo Is subsumed by OrderNo –› CustNo, OrderDate, OrderStatus. Thus, CustomerOrder is not in the final set of tables. Refer to textbook pages 196-199.

d. None of the above

Incorrect. CustomerOrder(OrderNo, CustNo) is a redundant table since the MVD CustNo –› –› OrderNo is subsumed by OrderNo –› CustNo, OrderDate, OrderStatus. Refer to textbook pages 196-199.

Solutions or Suggested Answers

Activity 1

A **deletion anomaly** occurs when needed data is lost while deleting other data. In this table, if we delete both of the Jones (StudentID = 100 and StudentID = 400), we will lose all data about Golf and its associated cost:

StudentID	StudentName	Activity	ActivityFee	AmounttPaid
200	Davis	Skiing	200.00	0.00
200	Davis	Swimming	50.00	50.00
300	Garrett	Skiing	200.00	100.00
300	Garrett	Swimming	50.00	50.00

An **insertion anomaly** occurs when we cannot insert needed data for one entity until there is matching data for a second entity present in the table. In this table, if we want to add the Activity of Running with a cost of $25.00, we also need associated data for at least one student:

StudentID	StudentName	Activity	ActivityFee	AmounttPaid
100	Jones	Golf	65.00	65.00
100	Jones	Skiing	200.00	0.00
200	Davis	Skiing	200.00	0.00
200	Davis	Swimming	50.00	50.00
300	Garrett	Skiing	200.00	100.00
300	Garrett	Swimming	50.00	50.00
400	Jones	Golf	65.00	65.00
400	Jones	Swimming	50.00	50.00
???	???	Running	25.00	???

An **update anomaly** occurs when needed data is inconsistent among rows of the table. In this table, if we change the Activity name of Swimming to Diving but only update one of the three rows where this occurs, we will have an update anomaly:

StudentID	StudentName	Activity	ActivityFee	AmountPaid
100	Jones	Golf	65.00	65.00
100	Jones	Skiing	200.00	0.00
200	Davis	Skiing	200.00	0.00
200	Davis	Diving	50.00	50.00
300	Garrett	Skiing	200.00	100.00
300	Garrett	Swimming	50.00	50.00
400	Jones	Golf	65.00	65.00
400	Jones	Swimming	50.00	50.00

Activity 2

Given a person's first name and last name, there is always, unambiguously a single phone number associated to that first name and last name.

Activity 3

In a functional dependency, a determinant determines one value of each of the attributes associated with it. In a multivalued dependency, a determinant determines a set of values of one or more attributes.

Activity 4

The primary key is the candidate key that has been selected from the set of all candidate keys to be the "official" key of a relationship. The entity integrity constraint enforces the fact that every primary key must have a value.

Activity 5

Normal forms 2NF through Boyce-Codd Normal Form (BCNF) are concerned with functional dependencies, particularly modification anomalies from functional dependencies.

Activity 6

A referential integrity constraint is a value constraint on a foreign key that states that no value can be placed in the foreign key unless it already exists as a primary key value in the linked table.

Technically, null values are allowed, but in practice this should never occur. The referential integrity constraint helps ensure database integrity by maintaining valid links between linked tables.

Activity 7

a.

Since there can only be one project meeting for a particular project per day, we have:

(ProjectName, Date) –› EmployeeName

Since there is only one employee assigned to the meetings for each project, we have:

ProjectName –› EmployeeName

b.

STAFF_MEETING functional dependencies:

(ProjectName, Date) –› EmployeeName

ProjectName –› EmployeeName

STAFF_MEETING candidate keys:

(ProjectName, Date)

Procedure:

STAFF_MEETING (EmployeeName, ProjectName, Date) is not in BCNF because ProjectName –› EmployeeName and ProjectName is not a key

Therefore, move ProjectName –› Employee into another relation leaving ProjectName as a foreign key in the original relation, and making ProjectName as a primary ley in the new relation.

STAFF_MEETING_EMPLOYEE (ProjectName, EmployeeName) is in BCNF since ProjectName is the only determinant and is also the key.

STAFF_MEETING_2 (ProjectName, Date) is also in BCNF since (ProjectName, Date) is the only determinant and is also the key.

The tables are now all in BCNF.

FINAL SET of relations:

STAFF_MEETING_2 (ProjectName, Date)

STAFF_MEETING_EMPLOYEE (ProjectName, EmployeeName)

Where ProjectName in STAFF_MEETING_EMPLOYEE must exist in STAFF_MEETING_2.

Activity 8

a.

We will assume that StudentNumber –› StudentName where name is not unique (i.e., there may be more than one "John Smith", each with a different student number). Then the multivalued dependencies are:

StudentNumber –›–›Club

StudentNumber–›–›Sibling

StudentNumber –›–› Nickname

Note: We cannot assume that StudentName –›–› Nickname because StudentName is not unique. For example, one John Smith may have the nickname "Johnny" while another John Smith has the nickname "Joe". If StudentName –›–›Nickname then John Smith ◎◎ ("Johnny", "Joe") which means that both nicknames apply to both

John Smiths. But this is not the case – each John Smith has only one nickname, and they are not the same.

b.

We will assume that StudentNumber –› StudentName where name is not unique (i.e., there may be more than one "John Smith", each with a different student number). Then the functional dependencies are:

StudentNumber –› StudentName, Dorm, RoomType

RoomType –› DormCost

Club –› ClubCost

Note: This assumes that only RoomType –› DormCost – that is, the cost of a certain type of dorm room is the same no matter what dorm the student is living in. An alternate assumption would be that (Dorm, RoomType) –› DormCost, where the cost of the type of dorm room varies from dorm to dorm.

c.

Functional dependencies and multivalued dependencies:

StudentNumber –› StudentName, Dorm, RoomType

RoomType –› DormCost

Club –› ClubCost

StudentNumber –›–›Club

StudentNumber–›–›Sibling

StudentNumber –›–› Nickname

Procedure:

Move the multivalued dependencies:

StudentNumber–›–›Sibling

StudentNumber –›–› Nickname

into their own tables, leaving the determinant, StudentNumber in the original relation as primary key, and making StudentNumber as a foreign ley in the new relations.

STUDENT_SIBLING (StudentNumber, Sibling) is in 4NF as the only two attributes in the multivalued dependency are the only determinant of the relation.

STUDENT_NICKNAME (StudentNumber, Nickname) is in 4NF as the only two attributes in the multivalued dependency are the only determinant of the relation.

STUDENT_2 (StudentNumber, StudentName, Dorm, RoomType, Club, ClubCost, DormCost) is not in BCNF because Club –› ClubCost and Club is not a key.

Therefore, move Club –› ClubCost into another relation leaving Club as a foreign key in the original relation, and making Club as a primary ley in the new relation.

STUDENT_CLUB_COST (Club, ClubCost) is in BCNF since Club is the only determinant and is also the key.

STUDENT_3 (StudentNumber, StudentName, Dorm, RoomType, Club, DormCost) is not in 4NF since StudentNumber –›–› Club

Move the multivalued dependency StudentNumber –›–› Club into a new relation, leaving the determinant, StudentNumber in the original table as primary key, and making StudentNumber as a foreign ley in the new relations.

STUDENT_CLUB_MEMBERSHIP (StudentNumber, Club) is in 4NF as the only two attributes in the multivalued dependency are the only determinant of the relation.

STUDENT_4 (StudentNumber, StudentName, Dorm, RoomType, DormCost) is not in BCNF because RoomType –›DormCost, and RoomType is not a key.

Therefore, move RoomType–›DormCost into another relation leaving RoomType as a foreign key in the original relation, and making RoomType as a primary ley in the new relation.

DORM_RATE (RoomType, DormCost) is in BCNF since RoomType is the only determinant and is also the key.

STUDENT_5 (StudentNumber, StudentName, Dorm, RoomType) is in BCNF since StudentNumber –› StudentName, Dorm, RoomType, and StudentNumber is the only determinant and key.

FINAL MODEL SPECIFICATIONS:

STUDENT_5 (StudentNumber, StudentName, Dorm, RoomType)

WHERE STUDENT.RoomType must exist in DORM_RATE.RoomType

DORM_RATE (RoomType, DormCost)

STUDENT_CLUB_MEMBERSHIP (StudentNumber, Club)

WHERE STUDENT_CLUB_MEMBERSHIP.StudentNumber must exist in

STUDENT_5.StudentNumber

AND STUDENT_CLUB_MEMBERSHIP.Club must exist in

STUDENT_CLUB_COST.Club

STUDENT_CLUB_COST (Club, ClubCost)

STUDENT_SIBLING (StudentNumber, Sibling)

WHERE STUDENT_SIBLING.StudentNumber must exist in STUDENT_5.StudentNumber

STUDENT_NICKNAME (StudentNumber, Nickname)

WHERE STUDENT_NICKNAME.StudentNumber must exist in STUDENT_5.StudentNumber