Database Design 2: College DB

Learning Outcomes: at the end of this section the student should be able to

1. Explain using examples the term dependency in database design
2. Explain using examples the term determinant in database design
3. Explain 1NF, and higher normal forms
4. devise and explain processing anomalies
5. apply rules of normal forms = normalisation process
6. Explain how logical modelling and normalisation are complementary.
7. Explain that analysis is crucial; i.e. setting meaning of attributes prior to design. If the spec is vague or missing info, then you must state assumptions prior to design.
8. Choose a Primary Key correctly; explain when to create Indexes and when to use Identifiers.

The following are two paper forms possibly used by the organisation.

Notice that we are complicating the problem by offering alternative paper views of the data. The student should be aware that a logical database design is independent of the display/paper format

C = Course; Cn = C1, C2 etc; S = Student number, Sn = S1,S2. etc.

Data Format 1a

C1 Database Btreacy B223l

S100 J.Bloggs 1, Kenley, Cork 40

S102 J.Murphy 3 Parchment Sq,Cork 45

C2 Modelling Mdavin F2.1

S100 J.Bloggs 1, Kenley, Cork 60

S102 J.Murphy 3 Parchment Sq,Cork 65

C3 Maths Kkelly C231

S104 F. O’Shea 5 Wilton Rd 80

Data Format 1b

S100 J.Bloggs 1, Kenley, Cork

C1 Database Btreacy B223l 40

C2 Modelling Mdavin F2.1 60

S102 J.Murphy 3 Parchment Sq,Cork

C1 Database Btreacy B223l 45

C2 Modelling Mdavin F2.1 65

S104 F.O’Shea 5 Wilton Rd

C3 Maths Kkelly C231 80

You should relate these forms to the Shipment forms in Design1. You should also imagine a folder full of these forms for every course or every student in CIT.

**The specification (problem description) is missing on purpose to make you realise the link between analysis and design. You cannot design unless you understand the meaning of each data attribute. Ask the lecturer or state your assumptions about the meaning before you start design.**

**You need**: the design procedure and solution of the Shipments database from Design 1.

Approach: Use the different design strategies of a) modelling and b) normalisation

a) Modelling; try use high level modelling to get a rough idea of how many objects and associations may be involved in this problem. How many table variables can you create to correctly store the data as given in one of the above data formats.

b) Normalisation: devise a single table to store all the attributes i.e. 7 columns in this case.

This is the most basic of design solutions. Each table must adhere to the basic properties of tables.

* Ensure there is no repeating groups i.e. the intersection of a row and column is a single value
* Identify a key for each table.

Now go on to

* Identify a list of Dependencies.
* Analyse this table for processing problems

Hint: to identify the key ignore attributes that describe and concentrate on attributes that identify. This should reduce the number of attributes you consider for the key of the table.

This design is of a similar complexity to the Shipments DB of Design 1. So, the solutions will ‘look similar’ but the attributes and table names will be different. So think about what data elements here could act like the elements from that design i.e. use the same design, but replace the old attributes with new ones from this new design specification. **It is important to develop your ability to recognise similarities in design.**

Report (C#, Title, Lname, Room#, S#, Name, Address, Mark )

This is a header that describes the structure of the table. A table can be created with this structure and have no rows in it (e,g, when starting off)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S# | Name | Address | C# | Title | Lname | Room# | Mark |

A row in this table is an instance of a report. Obviously as the student records database system runs over time more and more instances/rows will be added to the table as students take exams.

**A row in the Report table represents the information that a student obtained a mark on a course of a particular title given by a lecturer.**

**One important element that analysis would need to identify is whether a given course can be assigned to only one lecturer. We will assume that it is.** We will return to this issue later in this section.

Normalised to first normal form

Report

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| S# | Name | Address | C# | Title | Lname | Room# | Mark |
| S100 | J.Bloggs | 1, Kenley, Cork | C1 | Database | Btreacy | B223L | 40 |
| S102 | J.Murphy | 3 Parchment Sq,Cork | C1 | Database | Btreacy | B223L | 45 |
| S100 | J.Bloggs | 1, Kenley, Cork | C2 | Modelling | Mdavin | F2.1 | 60 |
| S102 | J.Murphy | 3 Parchment Sq, Cork | C2 | Modelling | Mdavin | F2.1 | 65 |
| S104 | F.OShea | 5 Wilton Rd, Cork | C3 | Maths | Kkelly | C213 | 80 |

Even at this stage you should be able to see similarities with the Shipments database.

All designs that mix a number of basic elements together in one table suffer from the same problems. In the Shipments design, we mixed the basic objects Supplier & Part along with the Shipment all in one table as our first design solution. This College database is similarly a mix of objects. Can you see them? A sign of potential problems is repetition in the data. Concentrate on what is being repeated.

What is the key(s) of this table? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Hint: to identify the key ignore attributes that describe and concentrate on attributes that identify. This should reduce the number of attributes you consider for the key of the table.

Key = (C#, S#)

List the dependencies for this table

(C#, S#) 🡪 all other attributes. This must be true as otherwise it couldn’t be key.

Note, however, that individually (i.e. taken on their own) some attributes within the key are determinants

C# 🡪 Title, Lname, Room# ; S# 🡪 Name, Address

Just as we say Mark is dependent on (C#,S#), we can say that (C#,S#) determines marks. We will use these terms interchangeably i.e. determines, or is determined by, dependent.

Examining the data in the table can you see any processing anomalies? i.e. undesirable side effects of an Insert, Update or Delete operation?

An update anomaly arises from repetition in the table. If some data exists (repeated) in different places then there is a risk of inconsistency in that you might update one instance but not the other thereby leaving the database in conflict (inconsistent)

As you can see

|  |  |  |  |
| --- | --- | --- | --- |
| C1 | Database | Btreacy | B223L |

Is repeated twice in the data given.

**NB: it is important to scale up the database to what it might look like in a real college like CIT. Since C1, Database, Btreacy, B223L is repeated for every student that has ever taken the database course, then we’d have much more repetition if this design was used as the student records system for CIT.**

**Repetition results in a waste of storage space and higher search times. It also carries the risk of inconsistencies in that a user could update one row and not another.**

**So identifying this update anomaly is important.**

As you can see

|  |  |  |  |
| --- | --- | --- | --- |
| C2 | Modelling | Mdavin | F2.1 |

Is also repeated.

We can think about an insert anomaly from two angles; from logical modelling we should be aware that a table may comprise of a mix of simple objects and more complex associations in one table. In reality, we would like to be able to insert information about the simple objects but we cannot because they are ‘mixed together’ with the association.

So we can’t have one without the other! Why is that?

**Well if you think about an insert anomaly from the angle of normalisation, we know that a key is required by each table. If a key is composite (compound) then you must have values for each and every attribute of that composite key.**

So in the above example if we have a key (C#, S#) then you must have both a C# AND an S# before you can insert a row into the table. So until a student (S#) enrols on a course (C#) you cannot store individual data about a student or a course.

So we cannot insert the data

|  |  |  |  |
| --- | --- | --- | --- |
| C1 | Database | Btreacy | B223L |

Until we have student S100 or S102 that enrols on the course. Once we have a student No( S#), we can form a key (e.g. S100, C1) and then insert the row.

When you think about the real world this is obviously a problem; in CIT the administration dept would like to enter data on all the available courses BEFORE or at least independently of any students who might enrol on those courses.

The delete anomaly is the ‘flip side’ of the insert anomaly. Suppose the last student on a course leaves? If we delete that last student (S#) then we have to delete (lose) all the data about that course since it cannot exist in the table without a proper key. Again, you cannot have

|  |  |  |  |
| --- | --- | --- | --- |
| C1 | Database | Btreacy | B223L |

Unless there is at least one student No (S#) enrolled on that course.

Recall the Shipment DB design; note the similarities with this College DB; in Shipments the key was (S#, P#) and so we could not insert data on an individual supplier Or on an individual part until we had a shipment. This is obviously not desirable as we should imagine a user such as an admin employee interacting with our database to manage and maintain our list of suppliers on its own i.e. without reference to any parts or shipments. Likewise for parts data and for course details here;

To eliminate these processing anomalies we need to ensure our tables are ‘better’. By this we mean they adhere to a rule that prevents these problems from occurring in the first place;

**Normalisation to higher normal forms**

**Take out any dependencies and make a new table for them, but still maintaining a link to the original table. The key of the new table must be the linking attribute(s) to the old.**

Applying this rule to our single table design solution, we see we must redesign by introducing new tables and reorganising (redistributing) the data.

So if we have a dependency : C# 🡪 Title, Lname, Room# we get

Course

|  |  |  |  |
| --- | --- | --- | --- |
| C1 | Database | Btreacy | B223L |

**Then to normalise to higher normal form we create a new table for this data with C# as the key. In effect we cut the repeated columns out of the existing table and paste them into a new table. But C# remains also remains as a ‘link’ column in the Enrols table.**

Higher Normal Form

Course

|  |  |  |  |
| --- | --- | --- | --- |
| C# | Title | LName | Room# |
| C1 | Database | Btreacy | B223L |
| C2 | Modelling | Mdavin | F1.2 |

This therefore leaves us with 3 columns left in the old table, with the same key **(C#, S#**) as before

Enrols

|  |  |  |
| --- | --- | --- |
| S# | C# | Mark |
| S100 | C1 | 40 |
| S102 | C1 | 45 |
| S100 | C2 | 60 |

Note keeping the link is essential. To leave just the two column S# and Mark in one table won’t work; can you see why?

|  |  |
| --- | --- |
| S# | Mark |
| S100 | 40 |
| S102 | 45 |
| S100 | 60 |

Ans: this table may tell me that Student S100 got marks 40 and 60; however I don’t know what course they got those marks in!!! In other words, the database must tell me in what course did the student get the mark; hence I need a C#, a S# and a mark. 3 columns.

Notice how our new two table design eliminates the processing problems from above.

Of course if we had analysed it correctly using logical modelling we would have never put these two independent things (Enrols and Course) mixed together in one table.

Alternative incorrect solution:

Some students try a solution to this problem using a course table for each course.

C1Databases

S100 40

S102 45

C2Modelling

S100 60

S102 65

C3Maths

S104 80

Why is this unworkable? It also does not adhere to our basic properties for table design, particularly the scalability requirement. That means, if we want to add a new course, our design should allow us to insert a new row in a table. In the design given above, we in fact have to change our complete design by adding a new table each time we want to add a new course. A final design is a set number of tables; you cannot add more as you go along!

**In a database you can only add new rows to existing tables once the design is implemented.**

We can repeat the exact same design process for another aspect of the given application; just replace ‘course’ with the notion of ‘student’ and you get the exact same problems/solutions.

The student repetition is harder to see in the table because it is not grouped together as the Course rows were. Also we only have two attributes for Student. (Recall similar to Part in Design 1). However, repetition still exists; it just takes a little work to see it.

So our solution to date using what we’ve learnt from the shipments design is that

Report **( S#, C#,** Title, Lname, Room#, Name ,Address Mark ) Is changed to become tables

Course (**C#,** Title, Lname, Room#) ; Report **( S#, C#,** Mark ) ; Student (**S#,** Name, Adrdress)

Is the design finished? Going back to the description of the application (spec) and noting that we should scale up the design to be realistic for a real college like CIT, can you see any potential problems arising? We were told earlier before starting the design that a given course has only one lecturer and that a given lecturer has one room#. (page 2). But there is another aspect of the problem that we DO NOT know for sure: What if

* A lecturer teaches on more than one course?

What effect(s) does this new aspect of the problem have on our table design?

It may not be immediately obvious but if we add in more rows in the course table for different courses we should be able to see a potential problem developing.

Course

|  |  |  |  |
| --- | --- | --- | --- |
| C# | Title | Lname | Room# |
| C1 | Database | Btreacy | B223L |
| C2 | Modelling | Mdavin | F1.2 |
| C5 | Networks | Btreacy | B223L |

Even though we only have a small number of attributes you should be realising that in the real world the meaning of what is to be designed is crucial; Good analysis is critical.

If you think about our new rule that a lecturer teaches on more than one course; as an analyst you might go a step further and ask,

* Can a course be taught by many lecturers?

This is sometimes true in certain cases here in CIT but for simplicity in this design we will take the meaning as set earlier (on page 2.2)

* A course has only one lecturer assigned to teach it.

So with this in mind Exercise:

* Examine the table for processing anomalies using the usual approach.
* What data might we like to insert independently of the other data in the table?
* What is repeated? Usually the data that is repeated is also the data you would like to insert independently!

From a modelling point of view, we still have an object embedded (mixed in with) another object or association. What is it? What ever it is, is causing the repetition.

Exercise: using the same approach used to break the table causing the problems up into smaller tables and redistributing the data but while still maintaining a link to the original table via the key(s) of the new table(s), write a new design solution.

What data is repeated?

|  |  |
| --- | --- |
| Btreacy | B223L |

What happens if we Btreacy moves office? We might update one entry but not another leaving btreacy in two different rooms in our system i.e. inconsistent data (Update anomaly)

Also, take the real world possibility that btreacy is removed from teaching duties for a year?

In our design we cannot insert the data

|  |  |
| --- | --- |
| Btreacy | B223L |

Unless we have C# to teach (since C# is the key) therefore an **Insert anomaly**.

Likewise, if we remove last C# allocation for btreacy, we can no longer store room# data

|  |  |
| --- | --- |
| Btreacy | B223L |

Solution!

We should redesign our tables to handle the dependency Lname 🡪 Room# as an independent table.

Follow the standard procedure; take out the columns that are causing the problems and create new table for them; but keeping the key of that new table as a link.

Take the existing course table

Course

|  |  |  |  |
| --- | --- | --- | --- |
| C1 | Database | Btreacy | B223L |
| C2 | Modelling | Mdavin | F1.2 |
| C5 | Networks | Btreacy | B223L |

And break it up by introducing a new table for the FD

Lecturer

|  |  |
| --- | --- |
| Btreacy | B223L |
| Mdavin | F1.2 |
| Btreacy | B223L |

Course

|  |  |  |
| --- | --- | --- |
| C1 | Database | Btreacy |
| C2 | Modelling | Mdavin |
| C5 | Networks | Btreacy |

Enrols table on next page.

Enrols

|  |  |  |
| --- | --- | --- |
| S# | C# | Mark |
| S100 | C1 | 40 |
| S102 | C1 | 45 |
| S100 | C2 | 60 |

Include the Student table is as it was.

Note: some students when designing this database wonder if the key of the new Lecturer table is room#? This could work if it is a fact that every room has only one lecturer in it. If this is true then you can state a dependency exists Room# 🡪 Lname i.e. room# is key and it determines the Lname however intuitively it seems better to use name as the identifier; think about it this way; when a lecturer introduces him/herself would they say’ hello I’m B223L’ no they’d use **their name to identify themselves**, Hi, I’m B.Treacy’

Students sometimes go a step further and say well Lname isn’t a good identifier as you might have lecturers with the same name e.g. two people called Sean Murphy. This is valid and would have to be cleared up at the analysis phase prior to design; In a way this is a separate issue. We must work with the attributes we are given; so for the sake of the exercise here we’ll assume that Lname is ok to use as an identifier.

Note: Even if you use room# as key you must remember that a link must be made back to the course table to maintain the information that a course is taught by a lecturer.

**Higher Normal Forms (2 & 3NF)**

**We have rules that define a 1NF table from Design 1 notes i.e. a table in 1NF has a fixed number of columns, no repeating groups and a primary key.**

**Database design has a number of Normal Forms, each one has more (or stronger) rules. Therefore, as we apply the rules, we say we move to a higher Normal Form, which is a higher level of correctness as the design adheres to more or stronger design rules.**

**All database text books have separate rules for each normal form, but we will simplify these into one rule.**

**A table is in a higher normal form is every attribute is dependent on the key the whole key and nothing but the key.**

Recall: to move a design from 1NF to a higher NF,

**Take out any dependencies and make a new table for them, but still maintaining a link to the original table. The linking attribute is the key of the new table.**

**Extra section (only required if you have covered partial and transitive dependencies)**

**Not for BBADM:**

**A partial dependency is where an FD exists A 🡪 B, but A is a composite and B is in fact only dependent on a subset of the attributes that make up A.**

**So, in Design 1 (S#, P#) is composite, it forms the Primary key, and therefore it determines all other attributes, but S# 🡪 Sname, status, city etc. These are partial dependencies.**

**2NF Definition: a table is in 2NF if it is 1NF and has no partial dependencies**

**A transitive dependency is where a chain of dependency exists A🡪 B, but in turn B🡪 C. TDs are about FDs that exist in a table outside of the key.**

**In this case, we may have no partial dependency, as C is dependent on an attribute(s) outside of the key. So, in Design 2: C# 🡪 Lname and Lname 🡪 room#.**

**3NF Definition: a table is in 3NF if it is 2NF and has no transitive dependencies.**

Exercise: review your final designs for Shipment and College to verify that the above rule is true for each attribute in the tables.

To end with one final analysis twist, what would happen if we now state that

* A given course can be taught by many lecturers

This will be case when modules are used in CIT. Our design must now be able to handle information like C1, Database, B.Treacy, J.Sherwin, D.Simpson. i.e. Database course C1 has three lecturers assigned to it.

Satisfy yourself that this cannot be handled in the final design above. i.e. try put in the information…you cannot without violating our properties or rules for normal forms.

Exercise: redesign to resolve this issue and allow many lecturers teach on a course.

What if you are good at modelling but not good at the formal normalisation process?

Some students prefer the abstract high level modelling methodology to do design. They can get an overall idea of the different tables required by just thinking about the specification i.e. the description of the problem. If you are such a student it is important to remember that in a database course you will be examined on your understanding of the normalisation process (which is integral to a database course). So, you should realise how the two approaches are really doing the same thing.

**Note that both design methodologies should end up with the same design!**

So if you can model well and devise a set of tables as a solution, this solution should match the normalisation process. Here’s how, assume you modelled the following set of tables

Lecturer

|  |  |
| --- | --- |
| Btreacy | B223L |
| Mdavin | F1.2 |
| FSheehan | PF43 |

Course

|  |  |  |
| --- | --- | --- |
| C1 | Database | Btreacy |
| C2 | Modelling | Mdavin |
| C5 | Networks | Btreacy |

Enrols

|  |  |  |
| --- | --- | --- |
| S# | C# | Mark |
| S100 | C1 | 40 |
| S102 | C1 | 45 |
| S100 | C2 | 60 |

Student

|  |  |  |
| --- | --- | --- |
| S100 | J.Bloggs | Cork |
| S102 | J.Murphy | Youghal |
| S104 | F.OShea | Cobh |

How does this correspond to normalisation?

1. Identify a key in each of your tables. The key of each table functionally determines all the other attributes in the table so list them all.
2. Find the most complex key in your set of tables (nearly always a composite key to link the basic objects in an association of some sort e.g. enrols above). That key is usually the key of the original single table in the normalisation process i.e. the 1NF table; (S#, P#) in Shipment Design1; and (C#, S#) here in Design 2.
3. The key of the first NF table determines ALL other attributes.
4. The insert, update, and delete processing anomalies stem from mixing tables together in one table when they should be separated. The lecturer and course table in a single table solution should never have been mixed up with the Enrols table to start with; this is what causes the anomalies. You cannot insert a lecturer until there is an enrolment; you cannot insert a course until there is an enrolment. For Update, every attribute in Lecturer is repeated many times if you only use the one table for all design. This means that you are forced to do multiple updates when you should just do one (e.g. to update room# you should just update that attribute just once in a good design, but in our first NF table that data is repeated for every course the lecturer teaches!) The same applies to every attribute in the other tables e.g. course attributes i.e. to update course name from Database to Database Systems should be done once, but in the first NF design that data is repeated many times for every student that enrols on that course!. Finally, for Delete, in a mixed up one table design, no data can exist without a key; if the key is made up of many attributes (as in every composite key) you must have all of them to store data. That means that for every basic table you model (lecturer and course above), you cannot insert data on any of these in isolation (as there is no independent table to store that data). So if you delete the last entry in the Enrols table for a lecturer, you lose all the lecturer data; same for course.

The exact same applies for ALL designs; in Design 1 Shipments; we should have independent table for Suppliers and Parts; but we did not. **We have a complex key that forces us to have both a Sno AND a Pno BEFORE you can store any other data**. So you cannot insert data on a supplier until there is a shipment; as for Part; Data on Supplier is repeated for all shipments in an 1NF table, so update problems; if you delete the last shipment for a supplier, then you lose the supplier data.

In any design you do, the 1NF is a mixed up table. Just break it up into its basic elements, and that indicates what was wrong with the original.

Rule of thumb:

To verify your design:

Check that each table you create works in isolation for processing operations Insert, Update and Delete. In other words, does it make sense from a processing point of view to isolate that data in a table of its own? Try to imagine the system in operation in the real world. For the Shipment database can you reasonably imagine some user going to your database and inserting or updating just info on a Supplier? Yes, the company will have a list of suppliers they store info on. Same for Parts, Shipments etc. Each table would be processed independently for additions, modifications etc.

Similarly for the College Students record database; someone like the dept secretary or head of department would be allowed insert, update and delete info on students, lecturers, results etc all at different times and the courses might be managed by the Registrars office so the inserts etc are performed there. But your design handles that.

Learning Outcomes: at the end of this section the student should be able to

1. Explain using examples the term dependency in database design
2. Explain using examples the term determinant in database design
3. Explain 1NF, and higher normal forms
4. devise and explain processing anomalies
5. apply rules of normal forms = normalisation process
6. Explain how logical modelling and normalisation are complementary.
7. Explain that analysis is crucial; i.e. setting meaning of attributes prior to design. If the spec is vague or missing info, then you must state assumptions prior to design.

8. Choose a Primary Key correctly; explain when to create Indexes and when to use Identifiers.