# NORMALIZATION

**NORMALIZATION**

Now we are ready to work our conceptual design into a format that can be translated into tables for actual database implementation. Historically, this once had been a process of intuitive logic. For many students this intuitive approach is often used. Without a LOT of experience the intuitive process often leads to failure. As in the past the intuitive method often led to data redundancy, and designs where relationships could not exist.

#### ANOMOLIES FIRST

## PROBLEMS & ANOMALIES

Example 1

The sports centre would like to keep track of students, what sports activity they enrol in and the fee they paid for that activity.

Assume the following entity and sample stored data

**ACTIVITY [SID, ACTIVITY, FEE]**

|  |  |  |
| --- | --- | --- |
| SID | ACTIVITY | **FEE** |
| **100** | **SKIING** | **200** |
| **150** | **SWIMMING** | **50** |
| **175** | **SQUASH** | **50** |
| **200** | **SWIMMING** | **50** |

## PROBLEM

If student 100 finishes doing the sport skiing, then the row with that student information should be deleted.

What happens when you delete student 100?

🡺 The information on student 100 is lost. That is not a problem since they are not doing the activity

🡺 How much is skiing? I know you can remember but if it isn’t stored how will the system know if we offer skiing and also how much the fee for skiing would be.

WHAT CAUSES THE PROBLEM?

🡺 FEE is dependent upon ACTIVITY and not on SID

🡺 ACTIVITY is not dependent on the student as a student can have more than one activity

Deletion anomaly is the term given to this problem.

**PROBLEM 1 SOLUTION**

Break the table into 2 tables

## RESULT

ACTIVITY

|  |  |
| --- | --- |
| ACTIVITY | FEE |
| SKIING | 200 |
| SWIMMING | 50 |
| SQUASH | 50 |

ENROL

|  |  |
| --- | --- |
| SID | ACTIVITY |
| 100 | SKIING |
| 150 | SWIMMING |
| 175 | SQUASH |
| 200 | SWIMMING |

Now delete student 100. The information about the skiing activity still exists.

**PROBLEM 2 – ADD ANOMOLY**

Given the 2 tables in the solution before:

ACTIVITY

|  |  |
| --- | --- |
| ACTIVITY | FEE |
| SKIING | 200 |
| SWIMMING | 50 |
| SQUASH | 50 |

ENROL

|  |  |
| --- | --- |
| SID | ACTIVITY |
| 100 | SKIING |
| 150 | SWIMMING |
| 175 | SQUASH |
| 200 | SWIMMING |

Suppose that student 250 wants to enrol in racquetball. The row would like as follows:

|  |  |
| --- | --- |
| 200 | RAQUETBALL |

The data can be added to the ENROL table

Should we add it if no such activity exists?

We shouldn’t is the initial answer, but it depends on how the organization handles this case. For example the sports centre may use programming that states if a student enrols in an activity that does not exist in the ACTIVITY table, then go to the screen that will add this activity to the table or prevent the ENROL row from being stored.

NOTE: the important thing is to not to let the student be enrolled in a non-existent activity. The way to do this is through INTEGRITY CONSTRAINTS. (Will be covered later in the course)

## SOLUTION TO ADD ANOMOLY

**The prevention from adding to one table when an important piece of information does not exist in the other table is called a constraint.**

**The constraint is called referential integrity.**

# KEY 🡺 Breaking table into 2 still has its problems

**READ also 07-update-anomolies.docNORMALIZATION**

**NORMALIZATION is a step-by-step approach to analyzing the functional dependencies between attributes (columns in a table).**

## STEPS IN NORMALIZATION - simplified

First Normal Form - 1NF

- Every row has a unique primary key, or unique identifier. In other words there are no duplicate rows

- There are no repeating attributes; most DBMS’s cannot deal with repeating columns

Second Normal Form - 2NF

- Must be in 1NF first before tackling the 2NF process

- Each attribute that is not part of the primary key must be **functionally dependent** of the entire key - - Put another way there can be no partial dependencies are not allowed

Third Normal Form - 3NF

- Must be in 2NF first before applying the 3NF process

-There are **no transitive dependencies**, meaning that an attribute cannot be dependent on another attribute that is not the primary key or could not be a unique identifier (alternate key).

Let's work through these steps with our conceptual design. Again we **assume we can only find 2 entities**. We used notation that show mandatory/optional attributes and also those that repeat. We also converted street, city, prov, and postal code into one attribute called ADDRESS. Here is the current status.

STUDENT [STUDNO, NAME, ADDRESS, (Phone), (Coursecd, Coursedescription, Grade)]

FACULTY [FACNO, FNAME, OFFICE, FPHONE]

## Assigning a unique identifier

To establish these entities into 1NF, we must first assign a primary or unique key.

Look at the attributes in STUDENT. There are two possible keys, the STUDNO and NAME. Names can often be the same and so if we choose name as the primary key, we cannot be sure the value will be unique. Our best choice is STUDNO. We are fortunate that our list of attributes contains a suitable primary key. There may be times when a primary key does not exist and you will have to create one.

## PRIMARY – CANDIDATE – ALTERNATE KEYS

In some cases there are several possible choices for a primary key, and these choices are called **candidate keys**. In a file about employees who drive trucks for a company there might be an employee number, social insurance number, drivers licence and OHIP number for each employee. All of these uniquely identify an employee and so each could be a candidate to be used as the primary key. Once a primary key is selected these others are known as **alternate keys**.

We also need a primary key for FACULTY. FACNO is our best choice.

The entities can now be written as

STUDENT [STUDNO, NAME, ADDRESS, (Phone), (Coursecd, Coursedescription, Grade)]

FACULTY [FACNO, FNAME, OFFICE, FPHONE]

### REMOVE REPEATING ATTRIBUTES

To completely establish our entities in 1NF we must remove the repeating attributes. Attributes repeat with respect to the primary key. Ask yourself in the case of the STUDENT entity; given one student id can there be more than one name. The answer is no so the attribute name does not repeat for one occurrence of a student id. Keep asking if each of the attributes repeats in relationship to the PK. As we discovered the repeating attributes before, we already know which ones repeat and which ones repeat together.

The way to remove repeating attributes or repeating groups of attributes is often to create a new entity. PHONE would then be one of the new entities.

ASIDE: There are two ways of handling PHONE. If there were 2 or 3 phones allowed per student, we could assign attributes such as PHONE1, PHONE2, PHONE3 or HOMEPHONE, CELLPHONE and WORKPHONE. For each attribute such as PHONE1 there would only be one value stored. This is likely the approach for most systems.

**However**, for this system we want to take the approach of creating a new entity in order to demonstrate normalization. To remove the (Phone) out of STUDENT, we can create a new entity called STUDPHONE. Not only do we remove (Phone) but we must also carry a copy of the primary key STUDNO along with it so that we know who's (Phone) it is. In other words we bring the original key with it to maintain the relationship. The result is now.

STUDENT [STUDNO, NAME, ADDRESS, (Coursecd, Coursedescription, Grade)]

STUDPHONE [STUDNO, PHONE]

Notice that both attributes in the new group are required or mandatory. If PHONE were still optional, there would be no point in having a row in the table with no data.

We must also remove the repeating group (Coursecd, Coursedescription, Grade). There are again two ways of handling this. We can make many attributes in the entity, similar to PHONE1 and PHONE2, but that would not be practical. If there were 70 courses that a student could take over time, then COURSECD1 to COURSECD70 would be needed as well as the attributes for course description and grade. That is 210 columns. For most occurrences of STUDENT the row of data would be very empty as most students do not take 70 courses. The better choice is to again create a new entity with the repeating attributes. This entity will be called STUDCOURSE. We must also carry the STUDNO so we will know which student has these courses.

STUDENT [STUDNO, NAME, ADDRESS]

STUDCOURSE [STUDNO, COURSECD, COURSEDESCRIPTION, Grade]

The Grade is still left as optional as a Grade will not be assigned until the end of the semester.

Updating our Conceptual design we see it looks like this.

STUDENT [STUDNO, NAME, ADDRESS]

STUDPHONE [STUDNO, PHONE]

STUDCOURSE [STUDNO, COURSECD, COURSEDESCRIPTION, Grade]

FACULTY [FACNO, FNAME, OFFICE, FPHONE]

Can we now say our design is in 1NF? Go back and check the rules. What is missing?

We still have two entities without primary keys. They are the two new entities created to remove the repeating groups.

Let's examine the alternatives for a primary key for STUDPHONE.

1. If STUDNO is selected, then its value will be duplicated for each PHONE the student has. Remember, the primary key must be unique. For this reason STUDNO will not be acceptable as a primary key.
2. The other attribute, PHONE looks better at first glance, but what situation arises when two students share the same apartment and the same phone?? We cannot be sure each value of PHONE will be unique.

We can create a unique key by concatenating the two attributes together. If I use all the characters in STUDNO and PHONE strung together, then the value will be unique. Often in database design we must use a combination of attributes to establish a primary key. Note that there is still one primary key per entity.

In STUDCOURSE we must use STUDNO and COURSECD concatenated together because each by itself will not be unique. Why?

Our conceptual design in 1NF will now look like this

STUDENT [STUDNO, NAME, ADDRESS]

STUDPHONE [STUDNO, PHONE]

STUDCOURSE [STUDNO, COURSECD, COURSEDESCRIPTION, Grade]

FACULTY [FACNO, FNAME, OFFICE, FPHONE]

NOTE: It is essential that 1NF be done correctly. A lot of the design problems that occur are due to missing repeating data.

**Redundant data or data that repeats over many rows**

There is another type of repeating data that occurs in each row. To see it you have to look at some sample data.

FOODMART is a company that sells groceries through 300 stores across Canada. It keeps track of the inventory that it sells. A sample report might look like the following:

| **Product Query** | | | | | |
| --- | --- | --- | --- | --- | --- |
| **Product\_ID** | **Product\_Name** | **Price** | **Brand\_Name** | **Subcategory** | **Category** |
| 280 | Bagels | $1.86 | Great | Bagels | Bread |
| 591 | Bagels | $1.22 | Colony | Bagels | Bread |
| 907 | Bagels | $3.69 | Fantastic | Bagels | Bread |
| 1218 | Bagels | $1.85 | Sphinx | Bagels | Bread |
| 1529 | Bagels | $3.40 | Modell | Bagels | Bread |
| 46 | Jack Cheese | $3.58 | Club | Cheese | Dairy |
| 47 | Muenster Cheese | $2.78 | Club | Cheese | Dairy |
| 48 | String Cheese | $1.90 | Club | Cheese | Dairy |
| 49 | Low Fat String Cheese | $3.68 | Club | Cheese | Dairy |
| 50 | Havarti Cheese | $4.19 | Club | Cheese | Dairy |
| 51 | Head Cheese | $1.48 | Club | Cheese | Dairy |
| 52 | Cheese Spread | $2.26 | Club | Cheese | Dairy |
| 53 | Sharp Cheddar Cheese | $3.68 | Club | Cheese | Dairy |
| 54 | Mild Cheddar Cheese | $2.99 | Club | Cheese | Dairy |
| 357 | Jack Cheese | $1.85 | Carlson | Cheese | Dairy |
| 358 | Muenster Cheese | $3.98 | Carlson | Cheese | Dairy |
| 359 | String Cheese | $1.35 | Carlson | Cheese | Dairy |

Note that there is a lot of vertical repetition such as BREAD and DAIRY in the category columns. There is also BAGELS, CHEESE in the subcategory column and the same for BRAND NAME. If you could see the entire 30,000 row PRODUCT report you would see there is a lot of redundancy. A lot of space used.

At Seneca all of the students belong to a program such as Computer Programming and Analysis, a long name to store for 2000 students compared to a code such as 122 or CPA. Seneca also stores the School (Computer Studies) for each student. Over 45 years of existence Seneca has probably a million student rows of data.

Suppose you were asked to read the list and count the number of Computer Studies students there were. Would an entry like Computor Studies be counted? The computerized system can not recognize the spelling error. The result is another group. The chance for a lot of errors has increased.

The solution is to reduce the redundancy is to break the entity into more than one entity as follows:

PRODUCT [PID, PRODNAME, PRICE, BRANDNAME, SUBCATNAME, CATNAME]

PRODUCT [**PID**, PRODNAME, PRICE, BRANDID, SUBCATID, CATID]

BRAND [**BRANDID**, BRANDNAME]

SUBCATEGORY [**SUBCATID**, SUBCATNAME]

CATEGORY [**CATID**, CATNAME]**2nd NORMAL FORM**

**To be in 2NF the design must be in 1NF**.

**The design must also eliminate partial dependency. A partial dependency exists when a non-key attribute is dependent on only part of the primary key.**

1

NOTE: It is very important that you check that the entities are really in first normal form. Many students err at first through lack of experience. Often if you think of possible data to put into the rows of a table you may see the repeating values.

Based on the above rules, we need only check those entities whose primary key is composed of more than one attribute. A partial dependency would exist if a non-key attribute were dependent on only one of the attributes that make up the concatenated key.

Therefore STUDPHONE and STUDCOURSE must be examined. STUDPHONE has no non-key attributes; therefore a partial dependency cannot exist. STUDCOURSE has two non-key attributes.

STUDCOURSE [STUDNO, COURSECD, COURSEDESCRIPTION, Grade]

Ask yourself this question

IS THE VALUE OF COURSEDESCRIPTION DETERMINED BY THE VALUE OF **BOTH** STUDNO AND COURSECD??

Your answer should be no. The value of the COURSEDESCRIPTION will be the same regardless of the STUDNO associated with it. The COURSEDESCRIPTION attribute is functionally dependent on COURSECD. We must remove this partial dependency from the design. This can be accomplished again by creating a new entity, COURSES.

COURSES [COURSECD, COURSEDESCRIPTION]

STUDCOURSE [STUDNO, COURSECD, Grade]

Notice that when we create the new entity this time, only the part of the primary key that carried the dependency from the original entity is **copied** into the new entity.

Is Grade a partial dependency?? No. The value of grade will be determined by the value of STUDNO and also the value of COURSECD i.e. this Grade is for a particular student taking a particular course. Grade is functionally dependent on the entire key.

Is the design in 2NF?? Go back and check the rules. The rules tell us that it must be in 1NF, and so COURSES must have a primary key. A good choice is COURSECD, leaving our design looking like this

STUDENT [STUDNO, NAME, ADDRESS]

STUDPHONE [STUDNO, PHONE]

COURSES [COURSECD, COURSEDESCRIPTION]

STUDCOURSE [STUDNO, COURSECD, Grade]

FACULTY [FACNO, FNAME, OFFICE, FPHONE]

**3rd NORMAL FORM**

**To be in 3rd normal form it must first be in 2nd normal form**.

The design must remove all transitive dependencies. Simply put no non-key attribute can depend on another non-key attribute. They should be functionally dependent upon the key.

2

We have to look at each attribute and decide if it is functionally dependent on the key, or another attribute in the entity. **These can be difficult to spot**. What you are really trying to determine is if the value of a non-key attribute is determined by the value of the key. First when we say non-key attribute we are referring to attributes that are not the primary key, or alternate key.

Let's look at the FACULTY entity. Is the value of the FNAME determined by the value of the FACNO?? Yes. The faculty number determines a faculty member’s name. A change in the FACNO would result in a change in the FNAME. Now do the same approach but comparing all non-key attributes only.

Again the OFFICE value tells us where to find the person (FNAME) represented by the FACNO.

Look carefully at FPHONE. If there were no faculty at all, just a lot of offices and phones at the campus, could we still determine the phone numbers? Yes, because the phones are placed in the offices and are dependent on the office that they are in. In our design FPHONE is functionally dependent on OFFICE. This is called a TRANSITIVE dependency. Given a value for OFFICE we can get a value for FPHONE. To have the design in 3NF, all transitive dependencies must be eliminated. This is most often done by creating a new entity.

OFFICES [OFFICE, FPHONE]

FACULTY [FACNO, FNAME, OFFICE]

You can see that to create the new entity, both attributes involved in the transitive dependency are included in the new entity. In order to place all of the entities in 3NF we must go back to the beginning and make sure all the newly created entities are taken through the process of normalization (from 1 to 3). The 1st normal form requires us to establish the primary key for OFFICES. OFFICE is a good choice, since it represents the office number. Also notice that OFFICE remains in the FACULTY entity in order to relate the two entities.

Our design now looks like this

STUDENT [STUDNO, NAME, ADDRESS]

STUDPHONE [STUDNO, PHONE]

COURSES [COURSECD, COURSEDESCRIPTION]

STUDCOURSE [STUDNO, COURSECD, Grade]

FACULTY [FACNO, FNAME, OFFICE]

OFFICES [OFFICE, FPHONE]

For the most part this design is in 3NF, but there still remains a transitive dependency.

Earlier you will remember we changed STREET, CITY, PROVINCE, and POSTAL CODE into the single attribute ADDRESS because it was shorter and helped the entity to appear on one line. It would not be stored as a single attribute, so let us look more carefully at those attributes.

Is the value of STREET, CITY or PROV solely dependent on STUDNO or can you see a dependency on POSTAL CODE? In other words, can POSTAL CODE determine the value of STREET, CITY and PROV? The answer is YES for CITY and PROV. That would require you to store all the possible postal codes in your database to achieve absolute 3NF?? In this case we need to make a judgement call and ignore this transitive dependency in our design.

ANOTHER EXAMPLE OF SOLVING 3NF

INVOICE [**INVOICE\_NUMBER**, INVOICE\_DATE, CUSTOMER\_NUMBER, CUSTOMER NAME]

Given an invoice number we can determine the values of date, customer number and name. If you look at the non-key attributes we can see that given a date would not give you a single value for customer number or name. There can be many thousands of invoices on a specific date. Given a customer number a single value for customer name exists. This means that customer name depends upon customer number. Remove customer number along with a copy of attribute it depends upon giving a new entity

CUSTOMER [CUSTOMER\_NUMBER, CUSTOMER\_NAME]

This new entity needs to be normalized.The **E/R DIAGRAM** represents the conceptual design.

OFFICE

COURSE

STUDENT

PHONE

STUDCOURSE

FACULTY

**Sometimes Normalization is not enough** to ensure that the design does not have problems. As you can see from the E/R diagram above, there is no relationship between FACULTY and COURSES. This occurs because there is no **FOREIGN KEY**.

A **foreign key** exists primarily so that a relationship can be built by the DBMS at the time of retrieval. Remember, relational DBMS’s do not maintain a physical link.

There is a common attribute of OFFICE between the FACULTY and OFFICES entities.

FACULTY [FACNO, FNAME, OFFICE]

OFFICES [OFFICE, FPHONE]

The only purpose for carrying the OFFICE attribute in the FACULTY entity is to relate it to the entity OFFICES. We call OFFICE a foreign key and it can be represented in the entity list with italics Or some other form like broken underline or AS OFFICE (fk).

FACULTY [FACNO, FNAME, *OFFICE*]

OFFICES [OFFICE, FPHONE]

**MODULE 04 WORKSHOP**

1. For the STUDENT COURSE APPLICATION system that we have been working with in the module, showing all foreign keys modify the design into 3NF with the following conditions

- Each course can have numerous sections

- A faculty member is assigned to each section

- We need to be able to retrieve all of the students in a faculty members classes

2. Draw the E/R diagram that will represent the design in 1 above.

3. For each of cases 1, 2 and 3 and using the entity lists and E/R diagrams which you have already developed in previous modules

- Normalize each design into 3NF

- Draw E/R diagrams for each design