CS4416 Project Group 22

## The names and IDs of the students in your project group and which parts of the project each student has worked on.

Group members are:

- Michelle Hourihan Watanabe 12174521

- Patrick James O'Neill 19225539

- Richard O'Brian 19212941

- Laszlo Szlatki 18169902

|  |  |
| --- | --- |
| **Activity** | **Author(s)** |
| Log | Laszlo Szlatki |
| Explanation of the database’s purpose | Michelle Hourihan Watanabe, Patrick James O’Neill |
| Entity-relationship diagram | Laszlo Szlatki, Michelle Hourihan Watanabe |
| schema | Laszlo Szlatki, Michelle Hourihan Watanabe |
| Sample data | Patrick James O'Neill |
| FDs in Student table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Past\_Student table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Enrollment table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Subject table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Subject\_allocation table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Exam\_Results table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Staff table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| FDs in Address table | Laszlo Szlatki, Michelle Hourihan Watanabe |
| Prove of Student table in 3NF | Laszlo Szlatki, Patrick James O’Neill, Richard O’Brien |
| Prove of Past\_Student table in 3NF | Laszlo Szlatki, Michelle Hourihan Watanabe, Richard O’Brien |
| Prove of Enrollment table in 3NF | Laszlo Szlatki, Richard O’Brien |
| Prove of Subject table in 3NF | Laszlo Szlatki, Michelle Hourihan Watanabe, Richard O’Brien |
| Prove of Subject\_allocation table in 3NF | Laszlo Szlatki, Richard O’Brien |
| Prove of Exam\_Results table in 3NF | Laszlo Szlatki, Michelle Hourihan Watanabe, Richard O’Brien |
| Prove of Staff table in 3NF | Laszlo Szlatki, Richard O’Brien |
| Prove of Address table in 3NF | Laszlo Szlatki, Richard O’Brien |
| View: Multiple\_Subject\_score\_over\_70 | Laszlo Szlatki |
| View: limerick\_students\_in\_arrears | Michelle Hourihan Watanabe |
| View: min5\_student\_failed\_per\_subject | Patrick James O'Neill |
| Index: scores | Laszlo Szlatki, Patrick James O'Neill |
| Index: fees\_paid | Laszlo Szlatki |
| Procedure: studentAddToPast\_student | Laszlo Szlatki, Patrick James O'Neill |
| Trigger: delete\_student | Laszlo Szlatki, Patrick James O'Neill |
| Function: number\_enrolled | Michelle Hourihan Watanabe |
| Trigger: Discount\_popular\_subject | Michelle Hourihan Watanabe |

## A couple of paragraphs explaining what your database is about and a description of an imaginary software system that uses your database

We have designed a database that is aimed at schools and educational institutions which includes present and past student and staff details, enrolment information, details of exam results, subjects taught and teacher allocations. Our initial design started with 5 tables, though we found that we needed to add more tables to avoid many to many relationships and transitive dependencies. Our final database has eliminated the transitive dependencies that existed in some of the initial tables. Our database has 8 tables in total, which include student and staff details, addresses, subjects, student exam results, student enrolments to subjects, teachers allocated to subjects and past students. All tables are organised around closely related schemas. We have tried to ensure that attributes within each schema are highly connected and relevant to the schema from the offset.

We imagine that such a database would be used by teachers, administration staff and students. Through the database, students would be able to 1) access information about the subjects they were enrolled on; 2) see their exam results, review and update their own personal details and access some details about their subjects and teachers for their subjects, 3) check fees they have paid and arrears. Teachers would be able to 1) access their own personal information, though not other teachers’; 2) view the subjects they are allocated to; 3) review the student exam results of the subjects they are allocated to. Admin staff could use the database to 1) retrieve and update personal data about all students and staff; 2) track payments paid and arrears and issue letters if someone is in arrears; 3) access academic details regarding subjects, subject allocation and student progress and notice if some of them need additional help with their studies based on exam results.

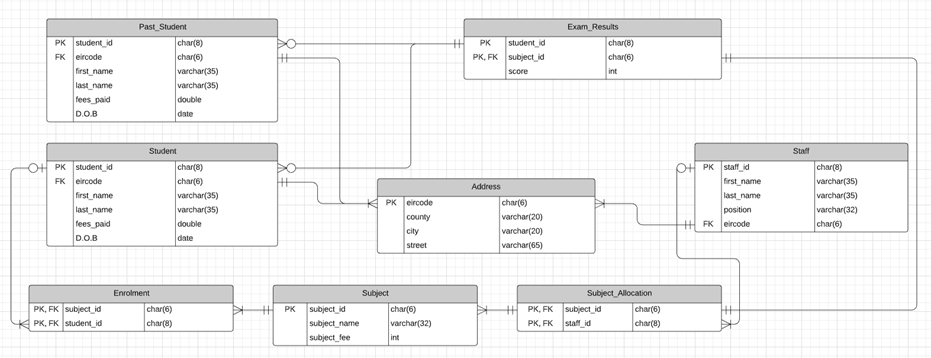
The software that would use our database would focus on administration and management within a school environment.

It would have features such as:

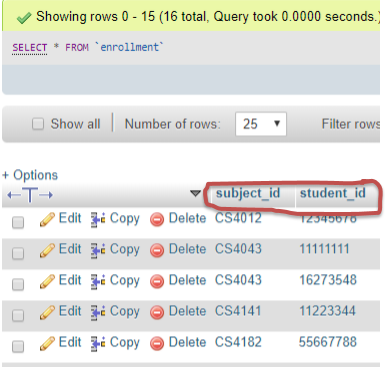
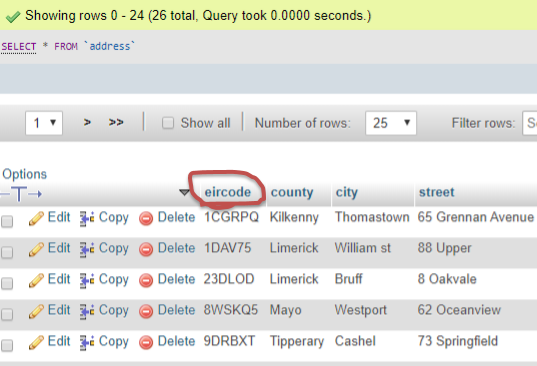
|  |  |
| --- | --- |
| * Course management * Enrolment * Examination management | * Student information * Employee/teacher management * Human resources |

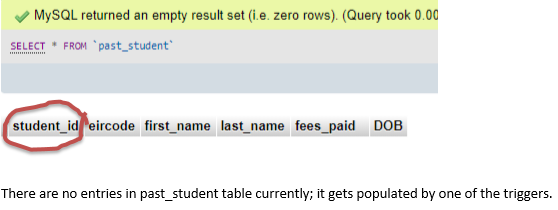
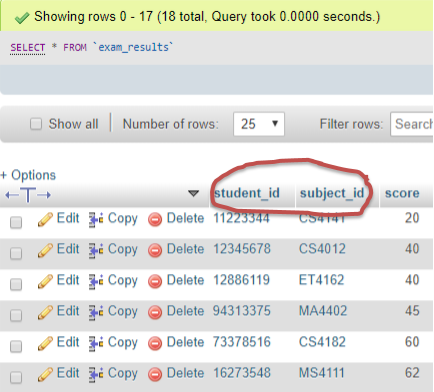
The software would probably offer other features for staff such as online calendars for scheduling, an email / messaging system. It would need to be accessible onsite and easy to use. We imagine that staff admin staff would be the primary users of the database to access and update when necessary, personal data about students and staff and professional data about subjects taught.

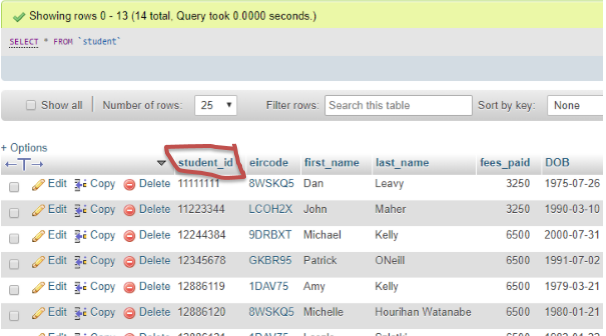
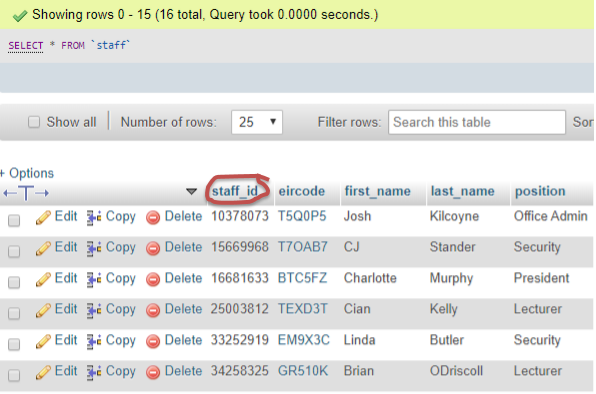
## Entity-relationship diagram for your database

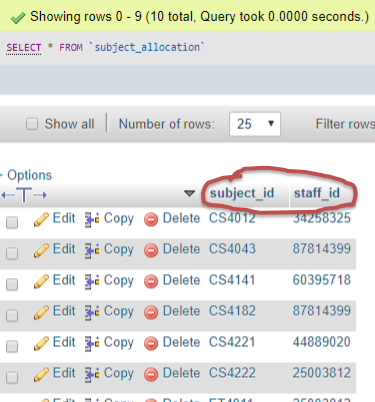
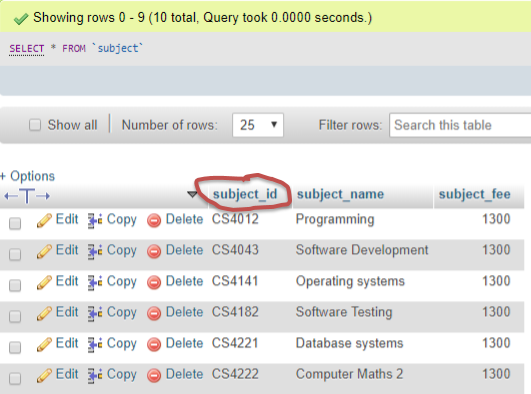


## An example of each table with some data and primary key attributes clearly identified.









## The list of FDs for each table

### 5.1 Student: (student\_id, first\_name, last\_name, fees\_paid, DoB, Eircode)

We presume none of the attributes on the RHS are unique. Several students can live in the same address (i.e. student accommodation) and some of them can have the same DoB.

**student\_id —> first\_name**

**student\_id —> last\_name**

**student\_id —> fees\_paid**

**student\_id —> DoB**

**student\_id —> eircode**

### 5.2 Past\_Student: (student\_id, first\_name, last\_name, fees\_paid, DoB, Eircode)

We presume none of the attributes on the RHS are unique. Several past\_students can live in the same address (i.e. student accommodation) with the same eircode and some of them can have the same DoB.

**student\_id —> first\_name**

**student\_id —> last\_name**

**student\_id —> fees\_paid**

**student\_id —> DoB**

**student\_id —> eircode**

### 5.3 Enrollment: (student\_id, subject\_id)

Here we have a composite primary key. Each attribute is dependent on the combination of the primary key.

**student\_id, subject\_id —> student\_id**

**student\_id, subject\_id —> subject\_id**

### 5.4 Subject: (subject\_id, subject\_name, subject\_fee)

If we know the subject\_id we can derive the subject name and subject\_fee. As there can be the same subject offered under different subject\_id’s, the name does not determine the id or the fee.

**subject\_id, —> subject\_name**

**Subject\_id —> subject\_fee**

### 5.5 Subject Allocation: (subject\_id, staff\_id)

We have a composite primary key and all attributes are part of the key. There are no other attributes. Each attribute is dependent on the combination of the primary key.

**subject\_id, staff\_id —> staff\_id**

**subject\_id, staff\_id —> subject\_id**

### 5.6 ExamResults:(student\_id, subject\_id, score)

We have a composite primary key and the other non-key attribute is dependent on both prime keys and cannot be determined otherwise. If we know a student’s id and the corresponding subject\_id, then we can determine the student’s score for that subject.

**student\_id, subject\_id —> score**

**student\_id, subject\_id —> subject\_id**

**student\_id, subject\_id —> student\_id**

### 5.7 Staff: (staff\_id, first\_name, last\_name, position, eircode)

All details about staff can be derived from the staff\_id. If we know a staff’s id number, we can find out their name, address details and position. We cannot determine these details without staff \_id. We assume that more than one staff can live in the same address, so eircode does not determine staff\_id or any other attributes.

**staff\_id —> first\_name**

**staff\_id —> last\_name**

**Staff\_id —> position**

**Staff\_id —> eircode**

### 5.8 Address: (Eircode, county, city, street)

Using the Eircode as a unique identifier for any address, we can identify the county city and street. We presume there is no dependency between city and county as several counties can have the same city or town (i.e. Holycross is in Co. Limerick and in Co Tipperary but not the same eircode).

**eircode —> county**

**eircode —> city**

**eircode —> street**

## Proof that each table is in 3NF.

To prove, that each relation is in Third Normal Form, we used the Bernstein synthesis algorithm.

We will prove that our database is in fact in 3NF. Firstly, you can see that all tables have atomic values, therefore there is no repetition of data; all tables have a primary key and tuples can be unique. Equally all non-prime attributes are fully functionally dependent on a primary key and there are no partial dependencies. There are no situations where a non-prime attribute is dependent on part of a candidate key attribute. Lastly, there are no transitive dependencies. There are no tables in which a non-prime attribute is transitively dependent on a candidate key attribute through another non-prime attribute. Infact, all our primary keys are on the left and there are no dependencies on attributes other than primary key attributes.

### 6.1. Student: (student\_id, first\_name, last\_name, fees\_paid, DoB, eircode)

**student\_id —> first\_name**

**student\_id —> last\_name**

**student\_id —> fees\_paid**

**student\_id —> DoB**

**student\_id —> eircode**

**Step 0: Finding the keys:**

*A Superkey is a set of attributes, that determines all attributes in a relation.*

*A Key is a minimal superkey.*

*A prime attribute is any attribute, that is present in the keys.*

The fact, that student\_id is the only attribute that is not on the RHS of any FDs, means that student\_id cannot be determined by any other attribute, therefore student\_id must be included in all keys.

If we find the closure of student\_id:

**{student\_id}+ = {student\_id, first\_name, last\_name, fees\_paid, DoB, eircode}**

We can see that all attributes are in the closure of student\_id. We already know that student\_id need to be in all keys, and the combination of any attribute and the student\_id won’t be minimal, therefore we concluded that student\_id is the only key in the student relation.

**Key: student\_id**

**Prime attributes: student\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side.

In the case of our Student relation, none of the attributes on the RHS are prime attributes (student\_id), but on the LHS of all FDs there is a key. Therefore, none of the attributes on the RHS can be transitively dependent on the primary key. This means they agree with the definition of the 3NF, they don't violate the rule. Therefore, as it is already in 3NF, we don’t need to decompose it.

### 6.2. Past\_Student: (student\_id, first\_name, last\_name, fees\_paid, DoB, Eircode)

**student\_id —> first\_name**

**student\_id —> last\_name**

**student\_id —> fees\_paid**

**student\_id —> DoB**

**student\_id —> eircode**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

The fact, that student\_id is not on the RHS of any FDs, means that student\_id cannot be determined by any other attribute; therefore student\_id must be included in all keys.

Find the closure of student\_id:

**{student\_id}+ = {student\_id, first\_name, last\_name, fees\_paid, DoB, eircode}**

We can see that all attributes are in the closure of student\_id. We already know that student\_id need to be in all keys, and the combination of any attribute and the student\_id won’t be minimal, we concluded that student\_id is the only key in the Past Student relation.

**Key: student\_id**

**Prime attributes: student\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our Past\_Student relation, none of the attributes on the RHS are prime attributes (student\_id), but on the LHS of all FDs there is a key. Furthermore, the RHS attributes are a subset of LHS meaning they are trivial FDs which do not provide real restrictions on the data in the relation and can be ignored. This means they agree with the definition of the 3NF, they don't violate the rule. Therefore, as it is already in 3NF, we don’t need to decompose it.

### 6.3. Enrollment: (student\_id, subject\_id)

**student\_id, subject\_id —> student\_id**

**student\_id, subject\_id —> subject\_id**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

As both attributes appear on the RHS in the enrolment table, we need to find the closures of each attribute first.

* Find the closure of subject\_id:

**{subject\_id}+ = {subject\_id}**

As the student\_id is not in the closure, subject\_id is not a superkey in its own right.

* Find the closure of student\_id:

**{student\_id}+ = {student\_id}**

As the subject\_id is not in the closure, student\_id is not a superkey in its own right.

* Find the closure of subject\_id and student\_id:

**{subject\_id student\_id}+ = {subject\_id, student\_id}**

We can see that all attributes are in the closure of the combination of subject\_id and student\_id. This means that our superkey is a composite key made of subject\_id and student\_id.

**Key: {student\_id, subject\_id}**

**Prime attributes: student\_id, subject\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our enrolment relation, all the attributes on the RHS are prime attributes (subject\_id or student\_id). Also, on the LHS of both FDs is the composite key. Furthermore, the RHS attributes are a subset of LHS meaning they are trivial FDs which do not provide real restrictions on the data in the relation and can be ignored. This means they agree with the definition of the 3NF, they don't violate the rule.

As it is already in 3NF, we don’t need to decompose it.

### 6.4. Subject: (subject\_id, subject\_name, subject\_fee)

In the subject relation, we assume that one subject can have multiple subject\_id’s to allow the school to charge different fees to residents and international students.

**subject\_id, —> subject\_name**

**subject\_id —> subject\_fee**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

The fact, that subject\_id is not on the RHS of any FDs, means that subject\_id cannot be determined by any other attribute, therefore subject\_id must be included in all keys.

Find the closure of subject\_id:

**{subject\_id}+ = {subject\_id, subject\_name, subject\_fee}**

We can see that all attributes are in the closure of subject\_id. We already know that subject\_id needs to be in all keys, and the combination of any attribute and the subject\_id won’t be minimal, we concluded that subject\_id is the only key in the Subject relation.

**Key: subject\_id**

**Prime attributes: subject\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our Subject relation, none of the attributes on the RHS are prime attributes (subject\_id), but on the LHS of all FDs there is a key. Therefore, none of the attributes on the RHS can be transitively dependent on the primary key. This means they agree with the definition of the 3NF, they don't violate the rule. As it is already in 3NF, we don’t need to decompose it.

### 6.5. Subject Allocation: (subject\_id, staff\_id)

**subject\_id, staff\_id —> staff\_id**

**subject\_id, staff\_id —> subject\_id**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

As in the Subject\_Allocation table, both attributes are appearing on the RHS, we need to find the closures of each attribute first.

* Find the closure of subject\_id:

**{subject\_id}+ = {subject\_id}**

Because the staff\_id is not in the closure, subject\_id is not a superkey on its own right.

* Find the closure of staff\_id:

**{staff\_id}+ = {staff\_id}**

Because the subject\_id is not in the closure, staff\_id is not a superkey on its own right.

* Find the closure of subject\_id and staff\_id:

**{subject\_id staff\_id}+ = {subject\_id, staff\_id}**

We can see that all attributes are in the closure of the combination of subject\_id and staff\_id. This means that our superkey is a composite key made of subject\_id and staff\_id.

**Key: {subject\_id, staff\_id}**

**Prime attributes: subject\_id, staff\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our Subject\_Allocation relation, all of the attributes on the RHS are prime attributes (subject\_id or staff\_id). Also, on the LHS of both FDs is the composite key. Therefore, none of the attributes on the RHS can be transitively dependent on the primary key. This means they agree with the definition of the 3NF, they don't violate the rule. As it is already in 3NF, we don’t need to decompose it.

### 6.6. ExamResults:(student\_id, subject\_id, score)

**student\_id, subject\_id —> score**

**student\_id, subject\_id —> subject\_id**

**student\_id, subject\_id —> student\_id**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

As all the attributes in the Exam Results table appear on the RHS, we need to find the closures of each attribute first.

* Find the closure of subject\_id:

**{subject\_id}+ = {subject\_id}**

As student\_id and score are not in the closure, subject\_id is not a superkey in its own right.

* Find the closure of score:

**{score}+ = {score}**

As subject\_id and student\_id are not in the closure, score is not a superkey in its own right.

* Find the closure of student\_id:

**{student\_id}+ = {student\_id}**

As score and subject\_id are not in the closure, student\_id is not a superkey in its own right.

* Find the closure of subject\_id and score:

**{subject\_id, score}+ = {subject\_id, score}**

As student\_id is not in the closure, subject\_id and score are not a superkey.

* Find the closure of score and student\_id:

**{score, student\_id}+ = {score, student\_id}**

As subject\_id is not in the closure, student\_id and score are not a superkey.

* Find the closure of subject\_id and student\_id:

**{subject\_id, student\_id}+ = {subject\_id, student\_id, score}**

We can see that all attributes are in the closure of the combination of subject\_id and student\_id. This means that our superkey is a composite key made of subject\_id and student\_id.

**Key: {student\_id, subject\_id}**

**Prime attributes: student\_id, subject\_id**

**Step 1: Decompose to 3NF tables:**

From looking at the closures we can see that neither subject\_id nor student\_id can be a key on their own. Therefore, there are no situations where attributes on the RHS could be partially dependent on any subset of the primary composite key. A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our ExamResults relation, not all the attributes on the RHS are prime attributes (subject\_id or student\_id), but on the LHS of all FDs are the composite key. Furthermore, some of the (subject\_id and student\_id) RHS attributes are a subset of LHS meaning they are trivial FDs which do not provide real restrictions on the data in the relation and can be ignored. This means they agree with the definition of the 3NF, they don't violate the rule. As it is already in 3NF, we don’t need to decompose it.

### 6.7. Staff: (staff\_id, first\_name, last\_name, position, Eircode)

**staff\_id —> first\_name**

**staff\_id —> last\_name**

**Staff\_id —> position**

**Staff\_id —> eircode**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

The fact, that staff\_id is not on the RHS of any FDs, means that staff\_id cannot be determined by any other attribute; therefore staff\_id must be included in all keys.

* Find the closure of staff\_id:

**{staff\_id}+ = {staff\_id, first\_name, last\_name, position, eircode}**

We can see that all attributes are in the closure of staff\_id. We already know that staff\_id needs to be in all keys, and the combination of any attribute and the staff\_id won’t be minimal, we concluded that staff\_id is the only key in the staff relation.

**Key: staff\_id**

**Prime attributes: staff\_id**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our Staff relation, none of the attributes on the RHS are prime attributes (staff\_id), but on the LHS of all FDs there are a key. Equally, if there are no FD’s on any attribute other than the whole primary key there can be no partial or transitive dependencies. This means they agree with the definition of the 3NF, they don't violate the rule. As it is already in 3NF, we don’t need to decompose it.

### 6.8. Address: (eircode, county, city, street)

In the Address relation, we assume that one city/town can be in different counties, therefore no functional dependency exists between city and county.

**eircode —> county**

**eircode —> city**

**eircode —> street**

**Step 0: finding the keys:**

*Superkey is a set of attributes, which determines all attributes in a relation.*

*Key is a minimal superkey.*

*Prime attribute is any attribute, what is present in the keys.*

The fact, that eircode is not on the RHS of any FDs means that eircode cannot be determined by any other attribute, therefore eircode must be included in all keys.

Find the closure of Eircode:

**{eircode}+ = {eircode, county, city, street}**

We can see that all attributes are in the closure of eircode. We already know that eircode needs to be in all keys, and the combination of any attribute and the eircode won’t be minimal, we concluded that eircode is the only key in the Address relation.

**Key: eircode**

**Prime attributes: eircode**

**Step 1: Decompose to 3NF tables:**

A relation is in 3NF, if each FD has either a superkey on the left-hand side, or a prime attribute on the right-hand side. In the case of our Address relation, none of the attributes on the RHS are prime attributes (eircode), but on the LHS of all FDs there is a key. Equally, if there are no FD’s on any attribute other than the whole primary key, there can be no partial or transitive dependencies. This means they agree with the definition of the 3NF, they don't violate the rule. As it is already in 3NF, we don’t need to decompose it.

## Justification for the usefulness of the views proposed in part B within a scenario for possible use of the views in an imaginary software system

### 7.1. Multiple\_Subject\_score\_over\_70

This view lists all students, who got at least 70% in at least two subjects. This view could help staff decide on granting scholarships to the high achieving students.

### 7.2. limerick\_students\_in\_arrears

This view lists the details of all students in arrears in Limerick. This view could help the administrative staff issuing letters of arrears to a certain group of students.

### 7.3. min5\_student\_failed\_per\_subject

This view lists the subject ID's and number of students where more than 5 students scored less than 40 in their exams results. This view could help staff pinpoint difficult subjects, where extra resources are needed, or can investigate teaching methods and recommend improvements.

## Analysis of the speed of the queries in your views and justification for the indexes proposed in part B

All the primary keys are indexed automatically. Indexes are useful for the maintenance of unique values. The DBMS can search the indexes easily and see if a new proposed value already exists or not.

The query in the “Multiple\_Subject\_score\_over\_70” view would benefit from the score being indexed. Since indexed entries are stored in a sorted order, the index would help when we are looking for the students with a score over 70. In our case, the database is small, but if we had thousands of students, each taking 5 or 6 exams, we could easily have over ten thousand tuples. In such cases, having the score indexed and sorted would speed up the query considerably, because the appropriate values will appear next to each other. The indexes are also slowing down the performance when the records are modified, due to the need to modify the index too. However, this won’t have much of a negative effect in our case, because the exam scores are seldom modified.

The query in the “limerick\_students\_in\_arrears” view would benefit from the fee\_paid attribute being indexed. Since indexed entries are stored in a sorted order, the index would help when we are looking for the students, who paid less than €1000. In our case, the database is small, but if we would have tens of thousands of students, having the fee\_paid attribute indexed and sorted would speed up the query considerably, because the appropriate values will appear next to each other. The indexes are also slowing down the performance when the records are modified, due to the need to modify the index too. In this case, when the fee\_paid is indexed, we must consider the pros and cons. The values in the fee\_paid attribute are being modified on a regular basis.

The query in the “min5\_student\_failed\_per\_subject” view would benefit from the score being indexed. Since indexed entries are stored in a sorted order, the index would help when we are looking for the students with score less than 40. In our case, the database is small, but if we would have thousands of students, each having 5 or 6 exams, we could easily have over ten thousand tuples. In such case, having the score indexed and sorted would speed up the query considerably, because the appropriate values will appear next to each other. The indexes are also slowing down the performance when the records are modified, due to the need to modify the index too. However, this won’t have much negative effect in our case, because the exam scores are seldomly modified.

## Justification for the necessity of the triggers and stored procedure and function proposed in part B within a scenario for possible use of the triggers within an imaginary software system

### 9.1. Procedure: studentAddToPast\_student

This procedure inserts student’s details in the past\_students table.

### 9.2. Trigger: delete\_student

This trigger calls the procedure. If a staff member tries to delete a student record, it is checked if the student is not currently enrolled to any subject. If not enrolled (graduated or dropped out), the student details are moved from the student table to the past\_students table. This trigger makes use of the procedure to insert new records to the past\_students table. The trigger is necessary to avoid current student records from being deleted accidentally. The procedure is recommended in order to maintain proper records and further traceability. The school does not want to lose details of past students, but for performance reasons, it wouldn’t make sense to keep all past students in the student table.

### 9.3. Function: number\_enrolled

The function returns the number of students currently enrolled on a specific subject.

### 9.4. Trigger: Discount\_popular\_subject

This trigger calls the function: The trigger activates, when on the given subject, there is at least 30 students enrolled. From that point, the subject fee is reduced by 10%. This is because we assume that in a larger class setting, the student experience diminishes compare to a smaller setting. Also, once the cost of running the subject is covered (in this case after 30 students), the school can afford to offer more seats on a discounted price while not increasing the overhead cost significantly. This discounted price might encourage additional students to enrol, generating additional profit for the school.