Database Notes
Online Notes
Reference Pages
Tutorial Activities
Online SQL
Online Quiz
Discussion Forum
Future Stuff
Online Relational Algebra
News

Lecture Slides available: PDF PowerPoint

Normalisation

Contents

- What is normalisation?
- Integrity Constraints
- Understanding Data
  - Student #2 - Flattened Table
- First Normal Form
- Flatten table and Extend Primary Key
  - Insertion anomaly:
  - Update anomaly
  - Deletion anomaly
- Decomposing the relation
- Second Normal Form
- Third Normal Form
- Summary: 1NF
- Summary: 2NF
- Summary: 3NF

What is normalisation?

Normalisation is the process of taking data from a problem and reducing it to a set of relations while ensuring data integrity and eliminating data redundancy

- Data integrity - all of the data in the database are consistent, and satisfy all integrity constraints.
- Data redundancy – if data in the database can be found in two different locations (direct redundancy) or if data can be calculated from other data items (indirect redundancy) then the data is said to contain redundancy.

Data should only be stored once and avoid storing data that can be calculated from other data already held in the database. During the process of normalisation redundancy must be removed, but not at the expense of breaking data integrity rules.

If redundancy exists in the database then problems can arise when the database is in normal operation:

- When data is inserted the data must be duplicated correctly in all places where there is redundancy. For instance, if two tables exist for in a database, and both tables contain the employee name, then creating a new employee entry requires that both tables be updated with the employee name.
- When data is modified in the database, if the data being changed has redundancy, then all versions of the redundant data must be updated simultaneously. So in the employee example a change to the employee name must happen in both tables simultaneously.

The removal of redundancy helps to prevent insertion, deletion, and update errors, since the data is only available in one attribute of one table in the database.

The data in the database can be considered to be in one of a number of `normal forms'. Basically the normal form of the data indicates how much redundancy is in that data. The normal forms have a strict ordering:

1. 1st Normal Form
2. 2nd Normal Form

3. 3rd Normal Form
4. BCNF

There are other normal forms, such as 4th and 5th normal forms. They are rarely utilised in system design and are not considered further here.

To be in a particular form requires that the data meets the criteria to also be in all normal forms before that form. Thus to be in 2nd normal form the data must meet the criteria for both 2nd normal form and 1st normal form. The higher the form the more redundancy has been eliminated.

Integrity Constraints

An integrity constraint is a rule that restricts the values that may be present in the database. The relational data model includes constraints that are used to verify the validity of the data as well as adding meaningful structure to it:

- entity integrity :

The rows (or tuples) in a relation represent entities, and each one must be uniquely identified. Hence we have the primary key that must have a unique non-null value for each row.

- referential integrity :

This constraint involves the foreign keys. Foreign keys tie the relations together, so it is vitally important that the links are correct. Every foreign key must either be null or its value must be the actual value of a key in another relation.

Understanding Data

Sometimes the starting point for understanding data is given in the form of relations and functional dependancies. This would be the case where the starting point in the process was a detailed specification of the problem. We already know what relations are. Functional dependancies are rules stating that given a certain set of attributes (the determinant) determines a second set of attributes.

The definition of a functional dependency looks like A->B. In this case B is a single attribute but it can be as many attributes as required (for instance, X->J,K,L,M). In the functional dependency, the determinant (the left hand side of the -> sign) can determine the set of attributes on the right hand side of the -> sign. This basically means that A selects a particular value for B, and that A is unique. In the second example X is unique and selects a particular set of values for J,K,L, and M. It can also be said that B is functionally dependent on A. In addition, a particular value of A ALWAYS gives you a particular value for B, but not vice-versa.

Consider this example:

R(matric_no, firstname, surname, tutor_number, tutor_name)

tutor_number -> tutor_name

Here there is a relation R, and a functional dependency that indicates that:

- instances of tutor_number are unique in the data
- from the data, given a tutor_number, it is always possible to work out the tutor_name.
- As an example tutor number 1 may be “Mr Smith”, but tutor number 10 may also be “Mr Smith”. Given a tutor number of 1, this is ALWAYS “Mr Smith”. However, given the name “Mr Smith” it is not possible to work out if we are talking about tutor 1 or tutor 10.

There is actually a second functional dependency for this relation, which can be worked out from the relation itself. As the relation has a primary key, then given this attribute you can determine all the other attributes in R. This is an implied functional dependency and is not normally listed in the list of functional dependents.

Extracting understanding

It is possible that the relations and the determinants have not yet been defined for a problem, and therefore must be calculated from examples of the data. Consider the following Student table.

Student - an unnormalised tablewith repeating groups

matric_no	Name	date_of_birth	subject	grade
960100	Smith, J	14/11/1977	Databases Soft_Dev ISDE	C A D
960105	White, A	10/05/1975	Soft_Dev ISDE	B B
960120	Moore, T	11/03/1970	Databases Soft_Dev Workshop	A B C
960145	Smith, J	09/01/1972	Databases	B
960150	Black, D	21/08/1973	Databases Soft_Dev ISDE Workshop	B D C D

The subject/grade pair is repeated for each student. 960145 has 1 pair while 960150 has four. Repeating groups are placed inside another set of parentheses. From the table the following relation is generated:

Student(matric_no, name, date_of_birth, (subject, grade))

The repeating group needs a key in order that the relation can be correctly defined. Looking at the data one can see that grade repeats within matric_no (for instance, for 960150, the student has 2 D grades). However, subject never seems to repeat for a single matric_no, and therefore is a candidate key in the repeating group.

Whenever keys or dependencies are extracted from example data, the information extracted is only as good as the data sample examined. It could be that another data sample disproves some of the key selections made or dependencies extracted. What is important however is that the information extracted during these exercises is correct for the data being examined.

Looking at the data itself, we can see that the same name appears more than once in the name column. The name in conjunction with the date_of_birth seems to be unique, suggesting a functional dependency of:

name, date_of_birth -> matric_no

This implies that not only is the matric_no sufficient to uniquely identify a student, the student's name combined with the date of birth is also sufficient to uniquely identify a student. It is therefore possible to have the relation Student written as:

Student(matric_no, name, date_of_birth, (subject, grade))

As guidance in cases where a variety of keys could be selected one should try to select the relation with the least number of attributes defined as primary keys.

Flattened Tables

Note that the student table shown above explicitly identifies the repeating group. It is also possible that the table presented will be what is called a flat table, where the repeating group is not explicitly shown:

Student #2 - Flattened Table

matric_no	name	date_of_birth	Subject	grade

960100	Smith, J	14/11/1977	Databases	C
960100	Smith, J	14/11/1977	Soft_Dev	A
960100	Smith, J	14/11/1977	ISDE	D
960105	White, A	10/05/1975	Soft_Dev	B
960105	White, A	10/05/1975	ISDE	B
960120	Moore, T	11/03/1970	Databases	A
960120	Moore, T	11/03/1970	Soft_Dev	B
960120	Moore, T	11/03/1970	Workshop	C
960145	Smith, J	09/01/1972	Databases	B
960150	Black, D	21/08/1973	Databases	B
960150	Black, D	21/08/1973	Soft_Dev	D
960150	Black, D	21/08/1973	ISDE	C
960150	Black, D	21/08/1973	Workshop	B

The table still shows the same data as the previous example, but the format is different. We have removed the repeating group (which is good) but we have introduced redundancy (which is bad).

Sometimes you will miss spotting the repeating group, so you may produce something like the following relation for the Student data.

Student(matric_no, name, date_of_birth, subject, grade)

matric_no -> name, date_of_birth
name, date_of_birth -> matric_no

This data does not explicitly identify the repeating group, but as you will see the result of the normalisation process on this relation produces exactly the same relations as the normalisation of the version that explicitly does have a repeating group.

First Normal Form

- First normal form (1NF) deals with the `shape' of the record type
- A relation is in 1NF if, and only if, it contains no repeating attributes or groups of attributes.
- Example:
- The Student table with the repeating group is not in 1NF
- It has repeating groups, and it is called an `unnormalised table'.

Relational databases require that each row only has a single value per attribute, and so a repeating group in a row is not allowed.

To remove the repeating group, one of two things can be done:

- either flatten the table and extend the key, or
- decompose the relation- leading to First Normal Form

Flatten table and Extend Primary Key

The Student table with the repeating group can be written as:

Student(matric_no, name, date_of_birth, (subject, grade))

If the repeating group was flattened, as in the Student #2 data table, it would look something like:

Student(matric_no, name, date_of_birth, subject, grade)

Although this is an improvement, we still have a problem. matric_no can no longer be the primary key - it does not have an unique value for each row. So we have to find a new primary key - in this case it has to be a compound key since no single attribute can uniquely identify a row. The new primary key is a compound key (matrix_no + subject).

We have now solved the repeating groups problem, but we have created other complications. Every repetition of the matric_no, name, and data_of_birth is redundant and liable to produce errors.

With the relation in its flattened form, strange anomalies appear in the system. Redundant data is the main cause of insertion, deletion, and updating anomalies.

Insertion anomaly:

With the primary key including subject, we cannot enter a new student until they have at least one subject to study. We are not allowed NULLs in the primary key so we must have an entry in both matric_no and subject before we can create a new record.

- This is known as the insertion anomaly. It is difficult to insert new records into the database.
- On a practical level, it also means that it is difficult to keep the data up to date.

Update anomaly

If the name of a student were changed for example Smith, J. was changed to Green, J. this would require not one change but many one for every subject that Smith, J. studied.

Deletion anomaly

If all of the records for the `Databases' subject were deleted from the table,we would inadvertently lose all of the information on the student with matric_no 960145. This would be the same for any student who was studying only one subject and the subject was deleted. Again this problem arises from the need to have a compound primary key.

Decomposing the relation

- The alternative approach is to split the table into two parts, one for the repeating groups and one of the non-repeating groups.
- the primary key for the original relation is included in both of the new relations

Record

matric_no	**subject**	**grade**
960100	Databases	C
960100	Soft_Dev	A
960100	ISDE	D

960105	Soft_Dev	B
960105	ISDE	B
...	...	...
960150	Workshop	B

Student

matric_no	name	date_of_birth
960100	Smith,J	14/11/1977
960105	White,A	10/05/1975
960120	Moore,T	11/03/1970
960145	Smith,J	09/01/1972
960150	Black,D	21/08/1973

- We now have two relations, Student and Record.
- Student contains the original non-repeating groups
- Record has the original repeating groups and the matric_no

Student(matric_no, name, date_of_birth)
Record(matric_no, subject, grade)

Matric_no remains the key to the Student relation. It cannot be the complete key to the new Record relation - we end up with a compound primary key consisting of matric_no and subject. The matric_no is the link between the two tables - it will allow us to find out which subjects a student is studying . So in the Record relation, matric_no is the foreign key.

This method has eliminated some of the anomalies. It does not always do so, it depends on the example chosen

- In this case we no longer have the insertion anomaly
- It is now possible to enter new students without knowing the subjects that they will be studying
- They will exist only in the Student table, and will not be entered in the Record table until they are studying at least one subject.
- We have also removed the deletion anomaly
- If all of the `databases' subject records are removed, student 960145 still exists in the Student table.
- We have also removed the update anomaly

Student and Record are now in First Normal Form.

Second Normal Form

Second normal form (or 2NF) is a more stringent normal form defined as:

A relation is in 2NF if, and only if, it is in 1NF and every non-key attribute is fully functionally dependent on the whole key.

Thus the relation is in 1NF with no repeating groups, and all non-key attributes must depend on the whole key, not just some part of it. Another way of saying this is that there must be no partial key dependencies (PKDs).

The problems arise when there is a compound key, e.g. the key to the Record relation - matric_no, subject. In this case it is possible for non-key attributes to depend on only part of the key - i.e. on only one of the two key attributes. This is what 2NF tries to prevent.

Consider again the Student relation from the flattened Student #2 table:

```
Student(matric_no, name, date_of_birth, subject, grade )
```

- There are no repeating groups
- The relation is already in 1NF
- However, we have a compound primary key - so we must check all of the non-key attributes against each part of the key to ensure they are functionally dependent on it.
- matric_no determines name and date_of_birth, but not grade.
- subject together with matric_no determines grade, but not name or date_of_birth.
- So there is a problem with potential redundancies

A dependency diagram is used to show how non-key attributes relate to each part or combination of parts in the primary key.

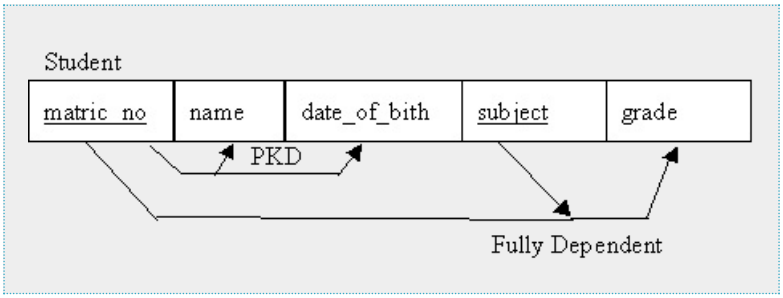
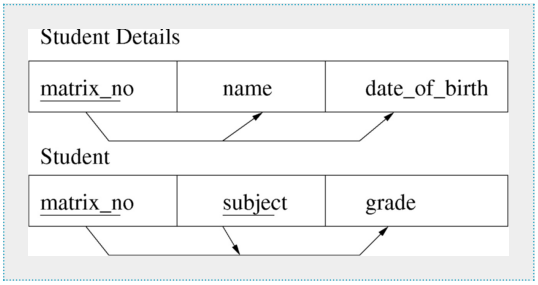


Figure : Dependency Diagram

- This relation is not in 2NF
- It appears to be two tables squashed into one.
- the solution is to split the relation up into its component parts.
- separate out all the attributes that are solely dependent on matric_no
- put them in a new Student_details relation, with matric_no as the primary key
- separate out all the attributes that are solely dependent on subject.
- in this case no attributes are solely dependent on subject.
- separate out all the attributes that are solely dependent on matric_no + subject
- put them into a separate Student relation, keyed on matric_no + subject



All attributes in each relation are fully functionally dependent upon its primary key

These relations are now in 2NF

Figure : Dependencies after splitting

Interestingly this is the same set of relations as when we recognized that there were repeating terms in the table and directly removed the repeating terms. It should not really matter what process you followed when normalizing, as the end result should be similar relations.

Third Normal Form

3NF is an even stricter normal form and removes virtually all the redundant data :

- A relation is in 3NF if, and only if, it is in 2NF and there are no transitive functional dependencies
- Transitive functional dependencies arise:
- when one non-key attribute is functionally dependent on another non-key attribute:
- FD: non-key attribute -> non-key attribute
- and when there is redundancy in the database

By definition transitive functional dependency can only occur if there is more than one non-key field, so we can say that a relation in 2NF with zero or one non-key field must automatically be in 3NF.

project_no	**manager**	**address**
p1	Black,B	32 High Street
p2	Smith,J	11 New Street
p3	Black,B	32 High Street
p4	Black,B	32 High Street

Project has more than one non-key field so we must check for transitive dependency:

- address depends on the value in the manager column
- every time B Black is listed in the manager column, the address column has the value `32 High Street'. From this the relation and functional dependency can be implied as:

  Project(project_no, manager, address)

  manager -> address
- in this case address is transitively dependent on manager. Manager is the determinant - it determines the value of address. It is transitive functional dependency only if all attributes on the left of the “->” are not in the key but are all in the relation, and all attributes to the right of the “->” are not in the key with at least one actually being in the relation.
- Data redundancy arises from this
- we duplicate address if a manager is in charge of more than one project
- causes problems if we had to change the address- have to change several entries, and this could lead to errors.
- The solution is to eliminate transitive functional dependency by splitting the table
- create two relations - one with the transitive dependency in it, and another for all of the remaining attributes.
- split Project into Project and Manager.
- the determinant attribute becomes the primary key in the new relation
- manager becomes the primary key to the Manager relation
- the original key is the primary key to the remaining non-transitive attributes
- in this case, project_no remains the key to the new Projects table.

Project

project_no	**manager**
p1	Black,B
p2	Smith,J
p3	Black,B
p4	Black,B

Manager

manager	**address**
Black,B	32 High Street

Smith,J	11 New Street

- Now we need to store the address only once
- If we need to know a manager's address we can look it up in the Manager relation
- The manager attribute is the link between the two tables, and in the Projects table it is now a foreign key.
- These relations are now in third normal form.

Summary: 1NF

- A relation is in 1NF if it contains no repeating groups
- To convert an unnormalised relation to 1NF either:
- Flatten the table and change the primary key, or
- Decompose the relation into smaller relations, one for the repeating groups and one for the non-repeating groups.
- Remember to put the primary key from the original relation into both new relations.
- This option is liable to give the best results.

Summary: 2NF

- A relation is in 2NF if it contains no repeating groups and no partial key functional dependencies
- Rule: A relation in 1NF with a single key field must be in 2NF
- To convert a relation with partial functional dependencies to 2NF. create a set of new relations:
- One relation for the attributes that are fully dependent upon the key.
- One relation for each part of the key that has partially dependent attributes

Summary: 3NF

- A relation is in 3NF if it contains no repeating groups, no partial functional dependencies, and no transitive functional dependencies
- To convert a relation with transitive functional dependencies to 3NF, remove the attributes involved in the transitive dependency and put them in a new relation
- Rule: A relation in 2NF with only one non-key attribute must be in 3NF
- In a normalised relation a non-key field must provide a fact about the key, the whole key and nothing but the key.
- Relations in 3NF are sufficient for most practical database design problems. However, 3NF does not guarantee that all anomalies have been removed.

Next Page Up One Level