BookName:Normalisation

normalization
normalization

Normalisation

The base of the relational database management system is the normalisation –meaning a method which gives the optimum way of the data's placement. In case of an inefficiently designed database there will be contradictions and anomalies in the data structure. Normalisation allows you to structure data appropriately, and it helps you to eliminate the anomalies and lower data redundancy.

Anomalies:

- Insertion anomaly: Adding a record wishes another record's enrolment which is not logically related to the record.
- Deletion anomaly: During deleting the item some instance of data is removed as well
- Update anomaly: Because of a change of a data we have to update it at its every place of occurrence

Normal Forms:

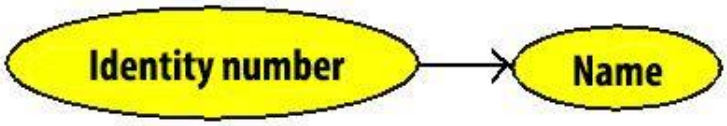
First Normal Form: there are no repeating elements or groups of elements. In every row of the relations one and only one value takes place in a column, the order of the values is the same in every row, and every row is different. There is always at least one or more feature(s) which make(s) the rows individually distinguishable.

Second Normal Form: the relation is in first normal form, and none of its secondary attributes depend on any of the genuine subset of its keys. (Primary attributes are the ones which belong to a key; in case of secondary attributes this is not true.)

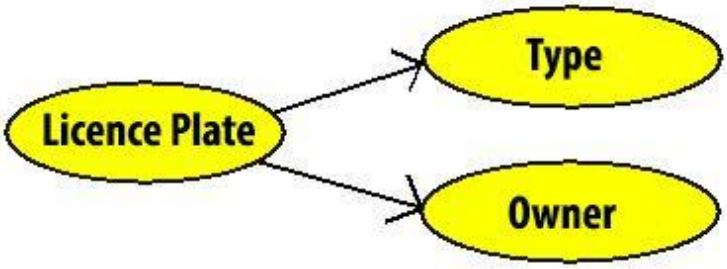
Third Normal Form: the relation is in second normal form and there is no functional dependence among the secondary attributes. If the value of "B" attribute depends on the value of "A" attribute, and the value of "C" attribute transitively depends on the value of "A" attribute. Elimination of these transitively dependences is an essential requirement of the third normal form. If the table of the database is not in third normal form we have to divide it into two tables, each of them in third normal form.

Dependences

Functional dependency: when any values of a feature of the system can be assigned to only one value of another feature. E.g.: one personal identity number can be connected to one person but a person is able to have more personal identity numbers.



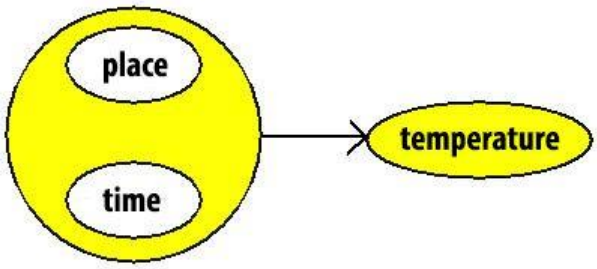
One-to-many relationship.



Mutual Functional Dependency: when the above-mentioned requirements come true in both directions. e.g.: registration number- number of the engine. One-to-one relationship.

Functionally Independents: when the above-mentioned requirements don't come true. e.g.: the colour of the student's eyes – the place of their school.

Transitive Functional Dependency: when some concrete values of a describing feature of an element determine other values of a describing feature.



Relation key

The relation key unambiguously identifies a row of the relation. The relation – as it is in the definition – cannot include two identical rows. Therefore, there is a key in every relation. The relation key must carry out the following terms:

- it is a group of such attributes, that identifies only one row (unambiguously)
- none of the attributes that are included in the key can form subset
- the value of attributes that are included in the key cannot undefined (NULL)

The storage of undefined (NULL) values is being specially solved by the relation database managers. In case of numerical values, the value of NULL and 0 are not equals.

Let's keep a record of the personal data of students of the class in a relation.

Id number	Date of birth	Name

PERSONAL_DATA=({ ID_NUMBER, DATE_OF_BIRTH, NAME}).

In the PERSONAL_DATA relation the ID_NUMBER attribute is a key. It is, because there cannot be two different people with same id numbers. The date of birth or the name cannot identify unambiguously a row of the relation, because there have been born students on the same day or there may be students with same names in the class. Together they identify a row of the relation. But they cannot satisfy the condition related to the keys that the subset of the attribute, that is included in them, cannot be a key. In this case, the id number is already a key. This way, combined it with any other attribute it cannot form key already.

There might also be such relation that in it the key can be formed by connecting more value for the attributes. Let's make a record of the given marks the students got with the following relation:

DIARY=({ID_NUMBER, SUBJECT, DATE, MARK)}

Id number	Subject	Date	Mark

In the DIARY relation the ID_NUMBER does not identify a row, because there can be marks for a student, even from the same subject. Because of this, even the ID_NUMBER and the SUBJECT cannot form key. Even the ID_NUMBER, SUBJECT and the DATE can only form key if we preclude the possibility of that that a student can get two marks from the same subject on the same day. In this case, if that condition could not be kept, then there must be stored not only the acquisition date of the mark, but also its point of time. In such cases the DIARY relation has to be extended with that new column. There are not just complex keys that can take place in the relation. There are also such relations that in it there can be found not just one, but more keys. To illustrate this let's see the next relation.

Consultation=({Teacher, POINT_OF_TIME, STUDENT)}

Teacher	**Point of time**	**Student**

Relation with more keys

In the CONSULTATION relation we imagine such identifier in the teacher and student columns that unambiguously identify the person (for example ID number). Every single student can take part in more consultation, and every teacher can hold more consultations. What is more, the same student can take part in the same teacher's consultation in different points of time. As a consequence, neither the TEACHER nor the STUDENT nor the two identifiers together are keys of the relation. But one person in one time can only be in one place. As a consequence the TEACHER, POINT_OF_TIME attributes are forming key, and with the same reasons the STUDENT, POINT_OF_TIME attributes are forming key as well. We have to notice that the keys are not being made by as a result of arbitrary decisions, but they come from the nature of the data as well as the functional dependence or the polyvalent dependence. In the relation we differentiate foreign/outer key, too. These attributes form key not in the certain relation, but in another relation of the database. For example, in the CONSULTATION relation if we use the ID number to the identification of the STUDENT then it is a foreign key to the relation record personal data.

Data model mistakes

Anomalies:

They are mistakes, because of inadequately designed data model. They may lead to the inconsistency of the database (because we are not storing only one entity's features or we store certain features multiple times).

Types

- insertion anomaly: The entry of new record cannot be done to one table, because in the table there are such attribute values which are available during entry or not available even later.
- modification anomaly: We store one in more tables, but during the modification of the attribute values we have not done the modification everywhere, or we have not done it the same way.
- deletion anomaly: We are deleting in a table and we are losing such important information that we would need later.

Redundancy:

It means overlap. In practice we refer physical overlapping to it – multiple data storage in the database – at designing it is also important paying attention to the logical overlaps.

Types

- Logical overlap:

Open logical overlap: the same attribute type with the same name is included in more entities. It results multiple storage. They may be necessary due to safety or efficiency, or for example to carry out connections (as foreign key). The lack of the logical overlap is also count as a mistake.

Hidden logical overlap (synonym phenomenon): We mark the same attribute with different name.

Apparent logical overlap (homonym phenomenon): We use the same name to different attributes.

- Physical overlap: the multiple store of the same attribute or – with synonym name – entity in the database.

Let's see the next relation.

Teacher	Subject	Total_number_of_lessons	Lessons_taught
Kiss Péter	Database management	64	12
Nagy Andrea	Mathematics	32	8
Szabó Miklós	Database management	64	4
Kovács Rita	Mathematics	32	5
	English	48	

Relation that contains redundancy

In the above mentioned relation we store the total number of lessons as many times as many teacher are teaching the certain subject. For example, let us suppose that a subject is being taught by more teachers. The redundancy has the following disadvantages:

- If the total number of lessons of a subject is changing, it has to be modified in the relation.
- Every time when a new teacher gets in the relation the total number of lessons data has to be taken out from the previous rows of the same subject.
- In the case of the subject in the last row (English) it has not been filled out who the teacher is. During the inclusion of new teacher to the list this case has to be managed in another way. In this case we just have to rewrite two empty values.

Redundancy can also occur if we store derived or derivable quantity in the relation.

A single relation can also contain derived data in that case if the value of certain attributes can be unambiguously determined based on the rest of the attribute. For example if we recorded the district beside the postal code. There are two methods to stop the redundant data. We have

to leave those relations or attributes that contain derived data. The redundant facts that are being stored in relations can be ended by taking the table apart, but we are doing it with its composition. We take to two pieces the relation that is in the 3.10 example
Lessons = {Teacher, Subject, Lessons_taught} and Total_number_of_lessons = {Subject, Total_number_of_lessons }

Stopping the redundancy

The goal of the logical design is a relation system, relation database without redundancy. The relation theory contains methods to stop redundancy with the help of the so called normal forms. From now on we will shot the definition of normal forms of the relations through examples. We will use notion of functional dependence, multivalued dependence and the relation key to make normal forms. During forming of normal form the goal is simply to write down such relations that we store facts which are related to the relation key. We differentiate five normal forms. The different normal forms build upon each other. The relation in the second normal form is also in first normal form. During designing the goal is to reach the biggest normal form. The first three normal forms concentrate on stopping redundancies in functional dependences. The fourth and fifth normal forms concentrate on stopping redundancies in multivalued dependences.
We have to get acquainted with two new notions that are connected to the relations. We call primary attributes that are at least in one relation key. The other attributes are called not primary.

Normal forms:

First normal form: All values are elementary in the relation. The relation cannot include data group. In every row per column of the relation there can be only one value. In every row the order of values are the same. All rows are different. There is at least one or more attribute that the rows can be unambiguously differentiated from each other.
For example, let it be here such kind of a relation that the attributes of it are also relations.

Study group	**Teacher**	**Students**	
Computer technology	Nagy Pál	**Name**	**Class**

		Kiss Rita III.b Álmos Éva II.c
Video	Gál János	**Name** **Class** Réz Ede I.a Vas Ferenc II.b

Study group	**Teacher**	**Student**	**Class**
Computer technology	Nagy Pál	Kiss Rita	III.b
Computer technology	Nagy Pál	Álmos Éva	II.c
Video	Gál János	Réz Ede	I.a
Video	Gál János	Vas Ferenc	II.b

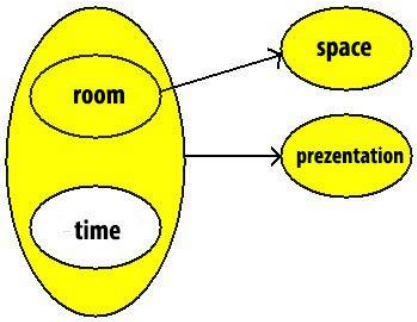
Second normal form: The relation is n first normal form. Furthermore, none of its secondary attributes depend on any of its key's subset. (The primary attributes are those attribute that belonging to some of the keys. Those attributes, which are not belonging to any of the keys, are secondary attributes.)

Conference			
Room	**Point of time**	**Presentation**	**Place**
B	10:00	Mythology	250
A	8:30	Literature	130
B	11:30	Theater	250
A	11:00	Painting	130
A	13:15	Archeology	130

Conference		
Room	**Point of time**	**Presentation**
B	10:00	Mythology
A	8:30	Literature

B	11:30	Theater
A	11:00	Painting
A	13:15	Archeology

Conference		
Room	**Point of time**	**Presentation**
B	10:00	Mythology
A	8:30	Literature
B	11:30	Theater
A	11:00	Painting
A	13:15	Archeology



Dependency diagram

Let’s see another example for the relation that breaks the term of the second normal form. For the check of the energy management of a building the temperature in the certain rooms is being regularly measured. For the evaluation of the measured results we are also recording the number of radiators in the certain rooms.

Temperature			
Room	**Point of time**	**Temperature**	**Radiator**
213	98.11.18	23	2

213	98.11.24	22	2
213	98.12.05	21	2
214	98.12.05	21	3
214	98.12.15	20	3

Conference		
Room	**Point of time**	**Temperature**
213	98.11.18	23
213	98.11.24	22
213	98.12.05	21
214	98.12.05	21
214	98.12.15	20

Rooms	
Room	**Radiator**
213	2
214	3

Third normal form:

The relation is in second normal form. Furthermore, there are not any functional dependence among the secondary attributes. If the value of attribute “B” depends on the value of attribute “A” as well as the value of attribute “C” transitively depends on the value of “A”. The elimination of such transitive dependences is inevitable requirements of the third normal form. If the table of the database is not in third normal form then it must be broken to two tables so that the certain tables separately are in third normal form.
This will be demonstrated with the help of an example again.

Study groups		
Study group	**Teacher**	**Date of birth**
Képzőművész	Sár Izodor	1943
Iparművész	Sár Izodor	1943
Karate	Erős János	1972

Study groups	
Study group	**Teacher**
Képzőművész	Sár Izodor
Iparművész	Sár Izodor
Karate	Erős János

Teachers	
Teacher	**Date of birth**
Erős János	1972
Sár Izodor	1943

Boyce/Codd normal form (BCNF)

During the discussion of the normal forms we showed examples to such relations which only have one relation key. Of course, the definition of the normal forms can be applied to those relations that have more keys. In this case, every attribute that is part some of the keys are primary attribute. But this attribute can depend on another key that does not include it as part of the key. If that is the case then the relation contains redundancy. The recognition of this led to a more strict definition of the third normal form that is called Boyce/Codd normal form.

• All primary attributes are in complete functional dependence with those keys that this is not part of it.

As an example, let's see the following relation:

Subjects

Teacher	Point_of_time	Subject	Semester	Number_of_students
Kiss Pál	93/1	Database	1	17
Jó Péter	93/1	Unix	1	21
Kiss Pál	93/2	Database	2	32
Jó Péter	93/1	Unix	2	19
Kiss Pál	93/1	Database	3	25

The relation's third normal form and the decomposition of the Boyce/Codd normal form

Let us suppose that every teacher is teaching only one subject, but they are teaching it in different semesters. Based on this the following functional dependence can be written down: Teacher, Semester Subject, and Semester Teacher. The relation has two keys. They are the (Teacher, Point of time, Semester) and the (Subject, Point of time, Semester). In the relation there is only one non-primary attribute which is the Number_of_students. That is complete functional dependence with both of the relation keys. There is no dependence relation between the primary attributes. Based on these the relation is in third normal form. However, it contains redundancy, because beside the same teacher we store the subject multiple times in the same points of time. The reason for the redundancy is due to the fact that the teacher attribute depends on the relation key that does not include the teacher attribute (Subject, Point of time, Semester) only the part of it (Subject, Semester).

Fourth normal form (4NF)

Unfortunately, even the Boyce/Codd normal form can contain redundancy. Up to this point we have only examined the functional dependences, but not the polyvalent dependences. The following two normal forms serve to eliminate the redundancy from polyvalent dependences. A relation is in fourth normal form if in an XY polyvalent dependence it only contains those attributes that can be found in X and Y.

Fifth normal form (5NF)

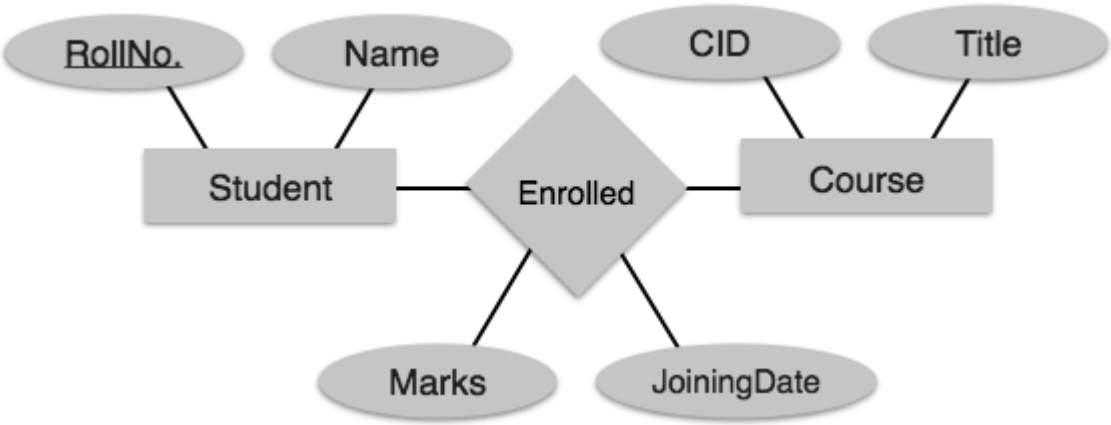
For a long time the fourth normal form was considered the last step of the normalization. However, we can lose information due to the storage of the polyvalent dependences in separate relations. Let's see an example to show this. An Ltd, which is specialized in teaching computer knowledge, has more, well qualified teachers. The teachers are suitable for educate in various courses. The courses are being held in different parts of the country.
Let's make the Teacher-Course-Place relation based on these facts.
The only relation key contains all of the attributes (Teacher, Course, Place). It comes from that the relation is in Boyce/Codd normal form, but despite of that, it contains redundancy. For example, there can be found in two rows that Kiss Pál is teaching Database 1 course. By breaking the relation down in two - that only contains one polyvalent dependence – relation (Teacher, Course) and (Teacher, Place). If we stop the redundancy, that would also lead to information loss. After the break down we already do not know which subject is being taught in the certain place by the teacher. For example, is Kiss Éva holding database I. or database II. course in Pécs. Breaking the original relation in three, we will get the fifth normal form. The original relations can be produced from the connection of three relations that we have got as a result, but the connection of any of the two relations is not enough. At the end of the discussion of the normal forms we mention that it is recommended to normalize the relations by all means up to the third normal form. This eliminates much of the redundancy. Those cases are rarer that requires the use of the fourth and fifth normal form. In the case of the fifth normal form, it is possible to stop the redundancy we use bigger storage space. So, the designer of the database can decide whether he/she choose the fifth normal form and the bigger database, or the redundancy and the more complicated refresh and modification algorithms.

Physical designing

In case of the relation database, during the logical designing the relations can already adopt their final form which can be easily formed down in the database manager. During the physical designing we are rather concentrating on that, is the logical structure suitable for the terms of the effective implementation, and what indexes should we summon to the certain relation. The jointly implemented operations on the relation are called transaction. In general, we would like to reach fast implementation of transactions.

Mapping Relationship

A relationship is an association among entities.



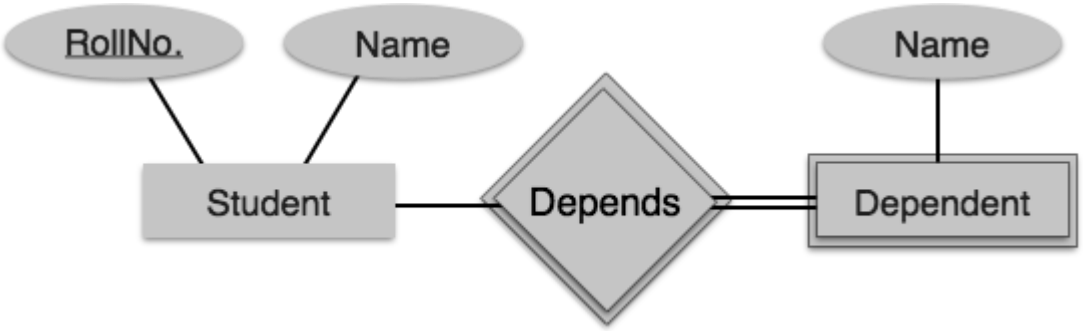
[*Image: Mapping relationship*]

Mapping Process:

- Create table for a relationship.
- Add the primary keys of all participating Entities as fields of table with their respective data types.
- If relationship has any attribute, add each attribute as field of table.
- Declare a primary key composing all the primary keys of participating entities.
- Declare all foreign key constraints.

Mapping Weak Entity Sets

A weak entity set is one which does not have any primary key associated with it.



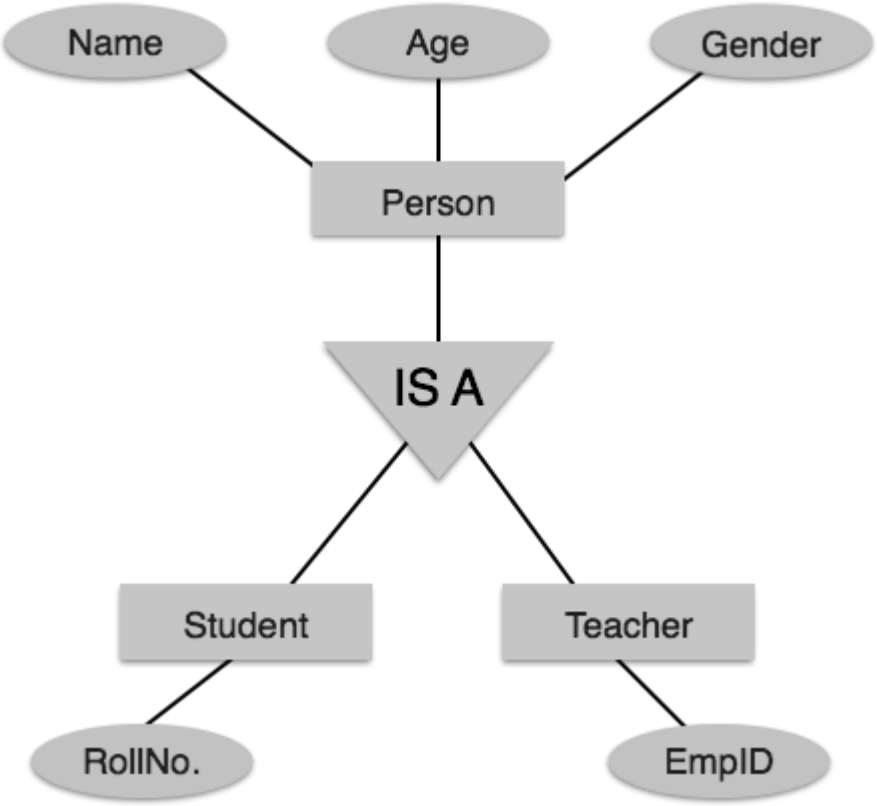
[*Image: Mapping Weak Entity Sets*]

Mapping Process:

- Create table for weak entity set.
- Add all its attributes to table as field.
- Add the primary key of identifying entity set.
- Declare all foreign key constraints.

Mapping Hierarchical Entities

ER specialization or generalization comes in the form of hierarchical entity sets.



[*Image: Mapping hierarchical entities*]

Mapping Process

- Create tables for all higher-level entities.
- Create tables for lower-level entities.
- Add primary keys of higher-level entities in the table of lower-level entities.
- In lower-level tables, add all other attributes of lower-level entities.
- Declare primary key of higher-level table and the primary key for lower-level table.

- Declare foreign key constraints.

13. SQL OVERVIEW

SQL is a programming language for Relational Databases. It is designed over relational algebra and tuple relational calculus. SQL comes as a package with all major distributions of RDBMS.

SQL comprises both data definition and data manipulation languages. Using the data definition properties of SQL, one can design and modify database schema, whereas data manipulation properties allows SQL to store and retrieve data from database.

Data Definition Language

SQL uses the following set of commands to define database schema:

CREATE

Creates new databases, tables, and views from RDBMS.

For example:

```
Create database tutorialspoint;
Create table article;
Create view for_students;
```

DROP

Drops commands, views, tables, and databases from RDBMS.

For example:

```
Drop object_type object_name;
Drop database tutorialspoint;
Drop table article;
Drop view for_students;
```

ALTER

Modifies database schema.

```
Alter object_type object_name parameters;
```

For example:

```
Alter table article add subject varchar;
```

This command adds an attribute in the relation **article** with the name **subject** of string type.

Data Manipulation Language

SQL is equipped with data manipulation language (DML). DML modifies the database instance by inserting, updating, and deleting its data. DML is responsible for all forms data modification in a database. SQL contains the following set of commands in its DML section:

- SELECT/FROM/WHERE
- INSERT INTO/VALUES
- UPDATE/SET/WHERE
- DELETE FROM/WHERE

These basic constructs allow database programmers and users to enter data and information into the database and retrieve efficiently using a number of filter options.

SELECT/FROM/WHERE

- **SELECT**

  This is one of the fundamental query command of SQL. It is similar to the projection operation of relational algebra. It selects the attributes based on the condition described by WHERE clause.

- **FROM**

  This clause takes a relation name as an argument from which attributes are to be selected/projected. In case more than one relation names are given, this clause corresponds to Cartesian product.

- **WHERE**

  This clause defines predicate or conditions, which must match in order to qualify the attributes to be projected.

For example:

```
Select author_name
From book_author
Where age > 50;
```