Geographic Information Technology Training Alliance (GITTA) presents:

The relational database model

Responsible persons: Anca Dobre, Dominique Schneuwly, Susanne Bleisch

Table Of Content

1. The relational database model

The relational database model is used in most of today's commercial databases. It is used since the early 80ies and was developed 1970 by E. F. Codd. The relational database model is based on a mathematical concept where relations are interpreted as tables.

The focus of this lesson lies in the conversion of a conceptual into a logical data scheme (the relational database model) using an entity-relationship-schema. You will find more information about schemas in the lesson about **Database models, schemas and instances**.

1.1. Concept of the relational model

In contrast to the entity-relationship-model (ERM) which is a conceptual model, the relational model is a logical data model. It can be seen as lying one step or layer below the ERM. The relational model is not about abstract objects but defines how data should be represented in a specific database management system. The goal of a logical data model is to arrange the data in such a form that it is consistent, non-redundant and supports operations for data manipulation.

1.1.1. Data organization in a relational data model

A logical data schema (model) is in most cases based on a conceptual data scheme which, with the use of certain guidelines and rules, is transformed into a relational scheme (model). The main organization unit in a relational data model is the relation. A relation can be represented as a table but the definition of the relation is not necessarily equal to the definition of the table and vica versa.

Why use the relational model?

- **Simplicity.** Data in a relational model is represented through values that are structured with only one construct: the "relation".
- **Classification.** The relational model is based on mathematical fundamentals: the set theory.

1.1.2. Definitionen

domain:

A domain **D** is a set of atomic values that defines the value range of attributes.

Examples:

- Value = {1'000.-, ..., 9'999.-}
- Date = {1.1.1900, ..., 12.12.1999}
- Character = {'a', ... 'z', 'A', ..., 'Z'}

Example domain

tupel:

A tuple **t** is a list with n values **t = <d1, d2, ..., dn>** where each value di is either an element of the domain **Di** or NULL. A tuple is a record in a relation (row in a table).

Examples:

- t1 = <Mueller, 23.3.1965, 6'500.->
- t2 = <Meier, 27.7.1963, 2'300.->

Example tupel

Attribute:

A column of a table represents an attribute. It can also be described as a that a domain D has in arelation scheme R.

Example

In the description for a person the domain "value" could mean the income, the domain "date" could mean the date of birth and the domain "chain of characters" could mean the name of the person. E.g., the attribute "income" would map t1 to 6'500 and t2 to 2'300.

Example attribute

Relation scheme:

A relation scheme **R** (A1, A2, ..., An) is made up of a relation name R and a list of attributes {A1, A2, ..., An}.

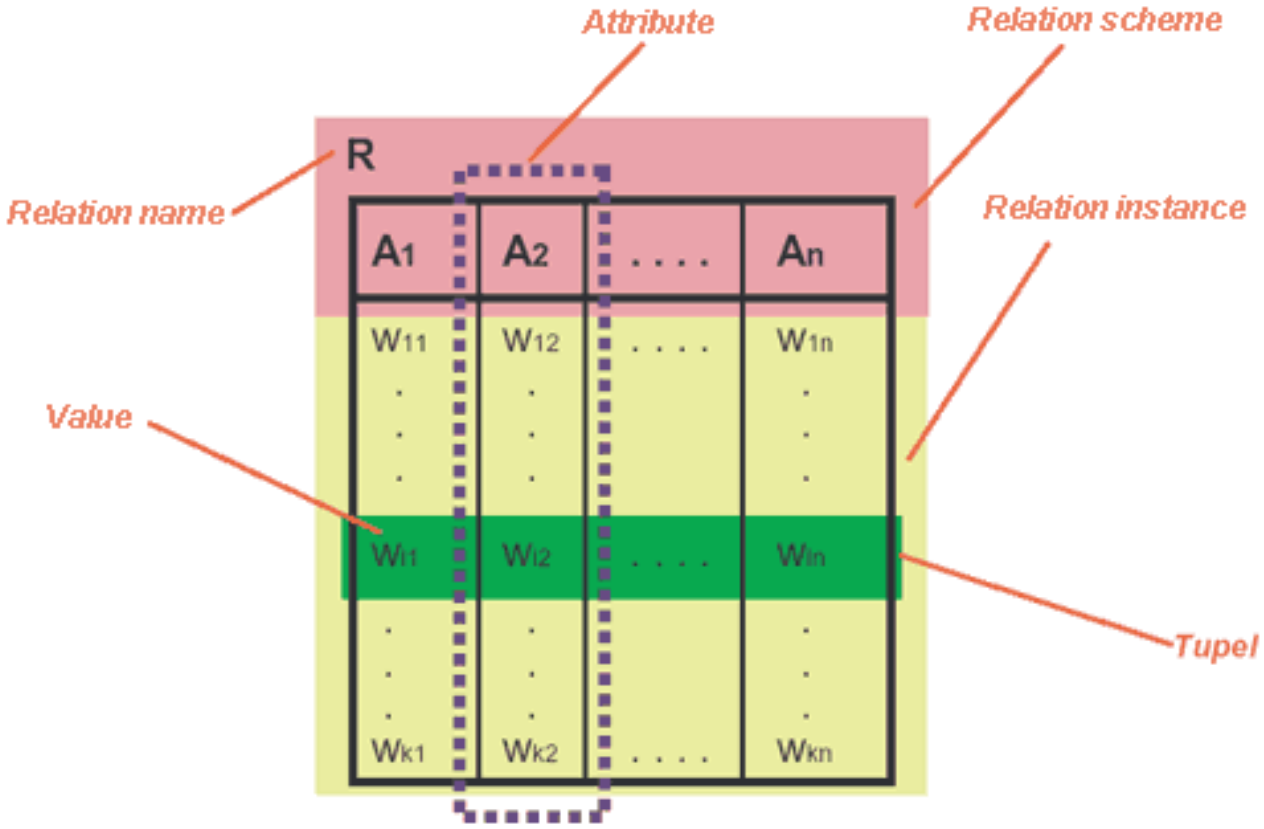
Example:

- Person = {Name, date of birth, income}

Example relation scheme

Relation:

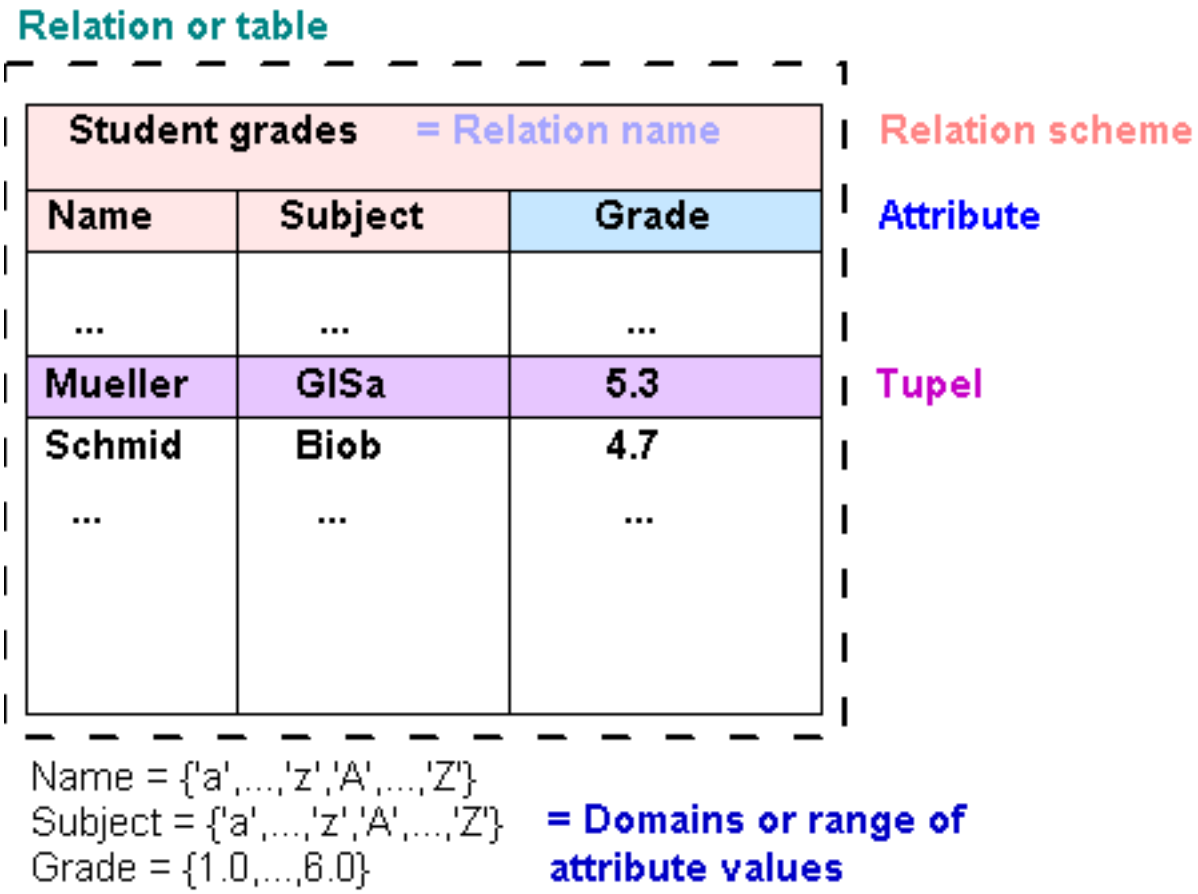
A relation **r** is one instance of the relation scheme R(A1, A2, ..., An) containing a set of n-tuples r ={t1, t2, ... , tn}.



Example relation

Relational database scheme:

A relational database scheme is a set of relation schemes S = {R1, ..., Rn} together with a set of integrity conditions. A relational database is a relational database scheme together with a database instance.



Relational scheme terms

The relational scheme of an object (entity) can be represented as a table (relation). In this example the entity are grades. This entity is described with the attributes name, subject and the grade. The domain (or value range) for the attributes name and subjects are all lower- and upper-case characters of the alphabet, the domain for the grades are real numbers from 1 to 6. The structure of this entity without any content is called a relational scheme. Any value that is entered has to be within the defined value range or domain. A row in the table is also called a tuple.
There is a small mistake in the above table. Do you find it?

1.2. Transforming an ERM to a relational database scheme

In this unit we will learn the rules and methods to represent entity relationship models in relational database schemes.
Each of the following mapping rules describes one of the components of the entity relationship model. To reduce an ERM into a relational scheme all 8 rules have to be worked out. Each rule has to be applied on every entity set (rule 1, 2, 7 and 8) or relationship set (rule 3, 4, 5 and 6). Usually the correct processing sequence is: 1, 7, 8, 2, 3, 4, 5, and then 6.
For every rule a definition and an example is given.

1.2.1. ERM concepts

Entity relationship diagram
To understand the following mapping rules, you must be familiar with the concepts of the entity relationship model (ERM). Using this flash example, you can refresh your memory about ERM.
hould any of the concepts be new, please consult the following book:

- Fundamentals of Database Systems, Elmasri, Ramez; Navathe, Shamakant B. (1994)

You should be familiar with the following terms: strong entity, weak entity, relationship, identifying relationship, attribute, derived attribute, multivalued attribute, composite attribute, identifier or primary key and discriminator.

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

1.2.2. Rule 1

In this first step, all strong entity sets are transformed into the relational database schema. For subclasses use rule 8.

Definition rule 1
For each strong entity set G define a relational scheme R with the entity properties as attributes A. For multivalued attributes use rule 7. Define the primary key (identifier).
In this example you can see the application of rule 1::

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A strong entity set is taken for the creation of the relation:
Customer
All entity properties are used as attributes for the relation:
Customer (CustomerNo, FirstName, LastName, Street, StreetNo)

The primary key (or identifier) is defined. Here it is the attribute CustomerNo:
Customer (CustomerNo, FirstName, LastName, Street, StreetNo)

1.2.3. Rule 2

In this step the weak entity sets are transformed into the relational database scheme.

Definition rule 2
For each weak entity set S with owner G create a relational scheme R with the entity properties as attributes A. For multivalued attributes use rule 7. Use the primary key of G as foreign key in R. Choose a discriminator (combination of attributes) that, together with the foreign key, will act as primary key for R. Note that only the combination of the foreign key together with the discriminator can act as primary key for R.
IIn this example you can see the application of rule 2:

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A weak entity set is taken for the creation of the relation:
Part
All entity properties are used as attributes for the relation:
Part(Name, Editor)
The primary key of the owner's relational scheme is taken and added as a foreign key to the relation. In this example the owner is Newspaper(Name, Circulation, Price) and the primary key is Name which we use in the relation as NewspaperName (since Part already has a Name):
Part(NewspaperName, Name, Editor)
The discriminator attribute Name (of Part) together with the foreign key NewspaperName form the primary key of the relation Part:
Part(NewspaperName, Name, Editor)

1.2.4. Rule 3

In this step the binary relationships of the following type (1,1)(1,1), (0,1)(1,1) or (0,1)(0,1) are transformed into the relational database scheme.

Definition rule 3
Search for all binary relationships B (1,1)(1,1), (0,1)(1,1) or (0,1)(0,1). Find the relational scheme S and T that are connected through relationship B. Choose one of them (eg. S) and insert the primary of the other relational scheme (eg. T) as a foreign key. Also add the properties of B as attributes into this relational scheme.
In this example you can see the application of rule 3:

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A binary relationship (type (1,1)(1,1)-, (0,1)(1,1)- and (0,1)(0,1)) is chosen: leads.
Then we choose an entity set that is related to this relationship:
Newspaper(Name, Circulation, Price)
The primary key of the second entity set (eg. ChiefEditor) is inserted as foreign key in the first relational scheme:
Newspaper(Name, Circulation, Price, *ChiefEditor_PersNo.)*
Now the properties of the relationship are used as attributes in the relational scheme:
Newspaper(Name, Circulation, Price, *ChiefEditor_PersNo.*, SinceDate)

1.2.5. Rule 4

In this step all binary relationships one to many/many to one (1,n)(1,1), (0,n) (1,1), (1,n)(0,1) or (0,n)(0,1) are transformed into the relational database scheme.

Definition rule 4
Search for all regular binary relationships B of type (1,n)(1,1), (0,n)(1,1), (1,n)(0,1) and (0,n)(0,1) and for their relational schemes S and T of the corresponding entity sets. Choose the relational scheme on the "(1,1)"/"(0,1)"-side (here S) and insert there the primary key of T as a foreign key. Also add the properties (if there are any) of B as attributes into this relational scheme.
In this example you can see the application of rule 4:

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A binary relationship of type one to many/many to one ((1,n)(1,1)-, (0,n)(1,1)-, (1,n)(0,1)- und (0,n)(0,1)) is chosen: "commissions"
Then we choose the entity set that is on the "(1,1) or (0,1)"-side:
Advertisment(AdNo, Size, Price, PublDate)
The primary key of the second entity set (eg. Customer) is inserted as foreign key in the first relational scheme:
Advertisment(AdNo, Size, Price, PublDate, *ClientCustomerNo.)*

1.2.6. Rule 5

In this step all binary relationships many to many (0,n)(0,n), (1,n) (0,n) or (1,n)(1,n) are transformed into the relational database scheme.

Definition rule 5
Search for all regular binary relationships B of type many to many and the according relational schemes S and T. For each B create a new relational scheme R. The primary keys of S and T are used as foreign keys in R. Together they form the primary key of this new relational scheme R. Also add the properties (if there are any) of B as attributes into this relational scheme R.
In this example you can see the application

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A binary relationship of type many to many ((0,n)(0,n)-, (0,n)(1,n)- and (1,n)(1,n)) is chosen: "published in"
For this relationship we define a new relational scheme:
AdpublishedInNewspaper
The primary keys of the connected relational schemes S and T are used as foreign key in the new scheme R. Together they form the primary key for R:
AdpublishedInNewspaper (Ad_OrderNo, Newspaper_Name)
Now the properties of the relationship are used as attributes in the relational scheme:
AdpublishedInNewspaper (NameAd_OrderNo, Newspaper_Name, Position)

1.2.7. Rule 6

In this step all relationships of order n are transformed into the relational database scheme. This happens according to rule 5.

Definition rule 6
For all n-ary relationship types (n>2) use rule 5: Create a new relational scheme R and use the primary keys of all connected relational schemes (S, T, etc.) as foreign keys. Together they form the primary key for this new relational scheme R.
See the example of rule 5.

1.2.8. Rule 7

If in rule 1 multivalued attributes are encountered, they are transformed into separate relational schemes according to this rule.

Definition rule 7
Define for each multivalued attribute A a new relational scheme R' which also contains this attribute A and the primary key of the according relational scheme R. Together they form the primary key for R'.
In this example you can see the application of rule 7:

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

A new relational scheme is created for the multivalued attribute TelNo containing TelNo as attribute:
ChiefTelNo(TelNo)
The primary key of this multivalued attribute connected to the relational scheme is used as foreign key in the new scheme:
ChiefTelNo(*ChiefEditor_PersNo*, TelNo)
The foreign key and the attribute together form the primary key for this new relational scheme:

ChiefTelNo(ChiefEditor_PersNo, TelNo)

1.2.9. Rule 8

If in rule 1 you find subclasses then transform them according to this rule.

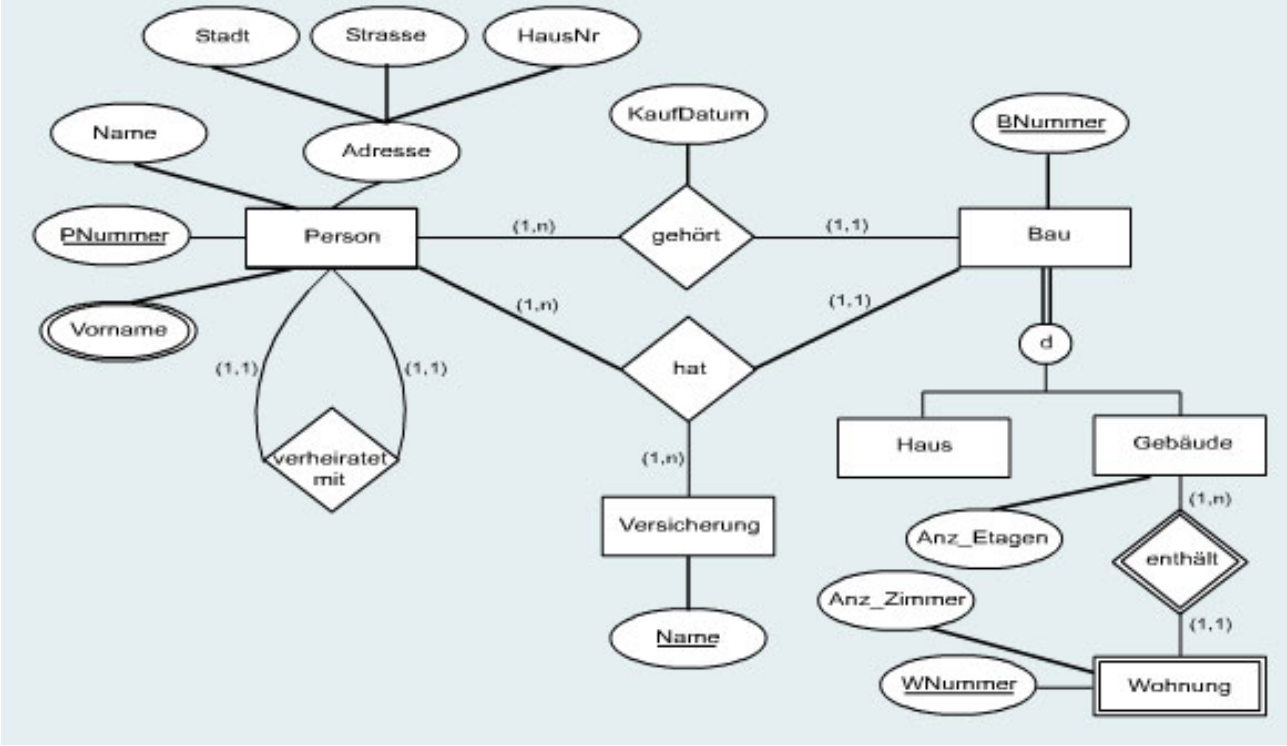
Definition rule 8
Define a relational scheme R for the superclass C with the attributes A(R)=(K, A1, A2, .., An), where K is the primary key. For each subclass create a new relational scheme Ri with their attributes and the primary key K of superclass C as an additional attribute. The primary key of Si is also K.
In this example you can see the application of rule 8:

Only pictures can be viewed in this version! For Flash, animations, movies etc. see online version. Only screenshots of animations will be displayed. [link]

Using the superclass define a relational scheme R with a primary key K:
Employee(ENumber)
For each subclass create a relational scheme:
Technician
Engineer(Training)
Add the primary key K of the superclass as attribute of the subclasses and use it also as primary key:
Employee(ENumber),
Technician(ENumber),
Engineer(ENumber, Training)

1.2.10. Using the 8 rules

The following flash animation allows you to practice the 8 rules that you have now learned. Please click on the linked flash-graphic below and follow the instructions.

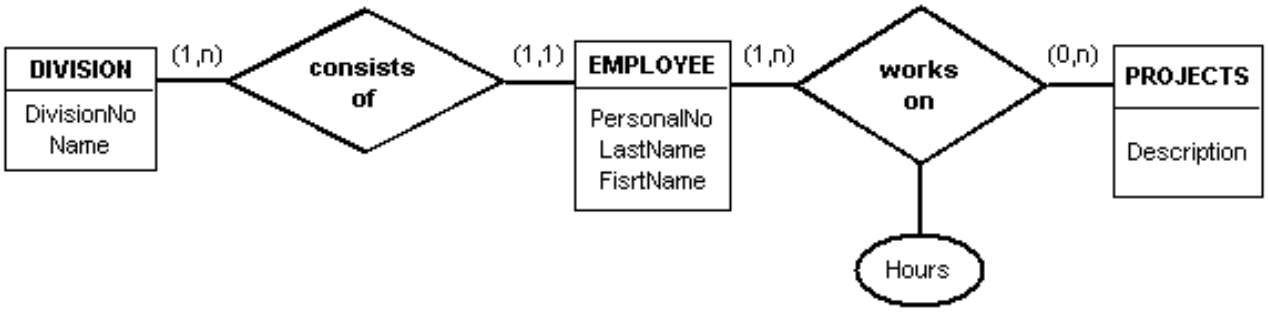


Flash exercise

1.2.11. Reducing an ERM to a relational scheme

Create a relational database scheme using the following conceptional data scheme (ERM - Entity Relationship Model) using the rules you here learned in this unit. Primary keys should be marked as underlined and foreign keys italic.

Publish your solution as a Word- or PDF-document on the discussion board. Check out the solutions of the other students and comment them. Any problems with this exercise can also be published on the discussion board. The solutions and questions will be checked by a tutor and general feedback published.



A conceptional model (ERM)

1.3. Data integrity

Integrity or consistency stands for the quality and reliability of data of a database system. A database is consistent if the data reflects the referenced objects correctly. It is inconsistent if there exist ambiguous or contradictory tuples, relations or tables in the database.

1.3.1. Key integrity

Relations are defined as a unique set of tuples where no two tuples have the same combination of values for all their attributes. To be distinct, they must have some primary key that allows to reference each tuple. Key constraints deal with this issue.

Candidate key:

Each attribute or minimal combination of attributes that uniquely identifies any tuple in a relation is called a candidate key. Minimal means that removing an attribute leaves the key without the ability to uniquely identify any tuple and therefore not being a candidate key anymore.

Primary key:

The primary key is one chosen key candidate that acts as the identification key for a relation. Usually this is a short attribute like a ID-number (identification key) or username. Attributes of the primary are commonly underlined.

1.3.2. Entity integrity

The entity integrity is a result of the key integrity: No primary key is allowed to be the NULL(=no value). If NULL-values would be allowed for primary keys, than two tuples could have NULL as key value and therefore could not be distinguishably anymore. The key integrity would not be respected. In SQL special routines (unique, not null) are used for key and entity integrity.

Example:

ID	Name	Surname	Year
NULL	Miller	John	1955
NULL	Miller	John	1985

Since the primary key (here ID) of both tuples is NULL, we are not able to distinct these tuples.

1.3.3. Referntial integrity

In a relationship model, in contradiction to an entity relationship model ERM, there is no way to model relationships between tuples explicitly. Therefore relationships are modeled implicitly using a primary- and foreign-key concept.

Foreign key:

An attribute in a relational scheme R1 is a foreign key if it is in relationship with a primary key from R2 and if:

- The domain (value range) of the foreign key in R1 is the same as the domain of the primary key in R2.
- The set of values of the foreign key in R1 is a subset of all primary key values in R2.

Foreign keys usually are marked dotted underlined.

A relationship between two schemes is established by using the domain of the primary key in one scheme as the domain of a foreign key in a second scheme (with the according attributes).
Referential integrity constraints ensure that any foreign key value is always pointing on an primary key of an existing tuple. References to non-existing primary keys are not allowed. In such a case the foreign key value must be set to NULL.
The relational scheme containing the foreign key is called the referencing scheme, the scheme with the primary key is called the referenced scheme.A foreign key can also point to its own scheme.
In the early days of computer databases, only workstations were able to check for referential integrity. Today most PC-based database systems are able to check for referential integrity.

Employee

ID	LastName	Division
1	Mueller	A1
2	Meier	A3
3	Tobler	A2

Division

Div_No	Professor
A1	Informatics
A2	Marketing
A3	Finance

Example Referntial integrity

In the above table Division we have Div_No as a primary key. In the second table Employee this key is used as a foreign key to associate each member to one department. For each department number in the table Employee there exists a department in the table Division. These tables are integer. If we would insert a tuple "4, Weber, A5" into table Employee then a database system should refuse this operation because in table Division there exists no value A5 and therefore there is no integrity.

1.3.4. Integrity endangering operations

There are three types of operation that could potentially endanger referential integrity:

- Inserting of a new tuple
- Deleting an existing tuple
- Updating attribute values of existing tuples

Of course with all these operations there has to be a check on all integrity rules. Selecting values (browsing in the database) does not change any values and is therefore no integrity endangering operation.

Inserting of a new tuple
With this operation all three integrity rules are concerned. The following problems can occur:

- Key integrity: The for this insertion used primary key value is already used in another tuple.
- Entity integrity: The primary key of the new tuple is NULL.
- Referential integrity: A foreign key without an associated primary key is used.

These operations must either be rejected by the database system or transformed into consistent operations according to defined rules.

Employee

ID	LastName	Division
1	Mueller	A1
2	Meier	A3
3	Tobler	A2

Division

Div_No	Professor
A1	Informatics
A2	Marketing
A3	Finance

When inserting following tupels in the relation Emplyoee integrity rules would be broken

3	Weber	A2	There is already a tuple witth identification key "3"
	Weber	A2	no identification key selected
4	Weber	A4	Division "A4" doesn't exist

Examlpe Inserting of tuples

Deleting an existing tuple

With this operation only referential integrity is concerned. If a foreign key points on a primary key that has been deleted, then the foreign key becomes invalid. The database system should react on a delete operation with one of the following solutions: Abort the operation, cascading delete (also the referenced tuple is deleted!) or set the value of the foreign key to NULL.

Employee

ID	LastName	Division
1	Mueller	A1
2	Meier	A3
3	Tobler	A2

Division

Div_No	Professor
A1	Informatics
A2	Marketing
A3	Finance

Example Deleting existing tuples

By deleting the tupel Div_No "A1" in the relation Division, the division number of the tupel ID "1" in the relation Employee would become invalid.

Updating attribute values of existing tuples

With this operation all three integrity rules are concerned. An update operation can be seen as first a delete of an existing tuple followed by a insert of a new tuple with updated values. Therefore all the above rules must be applied. Problems arise only with the updating of primary or foreign keys. All other attributes can be updated without any restriction.

Employee

ID	LastName	Division
1	Mueller	A1
2	Meier	A3
3	Tobler	A2

Division

Div_No	Professor
A1	Informatics
A2	Marketing
A3	Finance

Example Updating attribute values of existing tuples

By modifying the value Div_No in the relation Division from "A1" to "A4", the division number of ID "1" in the relation Employee would become invalid.

1.4. Normalisation

Normalisationis a process which we analyze and alter a database relation in order to get more concise and organized data structures. Normalised data is stable and has a natural structure. We call a relation normalized if:

- it does not contain any redundancy
- it does not cause maintenance problems
- it is an accurate representation of the data

Relations that aren't normalised contain non-atomic attributes and therefore can contain redundant information. Detailed planning of the ERM can help creating normalised relations. The following steps will explain how existing relations can be normalised step by step.

1.4.1. Dependencies

In order to be able to normalise a relation according to the three normal forms, we must first understand the concept of dependency between attributes within a relation.

Functional dependency:

If A and B are attributes of relation R, B is functionally dependenton A (denoted A --> B), if each value of A in R is associated with exactly one value of B in R.

Example:

ID	Name
S1	Meier
S2	Weber

The attribute Name is functionally dependent of attribute ID (ID --> Name).

Identification key:

If every attribute B of R is functionally dependent of A, than attribute A is a primary key.

Beispiel:

ID	Name	Surname
S1	Meier	Hans
S2	Weber	Ueli

Attribute ID is the identification key

Full functional dependency:

We talk about full functional dependency if attribute B is functional dependent on A, if A is a composite primary key and B is not already functional dependent on parts of A.

Beispiel:

IDStudent	Name	IDProfessor	Grade
S1	Meier	P2	5
S2	Weber	P1	6

The attribute Grade is fully functional dependent on the attributes IDStudent and IDProfessor.

Transitive dependency:

If A determines B and B determines C then C is determined by (dependent on) A. We write A --> B and B --> C but not B --> A.

Example:

ID	Name	Konto_Nr	Bank_Code_No	Bank
L1	Meier	1234-5	836	UBS
L2	Weber	5432-1	835	CS

There is a transitive dependency between Bank_Code_No and Bank because Bank_Code_No is not the primary key of the relation.

1.4.2. First normal form (1NF)

First normal form:

A relation is in first normal form if every attribute in every row can contain only one single (atomic) value.

A university uses the following relation:

Student(Surname, Name, Skills)

The attribute Skills can contain multiple values and therefore the relation is not in the first normal form.

But the attributes Name and Surname are atomic attributes that can contain only one value.

Students

FirstName	LastName	Knowledge
Thomas	Mueller	Java, C++, PHP
Ursula	Meier	PHP, Java
Igor	Mueller	C++, Java

Startsituation

Result after Normalisation

Students

FirstName	LastName	Knowledge
Thomas	Mueller	C++
Thomas	Mueller	PHP
Thomas	Mueller	Java
Ursula	Meier	Java
Ursula	Meier	PHP
Igor	Mueller	Java
Igor	Mueller	C++

Example First normal form

To get to the first normal form (1NF) we must create a separate tuple for each value of the multivalued attribute

1.4.3. Second normal form (2NF)

Second normal form:

A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.

A university uses the following relation:
Student(IDSt, StudentName, IDProf, ProfessorName, Grade)
The attributes IDSt and IDProf are the identification keys.
All attributes a single valued (1NF).
The following functional dependencies exist:
1. The attribute ProfessorName is functionally dependent on attribute IDProf (IDProf --> ProfessorName)
2. The attribute StudentName is functionally dependent on IDSt (IDSt --> StudentName)
3. The attribute Grade is fully functional dependent on IDSt and IDProf (IDSt, IDProf --> Grade)

Students

IDSt	LastName	IDProf	Prof	Grade
1	Mueller	3	Schmid	5
2	Meier	2	Borner	4
3	Tobler	1	Bernasconi	6

Startsituation

Result after normalisation

Students

ID	LastName
1	Mueller
2	Meier
3	Tobler

Professors

IDProf	Professor
1	Bernasconi
2	Borner
3	Schmid

Grades

IDStIDProf	Grade	
1	3	5
2	2	4
3	1	6

Example Second normal form

The table in this example is in first normal form (1NF) since all attributes are single valued. But it is not yet in 2NF. If student 1 leaves university and the tuple is deleted, then we loose all information about professor Schmid, since this attribute is fully functional dependent on the primary key IDSt. To solve this problem, we must create a new table Professor with the attribute Professor (the name) and the key IDProf. The third table Grade is necessary for combining the two relations Student and Professor and to manage the grades. Besides the grade it contains only the two IDs of the student and the professor. If now a student is deleted, we do not loose the information about the professor.

1.4.4. Third normal form (3NF)

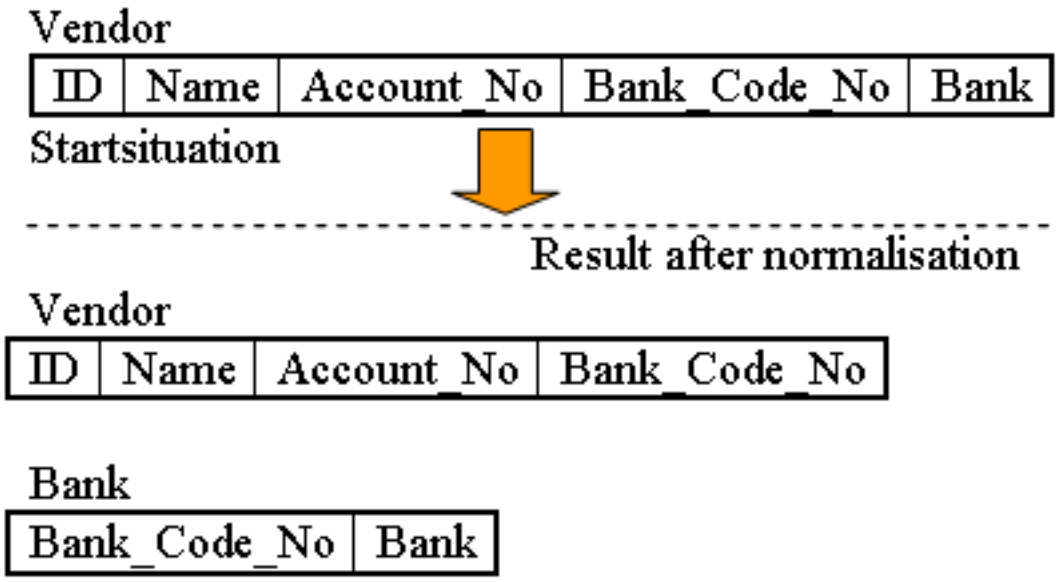
Third normal form:

A relation is in third normal form if it is in 2NF and no non key attribute is transitively dependent on the primary key.

A bank uses the following relation:

Vendor(ID, Name, Account_No, Bank_Code_No, Bank)

The attribute ID is the identification key. All attributes are single valued (1NF). The table is also in 2NF.
The following dependencies exist:
1. Name, Account_No, Bank_Code_No are functionally dependent on ID (ID --> Name, Account_No, Bank_Code_No)
2. Bank is functionally dependent on Bank_Code_No (Bank_Code_No --> Bank)



Example Third normal form

The table in this example is in 1NF and in 2NF. But there is a transitive dependency between Bank_Code_No and Bank, because Bank_Code_No is not the primary key of this relation. To get to the third normal form (3NF), we have to put the bank name in a separate table together with the clearing number to identify it.

1.4.5. Exercise Normalisation

The following table is already in first normal form (1NF). There is only one entry per field. Please convert this table to the third normal form (3NF) using the techniques you learned in this Unit. Write a short report about your solution and post it in the discussion board. Check the other solutions and comment them, if necessary. If you have questions, you can also post them in the discussion board. A tutor will look at the questions regularly and give feedback and answers.
A table with the students and their grades in different topics.

UnitID	StudentID	Date	TutorID	Topic	Room	Grade	Book	TutEmail
U1	St1	23.02.03	Tut1	GMT	629	4.7	Deumlich	tut1@fhbb.ch
U2	St1	18.11.02	Tut3	GIn	631	5.1	Zehnder	tut3@fhbb.ch
U1	St4	23.02.03	Tut1	GMT	629	4.3	Deumlich	tut1@fhbb.ch
U5	St2	05.05.03	Tut3	PhF	632	4.9	Dümmlers	tut3@fhbb.ch
U4	St2	04.07.03	Tut5	AVQ	621	5.0	SwissTopo	tut5@fhbb.ch

1.4.6. Unit-Zusammenfassung

This Unit showed how and why relations should be normalised. It prevents problems and saves money. A careful planning in the conceptual phase helps implementing and normalising a table properly. Although there is a fourth and a fifth normal form (that were not discussed here), usually it is enough to normalise up to the third normal form.

1.5. Summary

In a relational model real world objects are represented in tables. Each table is made out of rows and columns. Each row, also known as tuple or record, is made out of fields, also known as attributes. Each Attribute stands for a certain feature of the real world object. An attribute is defined by a name and its value.
Relations between tuples represent existing relationships between objects (tables). Furthermore key attributes have to be defined (usually displayed underlined in a relation). They are necessary for the allocation (relation) of objects (tables) and allow unique accesses to tables.
Integrity or consistency stands for the quality and reliability of data of a database system. A database is consistent if the data reflects the referenced objects correctly. It is inconsistent if there exist ambiguous or contradictory tuples, relations or tables in the database.
A relation model (scheme, entity) should reflect relationships that also logically (in the real world) belong together. To avoid anomalies different types of normalisations help keeping the database consistent.

1.6. Recommended Reading

- **ELMASRI, R., NAVATHE, S.B.**, 1994. *Fundamentals of Database Systems*. 2nd. Redwood City, California: Addison-Wesley.
  Introduction in databases and SQL (in English).

1.7. Glossary

Attribute:

A column of a table represents an attribute. It can also be described as a that a domain D has in arelation scheme R.

Candidate key:

Each attribute or minimal combination of attributes that uniquely identifies any tuple in a relation is called a candidate key. Minimal means that removing an attribute leaves the key without the ability to uniquely identify any tuple and therefore not being a candidate key anymore.

domain:

A domain **D** is a set of atomic values that defines the value range of attributes.

First normal form:

A relation is in first normal form if every attribute in every row can contain only one single (atomic) value.

Foreign key:

An attribute in a relational scheme R1 is a foreign key if it is in relationship with a primary key from R2 and if:

- The domain (value range) of the foreign key in R1 is the same as the domain of the primary key in R2.
- The set of values of the foreign key in R1 is a subset of all primary key values in R2.

Foreign keys usually are marked dotted underlined.

Full functional dependency:

We talk about full functional dependency if attribute B is functional dependent on A, if A is a composite primary key and B is not already functional dependent on parts of A.

Functional dependency:

If A and B are attributes of relation R, B is functionally dependenton A (denoted A --> B), if each value of A in R is associated with exactly one value of B in R.

Identification key:

If every attribute B of R is functionally dependent of A, than attribute A is a primary key.

Primary key:

The primary key is one chosen key candidate that acts as the identification key for a relation. Usually this is a short attribute like a ID-number (identification key) or username. Attributes of the primary are commonly underlined.

Relation:

A relation **r** is one instance of the relation scheme R(A1, A2, ..., An) containing a set of n-tuples r ={t1, t2, ... , tn}.

Relational database scheme:

A relational database scheme is a set of relation schemes S = {R1, ..., Rn} together with a set of integrity conditions. A relational database is a relational database scheme together with a database instance.

Relation scheme:

A relation scheme **R** (A1, A2, ..., An) is made up of a relation name R and a list of attributes {A1, A2, ..., An}.

Second normal form:

A relation is in second normal form if it is in 1NF and every non key attribute is fully functionally dependent on the primary key.

Third normal form:

A relation is in third normal form if it is in 2NF and no non key attribute is transitively dependent on the primary key.

Transitive dependency:

If A determines B and B determines C then C is determined by (dependent on) A. We write A --> B and B --> C but not B --> A.

Tupel:

A tuple **t** is a list with n values **t = <d1, d2, ..., dn>** where each value di is either an element of the domain **Di** or NULL. A tuple is a record in a relation (row in a table).

1.8. Bibliography

- **ELMASRI, R., NAVATHE, S.B.**, 1994. *Fundamentals of Database Systems*. 2nd. Redwood City, California: Addison-Wesley.
- **ZEHNDER, C.A.**, 1998. *Informationssysteme und Datenbanken*. Zürich: vdf Hochschulverlag AG.