Project Report

Data Storage Paradigms, IV1351

Lana Ryzhova

lana_lana67@ukr.net

Ångermannagatan 1B lgh 1003

162 64 Vällingby

2004 11 05

Data Storage Paradigms, IV1351 Project Report

Task 1, Conceptual Model

1 Introduction

1. **The purpose of this report is:**

   1.1 Creating a conceptual model of the Soundgood music school database.

   1.2 Construction of a diagram that is created based on the description of the requirements for solving the problem of creating a conceptual model.

   1.3 Transforming a conceptual model into a functioning database.

2. **The task was solved independently.**

2 Literature Study

1. A lecture on the conceptual model from the book "Fundamentals of Database Systems (7th Edition) by Ramez Elmasri and Shamkant B. Navathe" was read and understood.

2. Videos on conceptual modeling were viewed.

3. Was read in articles from the Internet about inheritance.

4. The PostgreSQL program has been installed and its basic functionality has been analyzed.

5. The task of creating a conceptual model of the Soundgood music school was solved using Data Modeling using the Entity–Relationship (ER) Model, using crow's Foot Notation.

6. The report and solution to this problem have been posted in my Git repository.

   https://github.com/Lana-1167/KTX-1167.git/

   The data for the first seminar is in the folder

   https://github.com/Lana-1167/KTX-1167.git /seminar1/

7. The document tips-and-tricks-task1.pdf was read and understood.

3 Method

Basic Components of Conceptual Modeling:

Entities: represent key objects or concepts in the domain, such as Student, Instructor, Lesson, or Musical Instrument, etc**.**

Attributes: define properties of objects such as Student Name, Person number, Data of rent instruments, Address, etc.

Relationships: represent connections between entities, such as a student placing multiple orders for different lessons on different instruments, musical instruments belonging to different brands and having different types, or an instructor who can be assigned to group, individual and ensemble lessons, etc.

Procedure for creating a conceptual model.

Description of the procedure to follow to create a conceptual model, following the steps covered in the conceptual modeling videos.

The steps to create a conceptual data model include several stages:

1. It is necessary to define key database objects:

    1.1. List the key database objects that will be included in this database.

    1.2. Determine which database objects are of primary importance and require storage and retrieval.

2. It is necessary to define the attributes of each entity that are relevant to the scope of the conceptual data model.:

    2.1. Identify the attributes of each entity that fall within the scope of the conceptual data model.

    2.2. Define the basic properties of each entity without going into details such as data types or constraints.

3. Establish relationships that make sense from a business perspective:

    3.1. Make an analysis of the relationships between objects.

    3.2. Based on the analysis, determine the existing relationships.

4. Review and refine the conceptual model for inconsistencies, redundancies and missing information.

    4.1 Analyze the initial conceptual model for inconsistencies, redundancies, and missing information.

4.2 If necessary, update the conceptual model to improve its accuracy and completeness.

At the end of the conceptual data modeling process, a high-level representation of the conceptual data model is obtained, which will serve as the basis for the next stage of the process - logical data modeling.

This solution meets the basic requirements for solving the problem of creating a conceptual model based on information about the organizational description of the Soundgood music school and can be transformed into a functioning database.

The created conceptual model covers the entire description of the Soundgood music school company.

When creating the conceptual model, used Data Modeling using the Entity–Relationship (ER) Model, using crow's Foot Notation.

4 Result

Remark from Leif Lindbäck from 12 Nov at 18:12

You haven't specified cardinality, NOT NULL or UNIQUE for attributes.

The main goal of conceptual data modeling is to understand business requirements and enable communication between stakeholders such as business analysts, developers, and end users.

4.1 Application for attend to the Soundgood music school

One of the key objects of the database is "Application for attending the Soundgood music school", which is created in accordance with the following requirements.

«Soundgood sells music lessons to students who want to learn to play an instrument. When someone wants to attend the school, they submit contact details, which instrument they want to learn, and their present skill».

"Application for attending the Soundgood music school" will have a web interface.

4.1 Application for attending (tab. attend_student)

attend_id	name	surname	telephone	e_mail	instrument_id	level_id
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL	NULLABLE	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	Henry	Jacob	075463248	Henry. Jacob@google.com	Violin	beginner

The "Application for attending" relationship is of primary importance and requires data storage and retrieval. Data from "Application for attending" will be

transferred to the following relationships using procedures: "Student", "Address", "Contact_details".

When data from the “Application for attending” is inserted into the Student table, the student is assigned a Person number. There are 3 types of Person numbers. For students, for instructors, and for other persons (e.g. student's parents). Different sequences of personal numbers will be used to separate these entities. For example, 100 to 999 are students, 1000 to 4999 are instructors, 5000 to 9000 are other people.

4.2 Invitation

According to the task condition "If there is a room, the student is offered a place". "Invitation" will have a web-interface.

4.2 Invitation (tab. invitations)

invitation_id	**name**	**surname**	**instrument_id**	**level_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	Henry	Jacob	Violin	beginner

4.3 Lesson.

According to the conditions of the task “There is no concept like 'course' or sets of lessons. Instead, students continue to take lessons as long as they wish. There are individual lessons and group lessons.

4.3 lesson (tab. lessons)

lessons_id	**type_ lessons**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	individual
2	group
3	ensemble

4.4 Level of lesson.

According to the conditions of the task “There are three levels: beginner, intermediate and advanced”.

4.4 levels (tab. levels)

level_id	**level_stud**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	beginner
2	intermediate
3	advanced

4.5 Ensemble genre

According to the conditions of the task «Ensembles have a specific target genre (e.g., punk rock, gospel band)».

4.5 Ensemble genre (tab. ensemble_genre)

ensembles_id	**genre_ensembles**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	punk rock
2	gospel band

Remark from Leif Lindbäck from 12 Nov at 18:12

There should be some record of individual instruments, the company will probably want to know where each specific instrument is, not just how many are left in stock.

Solution:

Tracking specific tools:

In the Tool entity, the system can track a specific rental tool, including its serial number, condition, and availability.

An added benefit: this approach also supports better inventory management and allows tools to be tracked in the event of loss or damage.

Updated Design:

The table "**types_instruments**" (type_id as PK, types_instrum).

The table "**brands**" (brand_id as PK, brands).

Let's add a table "**storage_places**" contains storage space for musical instruments (place_id as PK, storage_places).

Let's add a table "**conditions**"contains condition of musical instruments (condition_id as PK, instrument_id as FK, data_start, data_end, storage_places_id as FK, unique serial number, cost, balance in stock (computed attribute))

The "**mus_instruments**" table contains (instrument_id as PK, instruments, brand_id and type_id as FK, quantities (total).

The table "**renting_instruments**" (renting_id as PK, student_numb as FK, instrum_id as FK, period_start, period_end, mon_numb_renting, quantity_instruments, price_id as FK). This table used as an application for renting musical instruments.

Benefits:

The table "mus_instruments" allows cost of musical instruments updates in one place and enables tracking of individual instruments.

4.6 Type of musical instruments

According to the problem statement, musical instruments have such a property as instrument type.

4.6 type of musical instrument (tab. types_instruments)

type_id	**types_instrum**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	Strings
2	Wind
3	Reed
4	Drums

4.7 Brand of musical instruments

In accordance with the condition of the problem, musical instruments have such a property as the brand of the instrument.

4.7 Brand of musical instrument (tab. brands)

brand_id	**brand**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	Gibson
2	Harman Professional
3	Shure
4	Yamaha

4.8 Storage places for musical instruments

4.8 Storage places for musical instruments (tab. storage_places)

place_id	**storage_places**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	lease agreement
2	balance in stock
3	under repair
4	written off the balance sheet

4.9 Musical instruments

There is a wide selection of instruments, wind, string etc., supporting different brands and in different quantities in stock at the Soundgood music school.

4.9 Musical instruments (tab. mus_instruments)

instrument_id	**instruments**	**brand_id**	**type_id**	**quantities (total)**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL
1	Violin	1	2	5
2	Piano	6	1	2
3	Guitar	8	4	7
4	Drums	4	5	10

4.10 Condition of musical instruments

4.10 Condition of musical instruments (tab. conditions)

condition_id	**instrument_id**	**data_start**	**data_end**	**storage_places_id**	**unique serial number**	**cost**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, UNIQUE	NOT NULL
1	1	2024-01-09	2025-01-09	1	A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A11	10000
2	2	2024-10-05	2024-12-10	1	A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A12	50000
3	3	2024-01-09	2024-05-09	2	A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A13	5000
4	4	2024-03-20	2024-10-09	4	A0EEBC99-9C0B-4EF8-BB6D-6BB9BD380A14	10000

Remark from Leif Lindbäck from 12 Nov at 18:12

The attribute numb_sibling doesn't tell who's a sibling of who. This means that when a student quits, it won't be possible to know which student's sibling count shall be decremented.

Issue with the numb_sibling attribute:

- Instead of storing a count of siblings in the Student entity, we should explicitly model sibling relationships.
- **Solution:** Introduce a Sibling relationship as a self-referencing relationship on the Student entity. This way:
    - Each student can be linked to one or more siblings.
    - If a student leaves the school, we can still maintain the sibling relationships and decrement counts accordingly.

Updated Design:

- Student (student_numb as PK, name, contact details, etc.)
- Sibling (student_num, person_sibling)
    - Both student_numb and person_sibling refer to Student.student_numb.

Use a composite primary key (student_numb, person_sibling) to ensure no duplicate sibling relationships.

4.11 Student

According to the conditions of the task «Person number, name, address and contact details must be stored for each student».

4.11 student (tab. student)

student_numb	**firstname**	**surname**	**address_id**	**details_id**	**person_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
117	Henry	Jacob	1	1	1

4.12 Sibling

The "Sibling" relation stores information about siblings. According to the assignment - "Furthermore, it must be possible to see which students are siblings, since there is a discount for siblings. It's not sufficient to show just whether a student has siblings, it must be possible to see who's a sibling of who."

4.12 Sibling (tab. sibling)

student_numb	**person_sibling**
PRIMARY KEY, UNIQUE, NOT NULL	PRIMARY KEY, UNIQUE, NOT NULL
117	118
119	120
120	123

4.13 Address

The "Address" relation stores information about addresses.

4.13 Address (tab. address)

address_id	postal_code	city	street	home	room
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL
117	123 55	Stockholm	Nygatan	11	20

4.14 Contact details

The "Contact details" relation stores information about contact details.

4.14 Contact details (contact_detail)

contact_id	telephone	e_mail
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NULLABLE
117	741132337	Henry.Jacob@google.com

4.15 Persons

According to the task “It must also be possible to store contact details for a contact person (e.g., parent) for each student”.

For a relationship «persons», personal numbers will start with the 5000 number.

4.15 Person contact (tab. persons)

person_id	name_per	surname_per	student_numb	contact_id	related_id
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
5000	Kris	Jacob	117	1	1
5001	Kris	Jacob	118	2	1
5002	Kris	Jacob	119	3	2
5003	Lucas	Levi	120	4	2

4.16 Relatives

The "Relatives" relation stores information about the relatives of student.

4.16 Relatives (tab. relative

relativ_id	relationship
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL
1	Father
2	Mother

4.17 Instructor

In accordance with the task “Person number, name, address and contact details must be stored for each instructor”.

For instructors, personal numbers will start with the 1000 number.

4.17 Instructor (tab. instructor)

instruktor_numb	name_inst	surname_inst	address_id	contact_id
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1340	Mikael	Oliver	2	2

4.18 Proficiency in musical instruments

In accordance with the task "A student can take any number of lessons, for any number of instruments". And also "An instructor can teach a specified set of instruments, and may also be able to teach ensembles". To implement these conditions, the "hold_instrument" relation was created. The "contact_id" attribute records the personal numbers of both students and instructors.

4.18 Proficiency (tab. hold_instrument)

hold_id	**inst_stud_id**	**instrument_id**	**level_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	1340	1	1
2	117	1	2

Remark from Leif Lindbäck from 12 Nov at 18:12

The fee for an instrument rental can't be specified in the renting_instruments entity. First, it means the cost is duplicated for each rental of the same instrument, made by any student. Second, if the cost changes, there's no place to write the new cost before that instrument has been rented.

Solution:

Including price data in a separate table, such as price, avoids data duplication and provides centralized management of price changes. Let's look at the benefits and aspects that influence this decision:

Advantages of the price table for storing prices for services that are available in the Soundgood music school

Centralized price management:

All prices are stored in one place. If the rental price of an instrument changes, you can simply update the corresponding record in the price table. This eliminates the need to change multiple records in other tables.

Avoid duplicate data:

The rental price of the same instrument will not be duplicated in rental records, which reduces the amount of data stored and reduces the likelihood of errors.

Support for time-based price history:

The price_start and price_finish columns allow you to store data about the period of validity of the price. This is important if prices change over time and you want to keep a history of changes.

Flexibility:

The table supports not only instrument rental, but also other services (e.g. instrument delivery). This allows you to use the same table to store all price data, which simplifies the database architecture.

How the price table works in this case:

The type_service field specifies the type of service, for example, "Rent instruments" or "Home delivery".

The price_service field specifies the cost of this service.

The price_start and price_finish fields ensure that the price only applies between the specified dates.

To relate to other entities, such as renting_instruments, you can use a foreign key referencing the price_id.

4.19 Price

In accordance with the task “Currently, there is one price for beginner and intermediate levels, and another price for the advanced level. Also, there are different prices for individual lessons, group lessons and ensembles. There is also a discount for siblings, if two or more siblings have lessons taken during the same month, they all get a certain percentage discount. Soundgood wants to have a high level of flexibility to change not just prices, but also pricing schemes. They might, for example, not always have the same price for beginner and intermediate lessons".

In addition, “Soundgood offers students the ability to rent instruments to be delivered at their home. The fee is paid the same way lessons are paid, each month students are charged for the instruments that where rented the previous month.”

4.19 Price (tab. price)

price_id	**price_start**	**price_finish**	**lessons_id**	**levels_id**	**type_service**	**price_service**	**unit_measur**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL
1	2000-01-01	2050-01-01	0	0	Rent instruments	100	SWK
2	2000-01-01	2050-01-01	0	0	Home delivery	0	SWK
3	2000-01-01	2024-09-01	1	1	Individual beginner	100	SWK
4	2000-01-01	2050-01-01	1	2	Individual intermedia	100	SWK
5	2000-01-01	2050-01-01	1	3	Individual advanced	200	SWK
6	2000-01-01	2050-01-01	2	0	Group lessons	50	SWK
7	2000-01-01	2050-01-01	3	0	Ensembles	60	SWK
8	2000-01-01	2050-01-01	0	0	Sibling discounts	5	%
9	2024-09-01	2050-01-01	1	1	Individual beginner	500	SWK

4.20 Renting Instruments

In accordance with the task “Each student can rent up to two specific instruments at any given period, the renting happens with a lease up to 12 month period. Students can list and search current instruments and rent them if they don't exceed their two-instrument quota. Instruments are rented per month. The fee is paid the same way lessons are paid, each month students are charged for the instruments that where rented the previous month".

Used as an application for renting musical instruments. Has a web-interface.

4.20 Renting Insrtruments (tab. renting_instruments)

renting_id	**student_numb**	**instrum_id**	**period_start**	**period_end**	**mon_numb_renting**	**quantity_instruments**	**price_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY
1	117	1	2024-01-09	2025-01-09	1	2	1
2	117	2	2024-10-09	2024-05-09	12	1	2

4.21 Group_lessons

In accordance with the task “A group lesson has a specified number of places (which may vary), and is not given unless a minimum number of students enroll. A lesson is given for a particular instrument and a particular level. Group lessons are given at scheduled time slots".
“A group lesson has a specified number of places (which may vary), and is not given unless a minimum number of students enroll”.
“Maxsimum number of places on group lessons (for example, from 1 to a certain number of seats, which can change)”.

Used as an application for booking of group lessons. Has a web-interface.

4.21 Group lesson (tab. group_lessons)

groupless on_id	**students_ numb**	**lesson s_id**	**data_group**	**time_slot_id**	**min_pla ces_id**	**max_places_ id**	**instrumen t_id**	**lessons_id**	**instructor_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIG N KEY	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIG N KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	117	2	09.01.2024	1	1	3	1	1	1340
2	118	2	01.01.2024	2	2	4	2	2	1340

4.22 The actual number of seats for the group lessons

The relation is used to record the actual number of students participating in group lessons.

4.22 The actual number of seats (tab. actual_seats _ensemble)

reserv_id	**group_lessons_id**	**actual _seats**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL
1	1	10
2	2	8

4.23 Ensemble

In accordance with the task “Besides lessons for a particular instrument, there are also ensembles, where students playing different instruments participate at the same lesson. Ensembles have a specific target genre (e.g., punk rock, gospel band), and there is a maximum and minimum number of students also for ensembles. Ensembles are given at scheduled time slots.

Used as an application for booking Ensemble. Has a web-interface.

4.23 Ensemble (tab. ensemble)

ensem_id	**students numb**	**ensembles_id**	**data_esem**	**time_slot_id**	**min_students_ id**	**max_students _id**	**lessons_id**	**instructor_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	117	1	09.01.2025	3	1	5	3	1340
2	118	2	01.01.2025	4	2	6	3	1340

4.24 The actual number of seats for the ensemble

The relation is used to record the actual number of students participating in ensemble lessons.

4.24 The actual number of seats (tab. actual_seats_ensemble)

reserv_id	**ensemble_id**	**actual _seats**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL
1	1	10
2	2	8

4.25 Booking individual

In accordance with the assignment “A lesson is given for a particular instrument and a particular level. Individual lessons do not have a fixed schedule, but are rather to be seen as appointments, like for example an appointment with a dentist".

Used as an application for booking individual lessons. Has a web-interface.

4.25 Booking individual (tab. booking individual)

indiv_id	**student_numb**	**period_start**	**period_end**	**numb_lessons**	**instrument_id**	**lessons_id**	**level_id**	**instructor_id**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY
1	117	09.01.2024	09.01.2025	3	1	1	1	1340
2	117	01.01.2024	31.01.2025	13	1	1	1	1340
3	120	01.01.2024	31.01.2025	30	9	1	1	1340

Remark from Leif Lindbäck from 12 Nov at 18:12

You could have omitted the admin_staff entity. There's really nothing in the project description saying that Soundgood wants to store data about admin staff.

The table name is incorrect. It would be more correct to call this relation "register of completed lessons". This entity is very important. It is a summary table that stores records of all types of lessons conducted.

4.26 Register of completed lessons

In accordance with the task "Administrative staff must be able to make bookings, it must therefore be possible to understand which instructor is available when, and for which instruments".

The table is used to register all types of lessons conducted by instructors. Application is can be used by administrative staff so that it is possible to understand which instructor is available when, and for which instruments. Has a web-interface.

4.26 Register of completed lessons (tab. complet_ lessons)

admin_id	lessons_id	levels_id	period_start	period_end	slot_id	student_id	instructor_id	instrument_id	note	price_id
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NULLABLE	NOT NULL, FOREIGN KEY
1	1	1	01.01.25	31.01.25	6	120	1340	1	instructor is available	3
1	2	0	01.01.25	31.01.25	7	8	1340	2	instructor is available	5

4.27 Time slot

In accordance with the assignment "Group lessons and ensembles are given at scheduled time slots".

4.27 Time slot(tab. start_time)

time_slot_id	**start_time**	**end_time**
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL	NOT NULL
1	8	9
2	9	10
3	10	11
4	11	12
5	12	13
6	13	14
7	14	15
8	15	16
9	16	17
10	17	18
11	18	19

4.28 Student Payment

In accordance with the task “Students are charged monthly for all lessons taken during the previous month. There is no concept like 'course' or sets of lessons, a student who has been offered a place, and accepted, continue to take lessons as long as desired, and can either book one lesson at the time or book many lessons during a longer time period”.

Application is used for payment by students. Has a web-interface.

4.28 Student Payment (tab. student_payment)

st_payment_id	**student_numb**	**period_start**	**period_end**	**lessons_id**	**price_id**	**sibling_prices_id**	**total_numb_classes**	**total_paid** (computed attribute)
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL
1	117	01.01.25	31.01.25	1	5	8	16	(16 *200*2)*5%
2	120	01.01.25	31.01.25	1	3	0	30	30 * 100

4.29 Instructor Payment

In accordance with the quest “There are no instructors with fixed monthly salaries, instead they are paid monthly for all lessons given during the previous month. Instructor payments depend on the same things as student fees (see above), namely level of lesson and whether a given lesson was a group or individual lesson. Instructor payments are not affected by sibling discounts".

Application is used to receive payment for lessons for students. Has a web-interface.

4.29 Instructor Payment (tab. instructor_payment)

ins_payment_id	**instruktor_numb**	**period_start**	**period_end**	**lessons_id**	**price_id**	**total_number_classes**	**total_received** (computed attribute)
PRIMARY KEY, UNIQUE, NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL	NOT NULL, FOREIGN KEY	NOT NULL, FOREIGN KEY	NOT NULL	NOT NULL
1	1340	01.01.25	31.01.25	1	5	16	16 *200
2	1340	01.01.25	31.01.25	1	3	30	30 * 100

Conceptual Model in Crow's Foot Notation

Textual description of the relationships to represent the Crow's Foot notation:

Student - Sibling (Self-Referential Relationship)
1..M: A student can have zero or more siblings.
1..M: A sibling relationship requires at least one other student.

Student - Lesson
1..M: A student can book one or more lessons.
1..M: A lesson must have at least one student.

Instructor - Lesson
1..M: An instructor can teach one or more lessons.
1..1: A lesson is taught by one instructor.

Student - Rental
1..M: A student can rent one or more instruments.
1..1: The rent is carried out by one student

Instrument - Rental
1..1: A rental is for one specific instrument.
1.. M: An instrument can be rented multiple times over time.

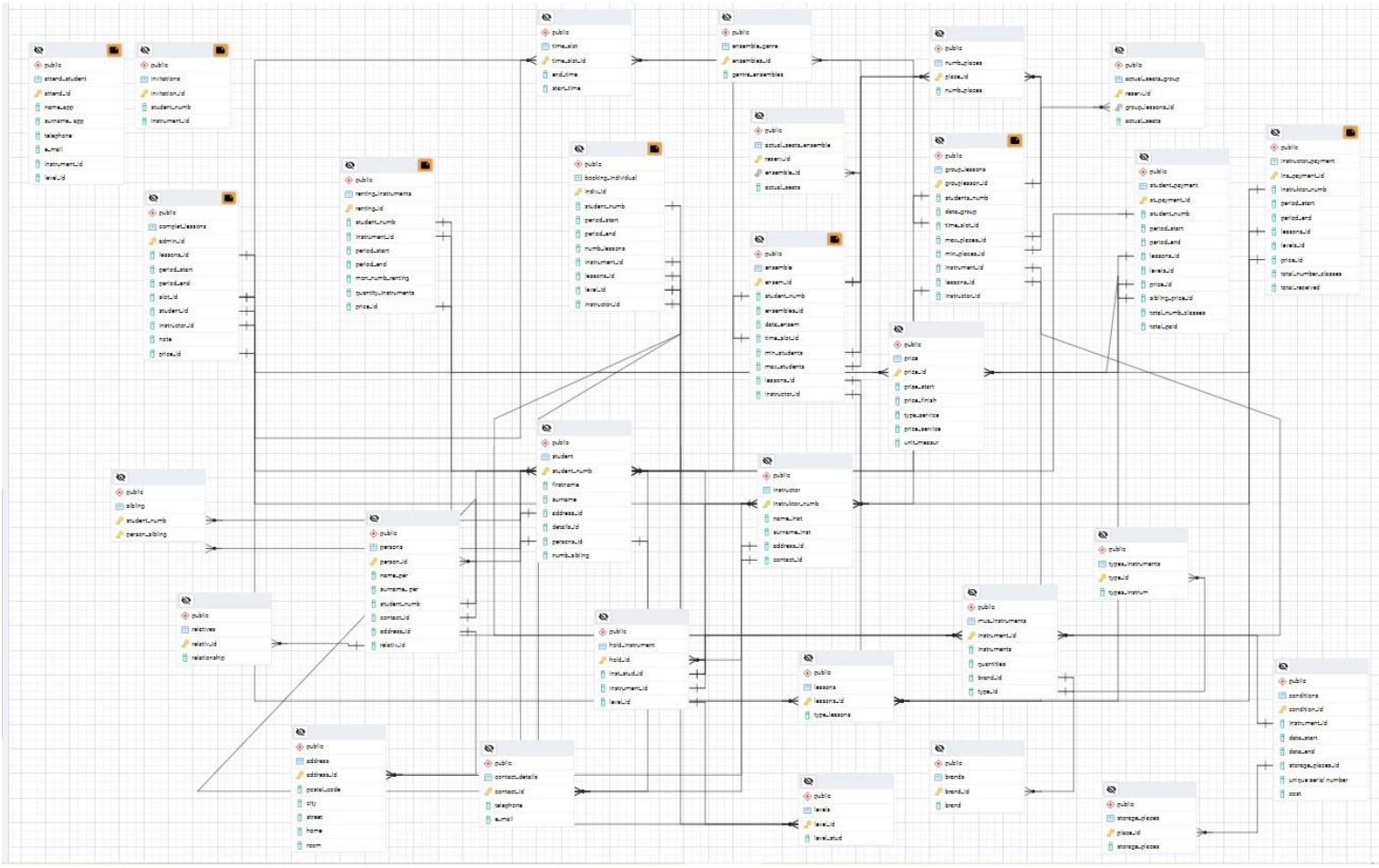
Instructor - Instrument
1..M: An instructor can teach multiple instruments.
1..M: Several instructors can teach the same musical instrument.

Lesson - Payment
1..M: A payment can include multiple lessons.
1..M: A lesson can be part of multiple payments.

Rental - Payment
1.. M: A payment can include multiple rentals.
1..M: A rental can be part of multiple payments.

Entity–Relationship (ER) Model, using crow's Foot Notation of the Soundgood Music School.
https://github.com/Lana-1167/KTX-1167.git

the data is in the folder https://github.com/Lana-1167/KTX-1167.git /seminar1/



Remark from Leif Lindbäck from 12 Nov at 18:12
The inheritance example is incorrect. There's no such thing as "inheritance via composition", your example shows just composition. For example, it's not true that "an address is a student", which means address can't inherit student.

Inheritance in a Relational Database the Soundgood Music School

Conceptual model with inheritance

Description:

In a model with inheritance, we extract common attributes of entities (student, instructor, persons_contact) into one base entity, for example, person. Attributes unique to each of the entities are extracted into child entities.

Example:

Base entity: person

Common attributes: person_id, name, surname, address_id, contact_id.

Child entities:

student: inherited from person, added student_numb attributes.

instructor: inherited from person, added instructor_numb attributes.

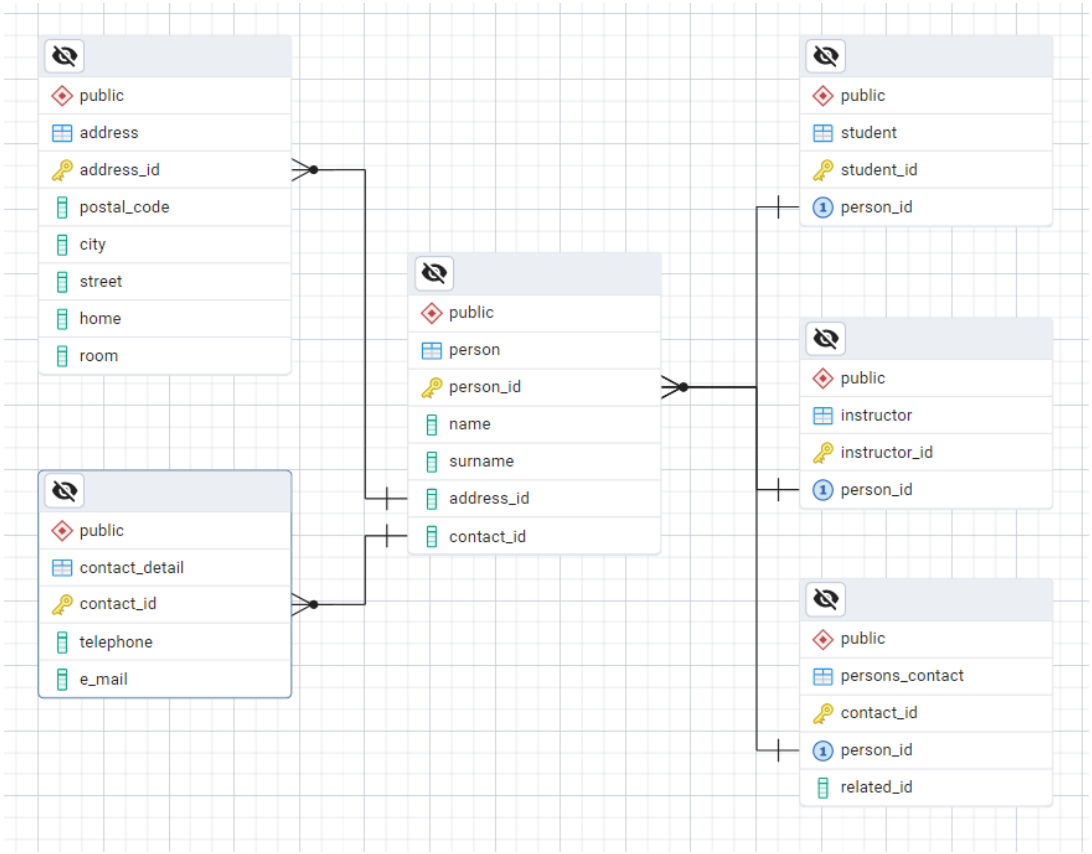
persons_contact: inherited from person, added related_id attributes.

ER diagram with inheritance:

Central entity: person (inherited by all child entities).

Child entities are related to person through inheritance.

Conceptual model with inheritance Entity–Relationship (ER) Model



SQL example:

```
-- Таблица student
CREATE TABLE student (
   student_id SERIAL PRIMARY KEY,
   person_id INT NOT NULL UNIQUE REFERENCES person(person_id)
);

-- Таблица instructor
CREATE TABLE instructor (
   instructor_id SERIAL PRIMARY KEY,
   person_id INT NOT NULL UNIQUE REFERENCES person(person_id)
);

-- Таблица persons_contact
CREATE TABLE persons_contact (
   contact_id SERIAL PRIMARY KEY,
   person_id INT NOT NULL UNIQUE REFERENCES person(person_id),
   related_id INT NOT NULL
);
```

Conceptual model without inheritance

Description:
The model without inheritance uses direct relationships between the student, instructor, persons_contact tables and the address and contact_details tables. Each table stores its own unique attributes. Common data such as address_id and contact_id are stored as foreign keys.

Example:

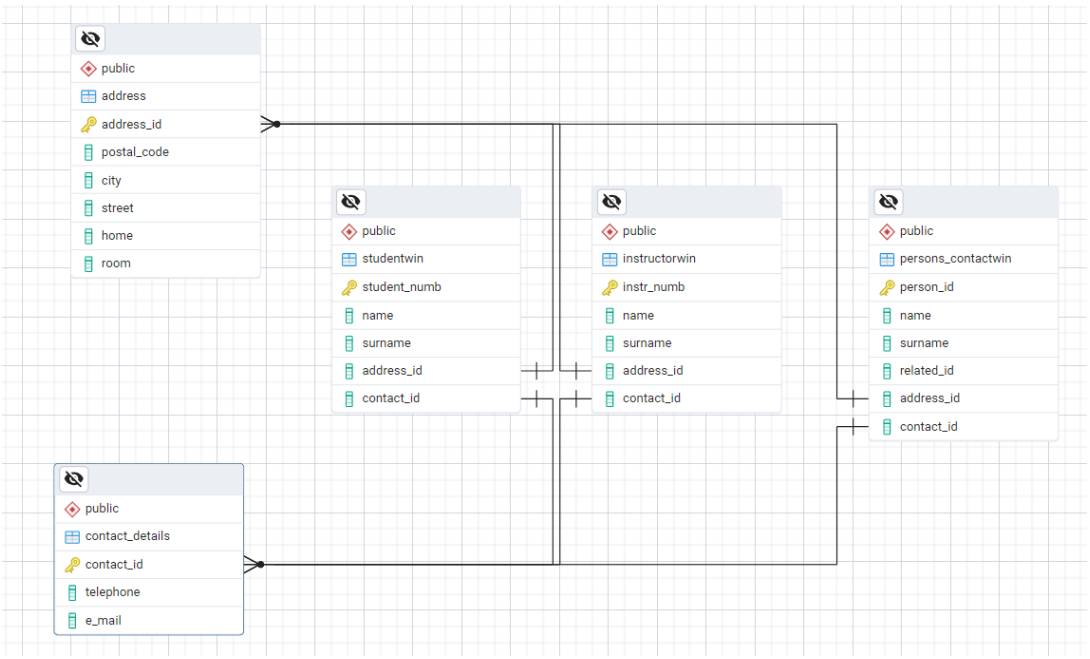
1. **Tables:**

    - student: stores student data and contains links to address and contact_details.
    - instructor: stores instructor data, also contains links to address and contact_details.
    - persons_contact: stores contact data.

ER diagram without inheritance:

- The tables student, instructor, persons_contact have direct relationships with the tables address and contact_details.

Conceptual model without inheritance Entity–Relationship (ER) Model



SQL example:

```
-- Таблица student
CREATE TABLE student (
   student_numb SERIAL PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   surname VARCHAR(50) NOT NULL,
   address_id INT NOT NULL,
   contact_id INT NOT NULL,
   FOREIGN KEY (address_id) REFERENCES address(address_id),
   FOREIGN KEY (contact_id) REFERENCES contact_details(contact_id)
);

-- Таблица instructor
CREATE TABLE instructor (
   instr_numb SERIAL PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   surname VARCHAR(50) NOT NULL,
   address_id INT NOT NULL,
   contact_id INT NOT NULL,
   FOREIGN KEY (address_id) REFERENCES address(address_id),
   FOREIGN KEY (contact_id) REFERENCES contact_details(contact_id)
);

-- Таблица persons_contact
CREATE TABLE persons_contact (
   person_id SERIAL PRIMARY KEY,
   name VARCHAR(50) NOT NULL,
   surname VARCHAR(50) NOT NULL,
   related_id INT NOT NULL,
   address_id INT NOT NULL,
   contact_id INT NOT NULL,
   FOREIGN KEY (address_id) REFERENCES address(address_id),
   FOREIGN KEY (contact_id) REFERENCES contact_details(contact_id)
);
```

5 Discussion

1. **The Benefit of Using Inheritance in the Soundgood Music School Database**

   Comparison of two approaches:

Criterion	Model with inheritance	Model without inheritance
Simplicity of design.	More complex due to hierarchical structure.	Simpler and more intuitive.
Data normalization.	Better normalization (minimize duplication).	Duplicate data is possible.
Performance.	Depends on implementation, JOINs can be more complex.	JOINs are simpler because the structures are linear.
Extensibility.	Easier to add new entities.	More changes required.

The recommended approach depends on the complexity of the project and scalability requirements.

However, in most cases it is recommended to use FOREIGN KEY relationships, as this provides greater flexibility and data consistency.

In the provided solution, there is no explicit inheritance in terms of relational databases in the description of table structures. However, there are connections between tables via foreign keys (FOREIGN KEY) and a hierarchical structure via attributes. This may be taken as a partial form of composition or association, but it is not inheritance.

2. **Does the CM contain all information needed by Soundgood?**

A conceptual data model is a high-level abstract representation of an organization's data. This model focuses on capturing entities, their attributes, and relationships without specifying any implementation details.

3. **Is it easy, that is a reasonable number of hops, to collect information related to all of the major entities (student, lesson, instructor, etc).**

The task is quite labor-intensive. It involves the use of 4 mandatory steps.

Identify objects: List the key objects that will be included in the database. Determine which objects are of primary importance and require storage and retrieval.

Define Attributes: Define the attributes of each entity that are relevant to the scope of your data model. Use the basic properties of each entity without going into details such as data types or constraints.

Establish relationships: Analyze the relationships between objects and identify existing relationships, ensuring that the proposed relationships make sense from a business perspective.

Review and refine. Analyze the original conceptual model for inconsistencies, redundancies, and missing information. Update the model as needed to improve its accuracy and completeness.

4. **Does the CM have a reasonable number of entities? Are important entities missing? Are there irrelevant entities, for example entities without attributes?**

In the solution presented here, the CM has a reasonable number of entities. All the important entities are present in this solution. For example, student, lessons, instructor, etc. There are no irrelevant entities in this solution, such as entities without attributes. Useless or irrelevant entities do not have any meaningful value or significance for the context or purpose of the database.

5. **Are there attributes for all data that shall be stored? Do all attributes have cardinality? Is the cardinality correct? Are the correct attributes marked as NOT NULL and/or UNIQUE?**

In the solution presented here, there are attributes for all the data that needs to be stored. This includes student data, instructors, student and instructor fees, musical instrument rentals, etc. Cardinality is the uniqueness of the data values contained in a column. Not all attributes have a cardinality. For example, phone or email in the contact_details relationship do not have a cardinality. In other entities, all attributes have the correct cardinality. Attributes that have the correct cardinality are marked as NOT NULL and/or UNIQUE.

6. **Does the CM have a reasonable number of relations? Are important relations missing? Are there irrelevant relations? Do all relations have cardinality at both ends and name at least at one end?**

Yes, the developed CM has a reasonable number of links. Important links are present. There are no irrelevant links. The power is the number of tuples in the relation (in simple terms, the number of records), the power of the relation can be any (from 0 to infinity), the order of the tuples is not important. Yes, all links have a power at both ends and a name at least at one end.

7. **Are naming conventions followed? Are all names sufficiently explaining?**

This solution follows naming conventions. All names are self-explanatory.

8. **Is the notation (UML or crow foot) correctly followed?**

Yes, the notation is correct. But the project used Entity–Relationship (ER) Model, using crow's Foot Notation.

9. **Are all business rules and constraints that are not visible in the diagram explained in plain text?**

All business rules and constraints that are not visible in the diagram are explained in plain text.

10 **Is the method and result explained in the report? Is there a discussion? Is the discussion relevant?**

Yes, the report explains the method and the result. There is a discussion that is relevant.

6 Comments About the Course

The conceptual data modeling assignment, including lectures, installation and learning of the required software, practical modeling, and preparation for the workshop, has taken up all the time since the beginning of the course. But since conceptual data modeling is the first step in the data modeling process, I believe that solving the practical problem will greatly help me in my future practice.