TU WIEN Informatics

SQL Import and Export Support for the Hybrid VS Code Modeling Tool bigER

BACHELOR'S THESIS

submitted in partial fulfillment of the requirements for the degree of

Bachelor of Science

in

Software- and Information Engineering

by

Christoph Lauscher
Registration Number 0925501

to the Faculty of Informatics

at the TU Wien

Advisor: Assistant Prof. Dipl.-Wirtsch.Inf.Univ. Dr.rer.pol. Dominik Bork

Vienna, 15th December, 2023

Christoph Lauscher	Dominik Bork

Erklärung zur Verfassung der Arbeit

Christoph Lauscher

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 15. Dezember 2023

Christoph Lauscher

Acknowledgements

I would like to thank my advisor Dominik Bork, who provided me with the opportunity to write my first bachelor thesis and the possibility to work on an active project with a growing community of users and for the support and encouragement during the process of writing this paper.

I want to thank Philipp-Lorenz Glaser for providing technical support and architectural guidance during the development of the functionality extensions for BIGER.

Abstract

Data modeling and tools to handle the models efficiently play an increasingly important role in the development and maintenance of all varieties of applications. The Entity-Relationship (ER) diagram was introduced to support the design phase of the data model, the mapping to the technical model is straight-forward. The BIGER modeling tool is a hybrid ER diagram editor with textual and graphical input options, that provides support for a default notation, as well as Bachman, Chen, Crow's Foot, Min-Max and UML. BIGER already provides limited SQL export functionality. In this work, a vendor-specific export and an import functionality are added for Oracle, MySQL, Microsoft SQL Server, PostgreSQL and IBM Db2, with future extensions in mind. The existing ER-capable tools ERD Preview, dBizzy, ApexSQL, MySQL workbench, DBeaver, vuerd and ERDSL are evaluated and compared to the BIGER modeling tool. We expect to enable even more users to design, maintain and migrate their models and databases with this tool.

Contents

CHAPTER 1

Introduction

At the heart of each application is the data, that is being processed. A good data model helps to ease the understanding and communication across different departments as well as different development teams[4]. The Entity-Relationship (ER) diagram [3] was introduced to support the design phase of the data model [12]. The Relational database model is the technical implementation of the abstract ER diagram. The mapping to the technical model is straight-forward, because of the similar structure of the ER model.

The BIGER modeling tool[1] [7][6] is a hybrid ER diagram editor with textual and graphical input options, that provides support for multiple notations. BIGER already provides limited SQL export functionality. In this work, a vendor-specific export and an import functionality is added.

Converting an ER diagram into a technical model and exporting it into vendor-specific SQL statements is an essential feature for the users of BIGER. It should enable the user to easily propagate changes in the diagram quickly into the actual implementation. This speeds up new developments and fixes by minimizing the need for manual adaptations. The creation of an ER diagram out of SQL vendor-specific statements is useful for reverse-engineering, documenting or refactoring existing software. As the vendors use sometimes different concepts, notations or keywords, this work aims to make working with these differences as transparent and effortless as possible for the BIGER users.

In chapter 2 the basis and required background information is covered. This reaches from basic data modeling to the frameworks used in BIGER. Chapter 3 provides an evaluation of several existing ER tools and presents more details of the vuerd tool, which offers a similar feature set as BIGER.

[1] `https://github.com/borkdominik/bigER`
last accessed 28.09.2023

In chapter 4 we will focus on the extension of the BIGER modeling tool. The chapter is structured into the architecture of import and export and a list of specific challenges that were addressed. Chapter 5 contains our observations, a short summary and an outlook on potential future work.

CHAPTER 2

Background

In this chapter, the basis and required background information to implement vendor-specific SQL import and export features, is covered. This reaches from basic data modeling to the frameworks used in BIGER.

2.1 Data Modeling

At the heart of each application is the data, that is being processed. Real data is often complex and unstructured. To handle this data efficiently, the relevant parts need to be modeled in a structured way. This is a strict requirement for approaches like data-driven development [9].

An application is only as good as the underlying data model. A good data model helps to ease the understanding and communication across different departments of a company, e.g. analysts/architects and developers, as well as different development teams working together [4]. Generally, the data model is used throughout the whole lifecycle of an application and even beyond. It is essential for the initial setup of a software architecture, the development of new features and understanding of error cases. In case an application reaches the end of its lifecycle and needs to be replaced, the migration of the data into the new system is eased, if the old data model is already known and can be reused or efficiently transformed into the new data model, according to the potentially changed requirements.

2.2 Entity-Relationship Modeling

The Entity-Relationship (ER) diagram was introduced by Peter Chen in 1976 [3] to support the design phase of the data model [12]. Over time it was extended to cover more use cases, but the original concept is still useful to structure data and the relations

between different parts of the data.

The main components are:

- **attribute**: A single value that describes one property of an object.
- **entity**: An entity should be directly derived from a real-world object, that is being modeled. It consists of a set of attributes, defining the object in the desired level of detail.
- **key**: This is a minimal sub-set of the entity attributes, uniquely identifying the entity. An entity can have multiple keys for identification, but only one primary key, that is later used by the underlying database to check for uniqueness.
- **relationship**: Entities are connected via relationships. There are different types of relationships, depending on the cardinality between the entities and whether an entity is dependent on another entity or can exist on its own. To further describe a relationship, it can have attributes too.

In the following section, a running example is used to illustrate the differences in output formats for different vendors. The representation of the example in the ER diagram is shown in Figure 2.1. For the full ER diagram model, see file A.1 in Appendix A.

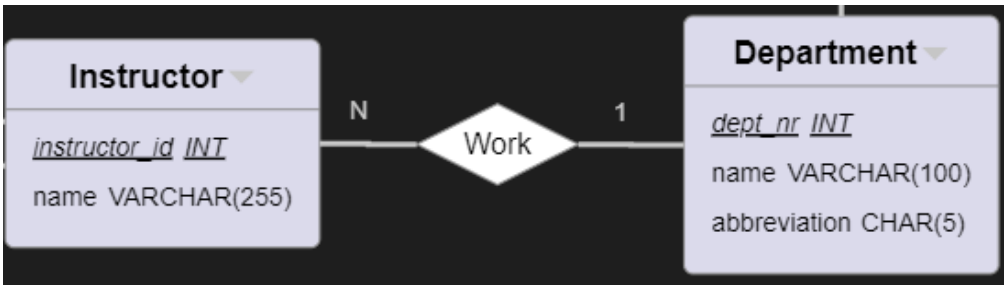

Figure 2.1: bigER diagram representation of running example

2.3 Relational database model

The Relational database model is the technical implementation of the abstract ER diagram. Relational databases follow a strict structure scheme, this enables the use of the mostly vendor-agnostic Structured Query Language (SQL) [10]. The mapping to the technical model should be easy in theory, because of the similar structure of the ER model. Only for some abstract concepts, a corresponding technical component has to be introduced, e.g. artificial tables for some relationships. There are several prominent vendors of relational database model systems (RDBMS), see Figure 4.1 for a popularity

ranking of RDBMS vendors from September 2022[1]. In the following we will inspect the first five vendors from the list in detail.

The different vendors support slightly different dialects of the SQL language, often only in specific versions of the specific dialect. The basic features are very similar in all dialects, but features introduced in a later version - or for a specific vendor only - make a cross-vendor support more complex. A number of related problems will be addressed in this work.

☐ sekundäre Datenbankmodelle berücksichtigen — 162 Systeme im Ranking, September 2022

Rang Sep 2022	Rang Aug 2022	Rang Sep 2021	DBMS	Datenbankmodell	Punkte Sep 2022	Punkte Aug 2022	Punkte Sep 2021
1.	1.	1.	Oracle	Relational, Multi-Model	1238,25	-22,54	-33,29
2.	2.	2.	MySQL	Relational, Multi-Model	1212,47	+9,61	-0,06
3.	3.	3.	Microsoft SQL Server	Relational, Multi-Model	926,30	-18,66	-44,55
4.	4.	4.	PostgreSQL	Relational, Multi-Model	620,46	+2,46	+42,95
5.	5.	5.	IBM Db2	Relational, Multi-Model	151,39	-5,83	-15,16
6.	6.	↑ 7.	Microsoft Access	Relational	140,03	-6,47	+23,09
7.	7.	↓ 6.	SQLite	Relational	138,82	-0,05	+10,17
8.	8.	8.	MariaDB	Relational, Multi-Model	110,16	-3,74	+9,46
9.	9.	↑ 14.	Snowflake	Relational	103,50	+0,38	+51,43
10.	10.	10.	Microsoft Azure SQL Database	Relational, Multi-Model	84,42	-1,75	+6,16

Figure 2.2: Popularity ranking of RDBMS vendors from September 2022

2.3.1 Oracle

Oracle offers its relational database under a free licence, named Oracle Database Express Edition (XE)[2]. With the Oracle SQL Developer, it is possible to export data into XLS or CSV format, or metadata and data into SQL format[3], see file B.4 in Appendix B.

The Oracle output format uses double quotes for schema, table and attribute names. All Primary and Foreign Keys are defined in separate ALTER TABLE statements. Size constraints of numeric data types can be defined in a wildcard notation. See Listing 2.1 for an example.

[1] https://db-engines.com/de/ranking/relational+dbms
last accessed 20.09.2022

[2] https://www.oracle.com/database/technologies/appdev/xe.html
last accessed 23.09.2022

[3] https://docs.oracle.com/cd/E17781_01/server.112/e18804/impexp.htm
last accessed 23.09.2022

```
--------------------------------------------------------
-- DDL for Table WORK
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."WORK"
   ( "INSTRUCTOR_ID" NUMBER(*,0),
     "DEPT_NR" NUMBER(*,0)
   ) [...];

--------------------------------------------------------
-- Constraints for Table WORK
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."WORK" ADD PRIMARY KEY ("INSTRUCTOR_ID",
     ↪ "DEPT_NR")
  [...];

--------------------------------------------------------
-- Ref Constraints for Table WORK
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."WORK" ADD FOREIGN KEY ("INSTRUCTOR_ID")
        REFERENCES "SYSTEM"."INSTRUCTOR" ("INSTRUCTOR_ID") ON
           ↪ DELETE CASCADE [...];
  ALTER TABLE "SYSTEM"."WORK" ADD FOREIGN KEY ("DEPT_NR")
        REFERENCES "SYSTEM"."DEPARTMENT" ("DEPT_NR") ON DELETE
           ↪ CASCADE [...];
```

Listing 2.1: Oracle output format

2.3.2 MySQL

The MySQL Community Edition is free to use[4] and provides export of data into JSON or CSV format, or metadata and data to SQL format via the MySQL Workbench[5], see file B.3 in Appendix B.

The MySQL output format uses back-ticks for name quoting. The Primary and Foreign Keys are defined together with the attributes in the CREATE TABLE statements, the Foreign Key constraints are named automatically. See Listing 2.2 for an example.

[4] https://www.mysql.com/de/products/community/
last accessed 23.09.2022

[5] https://dev.mysql.com/doc/workbench/en/wb-admin-export-import.html
last accessed 23.09.2022

```
CREATE TABLE `Work` (
  `instructor_id` int(11) NOT NULL,
  `dept_nr` int(11) NOT NULL,
  PRIMARY KEY (`instructor_id`,`dept_nr`),
  CONSTRAINT `Work_ibfk_1` FOREIGN KEY (`instructor_id`)
    ↪ REFERENCES `Instructor` (`instructor_id`) ON DELETE
    ↪ CASCADE,
  CONSTRAINT `Work_ibfk_2` FOREIGN KEY (`dept_nr`) REFERENCES `
    ↪ Department` (`dept_nr`) ON DELETE CASCADE
) [...];
```

Listing 2.2: MySQL output format

2.3.3 Microsoft SQL Server

Microsoft offers the SQL Server Express edition[6] as a free database. With the SQL Server Management Studio, it is possible to export data into XLS, CSV, text format or directly to another database using the correct providers/drivers via SQL Server Import and Export Wizard[7], or metadata and data into SQL format, see file B.2 in Appendix B.

See Listing 2.3 for an example of the Microsoft SQL output format. Identifier names are wrapped in square brackets. The Primary Key is defined together with the attributes in the CREATE TABLE statements. The Foreign Keys are defined in separate ALTER TABLE statements.

[6] https://learn.microsoft.com/en-us/sql/sql-server/editions-and-components-of-sql-server-2019?view=sql-server-ver15
last accessed 23.09.2022

[7] https://learn.microsoft.com/en-us/sql/integration-services/import-export-data/connect-to-data-sources-with-the-sql-server-import-and-export-wizard?view=sql-server-ver16
last accessed 23.09.2022

```
CREATE TABLE [dbo].[Work](
      [instructor_id] [int] NOT NULL,
      [dept_nr] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [instructor_id] ASC,
      [dept_nr] ASC
)[...]

ALTER TABLE [dbo].[Work] WITH CHECK ADD FOREIGN KEY([dept_nr])
REFERENCES [dbo].[Department] ([dept_nr])

ALTER TABLE [dbo].[Work] WITH CHECK ADD FOREIGN KEY([
   ↪ instructor_id])
REFERENCES [dbo].[Instructor] ([instructor_id])
```

Listing 2.3: MS SQL output format

2.3.4 PostgreSQL

The PostgreSQL database is open source under the PostgresSQL Licence[8]. Various internal command-line tools (e.g. pg_dump/psql) as well as pgAdmin support export of data into binary, text or CSV format, or metadata and data to SQL format[9], see files B.5 and B.6 in Appendix B.

See Listing 2.4 for an example of the PostgreSQL output format. By default no quoting is applied to the identifier names. As the Oracle format, the Primary and Foreign Keys are defined in separate ALTER TABLE statements, the Foreign Key constraints are named automatically. Additionally to these, there can be further ALTER TABLE statements, not needed for the ER diagram.

[8] https://www.postgresql.org/about/licence/
last accessed 23.09.2022

[9] https://www.a2hosting.com/kb/developer-corner/postgresql/import-and-export-a-postgresql-database/
last accessed 23.09.2022

```
CREATE TABLE public.work (
   instructor_id integer NOT NULL,
   dept_nr integer NOT NULL
);

ALTER TABLE public.work OWNER TO postgres;

ALTER TABLE ONLY public.work
   ADD CONSTRAINT work_pkey PRIMARY KEY (instructor_id, dept_nr
      ↪ );

ALTER TABLE ONLY public.work
   ADD CONSTRAINT work_dept_nr_fkey FOREIGN KEY (dept_nr)
      ↪ REFERENCES public.department(dept_nr);

ALTER TABLE ONLY public.work
   ADD CONSTRAINT work_instructor_id_fkey FOREIGN KEY (
      ↪ instructor_id) REFERENCES public.instructor(
      ↪ instructor_id);
```

Listing 2.4: PostgreSQL output format

2.3.5 IBM Db2

IBM offers the Db2 Community Edition as well as free cloud trials[10]. Various internal command-line tools (e.g. db2look) as well as the Db2 Control Center support export of data into text or CSV format, or metadata into the Integration Exchange Format (IXF) or SQL format[11], see file B.1 in Appendix B.

The Db2 output format uses double quotes for identifier names, see Listing 2.5 for an example. As the Oracle and PostgreSQL format, the Primary and Foreign Keys are defined in separate ALTER TABLE statements. Again, the Foreign Key constraints are named automatically.

[10] https://www.ibm.com/products/db2/pricing
last accessed 23.09.2022

[11] https://www.ibm.com/docs/en/db2/11.5?topic=types-exportimportload-utility-file-formats
last accessed 23.09.2022

```
------------------------------------------------
-- DDL-Anweisungen fuer Tabelle "DB2ADMIN"."WORK"
------------------------------------------------

CREATE TABLE "DB2ADMIN"."WORK" (
              "INSTRUCTOR_ID" BIGINT NOT NULL ,
              "DEPT_NR" BIGINT NOT NULL )
             [...];

-- DDL-Anweisungen fuer Primaerschluessel fuer Tabelle "
   ↪ DB2ADMIN"."WORK"

ALTER TABLE "DB2ADMIN"."WORK"
      ADD PRIMARY KEY
            ("INSTRUCTOR_ID",
             "DEPT_NR")
      [...];

-- DDL-Anweisungen fuer Fremdschluessel fuer Tabelle "DB2ADMIN
   ↪ "."WORK"

ALTER TABLE "DB2ADMIN"."WORK"
      ADD CONSTRAINT "SQL221210155240850" FOREIGN KEY
            ("INSTRUCTOR_ID")
      REFERENCES "DB2ADMIN"."INSTRUCTOR"
            ("INSTRUCTOR_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      [...];

ALTER TABLE "DB2ADMIN"."WORK"
      ADD CONSTRAINT "SQL221210155240860" FOREIGN KEY
            ("DEPT_NR")
      REFERENCES "DB2ADMIN"."DEPARTMENT"
            ("DEPT_NR")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      [...];
```

Listing 2.5: Db2 output format

2.3.6 Comparison

The various output formats have a similar structure, although they can be extended by various technical keywords, specific to the vendor. The key step in the parsing process is handling key definitions separated into ALTER TABLE statements.

Other considerations to cover across formats are:

- filtering quote characters
- processing size restrictions
- ignoring unused clauses and keywords

Table 2.1 shows a comparison of relevant differences across vendors.

	Oracle	MySQL	MS SQL	PostgreSQL	Db2
Naming convention	"WORK"	‘Work‘	[Work]	work	"WORK"
Primary Key in ALTER	+	-	-	+	+
Foreign Key in ALTER	+	-	+	+	+
Named Foreign Keys	-	+	-	+	+
Other ALTER statements	-	-	-	+	-

Table 2.1: Comparison of SQL output formats.

2.4 bigER modeling tool

The bigER modeling tool is a hybrid ER diagram editor with textual and graphical input options, that provides support for multiple notations, see Figure 2.3. Although provided as a Visual Studio (VS) Code extension[12], it is based on the Language Server Protocol (LSP) [2] to enable flexible use in other development environments [**?**].

bigER already provides limited SQL export functionality [5]. In this work, a vendor-specific export and an import functionality is added.

2.5 Xtext/Xtend

Xtext[13] allows to create a custom Domain Specific Language (DSL) with the possibility to automatically create parsers and code generators for the custom language. As an

[12] `https://marketplace.visualstudio.com/items?itemName=BIGModelingTools.erdiagram`
last accessed 26.10.2023

[13] `https://projects.eclipse.org/projects/modeling.tmf.xtext`
last accessed 26.10.2023

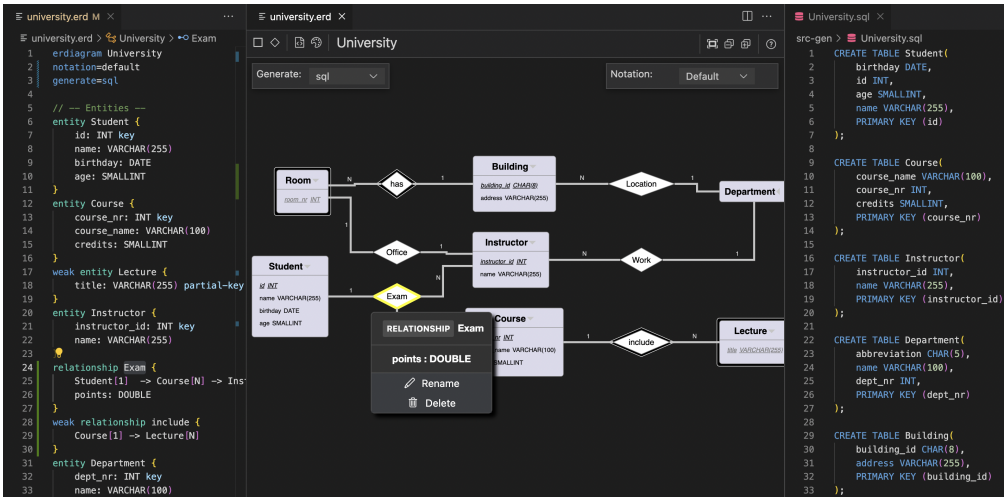

Figure 2.3: BIGER - Hybrid view of ER diagram in textual and graphical form [7]

Eclipse Framework it is able to fully integrate into the Eclipse IDE, including syntax highlighting, code completion and automatic builds [1]. BIGER implements a custom DSL for the textual representation of the ER diagram [5]. The SQL generator receives the fully parsed model as input, the SQL import produces an ER diagram in the textual form.

XTend[14] is a Java-like programming language, intended to provide language developers a more compact and easier to use syntax [1]. In this work, all components are implemented in regular Java, but had to be migrated in part from Xtend code.

[14] https://marketplace.eclipse.org/content/eclipse-xtend last accessed 26.10.2023

CHAPTER 3

Related work

Before the implementation, several existing ER tools were evaluated for their import and export capabilities. A brief overview of the evaluation results is provided in Table 3.1.

The evaluation criteria contain whether SQL import and export features are provided at all and whether these features offer support for multiple RDBMS vendors. Additionally it is considered whether the tool is available as a VS Code extension.

	SQL export	multi-vendor	SQL import	multi-vendor	VS Code extension
ERD Preview[1]	-	-	-	-	+
dBizzy[2]	-	-	+	?	+
MySQL workbench[3]	-	-	+	?	-
DBeaver[4]	-	-	+	?	-
vuerd[5]	+	+	+	~	+
ERDSL[6]	+	?	+	?	-
bigER	+	+	+	+	+

Table 3.1: Comparison of related tools.

[1] https://marketplace.visualstudio.com/items?itemName=kaishuu0123.vscode-erd-preview
last accessed 10.06.2023

[2] https://marketplace.visualstudio.com/items?itemName=dBizzy.dbizzy&ssr=false#overview
last accessed 10.06.2023

[3] https://dev.mysql.com/doc/workbench/en/wb-data-modeling-menus.html#wb-relationship-notation-menu
last accessed 10.06.2023

3.1 ERD Preview

This tool is a VS Code extension to preview ER diagrams, providing a very basic syntax to define tables, attributes and relationships. For visualization, only Crow's foot notation[11] is available for relationships. The diagram can be exported in various image formats. Prerequisite to run correctly, are the "erd" and "dot" programs. ERD Preview does not provide any SQL import or export features. For an example preview, see Figure 3.1.

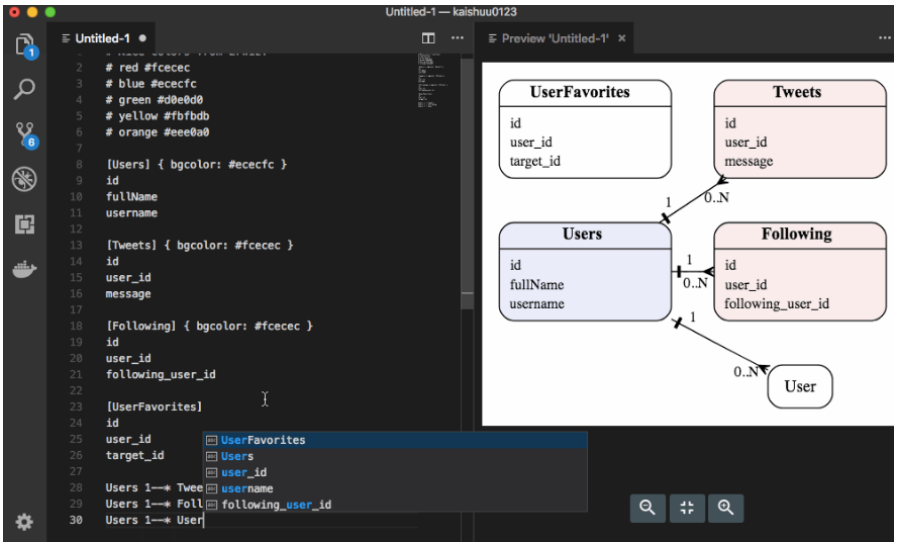

Figure 3.1: ERD Preview - Definition and visualization of an ER diagram

3.2 dBizzy

dBizzy is a VS Code extension to show a database definition in SQL form as an ER diagram, see Figure 3.2. The diagram can be exported in various image formats. Additionally it provides a database browser to query an in-memory database from the provided SQL structure. No SQL export functionality is available, the ER diagram shows no specific relationship notations.

3.3 MySQL workbench

MySQL workbench is the standard tool to interactively work with MySQL databases. Additionally to the standard features, like exploring the database and executing queries,

[4] https://dbeaver.com/docs/wiki/Custom-Diagrams/
last accessed 10.06.2023

[5] https://marketplace.visualstudio.com/items?itemName=dineug.vuerd-vscode
last accessed 10.06.2023

[6] https://github.com/ProjetoDSL/ERDSL
last accessed 10.06.2023

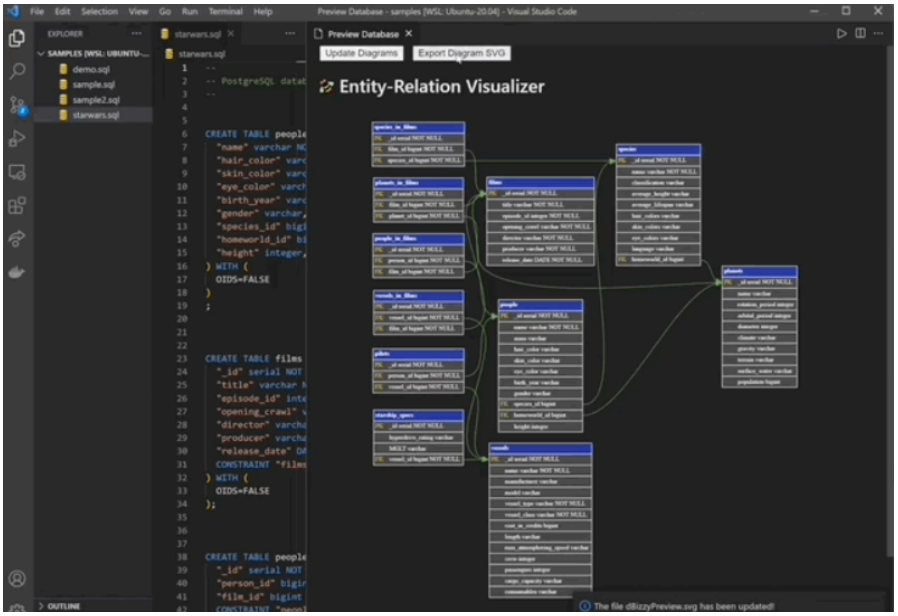

Figure 3.2: dBizzy - Showing an ER diagram from the SQL structure definition

it provides the possibility to create an ER diagram from an existing database, see Figure 3.3. The diagram can be exported in various image and document formats. The feature is limited to already existing databases, it cannot create the diagram from an SQL definition.

3.4 DBeaver

DBeaver is a standard tool to access existing databases too. It also offers a possibility to create custom ER diagrams from an existing database, see Figure 3.4. Changes done to the diagram can be automatically applied to the actual database. Similar to other tools, only existing databases can be used to create diagrams, no SQL definitions can be imported.

3.5 ERDSL

This project defines its own grammar to define ER diagrams. It supports the basic features like entity and relationship definitions, but misses some advanced functionality like weak entities or optional attributes[7]. The diagram can be exported to SQL format for MySQL and PostgreSQL, see Figure 3.5. As this project is still in development, some features may be added in the future.

[7] `https://github.com/ProjetoDSL/ERDSL/wiki/Features`
last accessed 14.12.2023

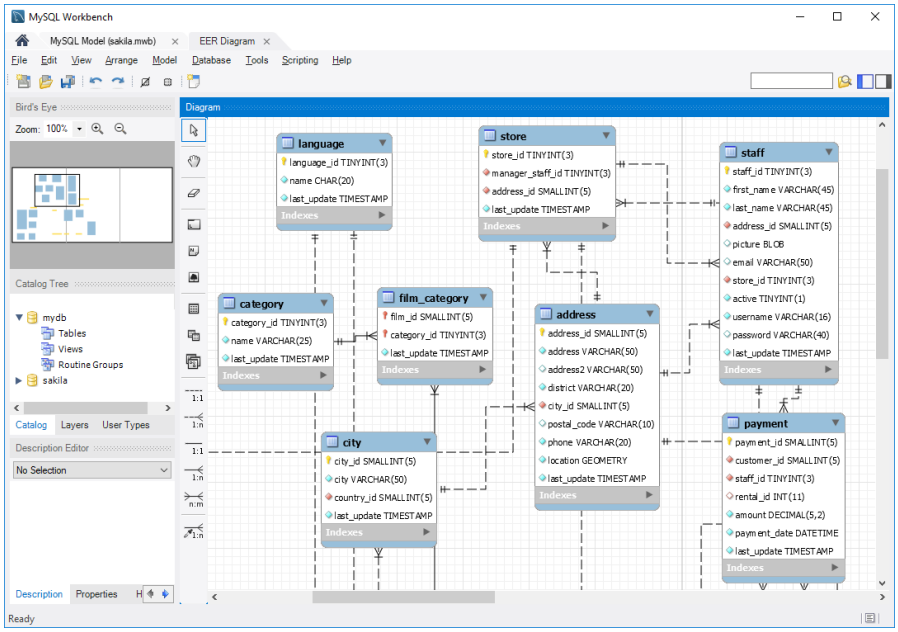

Figure 3.3: MySQL workbench - Showing an ER diagram created from an existing database

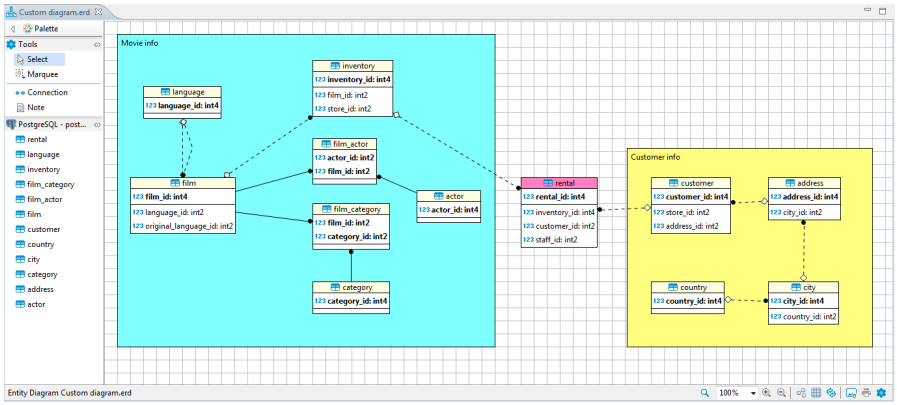

Figure 3.4: DBeaver - Showing an ER diagram created from an existing database

3.6 vuerd

The vuerd tool (now renamed to erd-editor[8]) offers an ER diagram editor, where entities, relationships and their attributes can be managed. It supports the definition of primary keys, foreign keys and the nullability of the attributes. Other than BIGER, default values and comments for attributes and entities can be specified. The attribute types can be selected from a pre-defined list or entered manually, see Figure 3.6.

[8] `https://github.com/dineug/erd-editor`
last accessed 10.06.2023

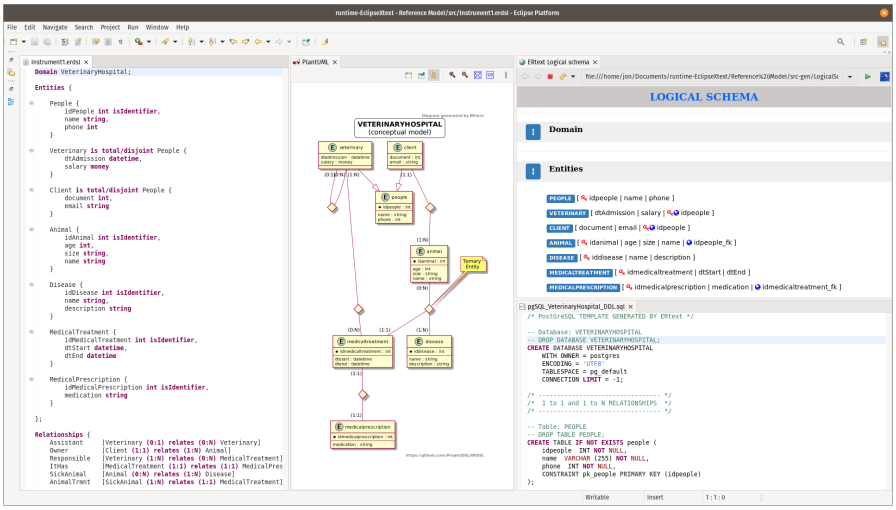

Figure 3.5: ERDSL - Hybrid view of an ER diagram and the corresponding SQL definition

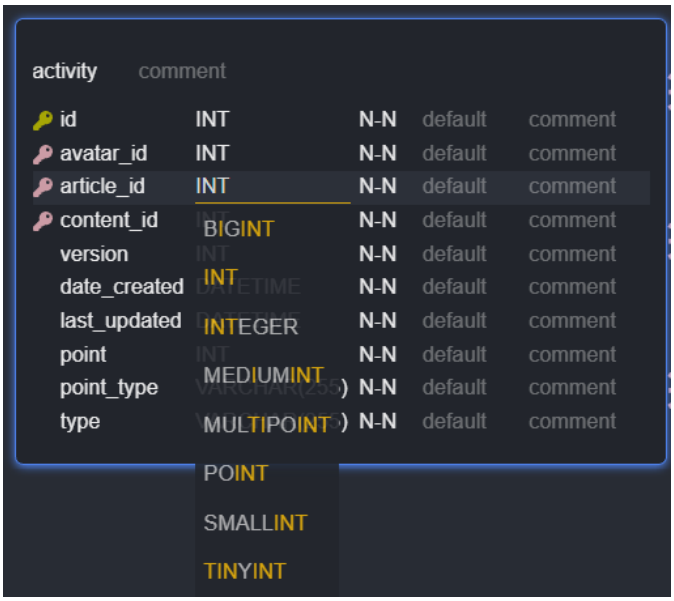

Figure 3.6: vuerd - View of a table in the ER diagram

Currently only Crow's foot notation[11] is available for relationships, see Figure 3.7.

The ER diagram can be exported into SQL DDL statements in vendor-specific formats. Currently vuerd supports MariaDB, MSSQL, MySQL, Oracle, PostgreSQL and SQLite, see Figure 3.8.

Depending on the chosen vendor, the exported SQL is split into the table declaration and separate alter statements to define the foreign keys, see Figure 3.9. No transformations are done on the attribute name and type selected by the user, see Figure 3.10.

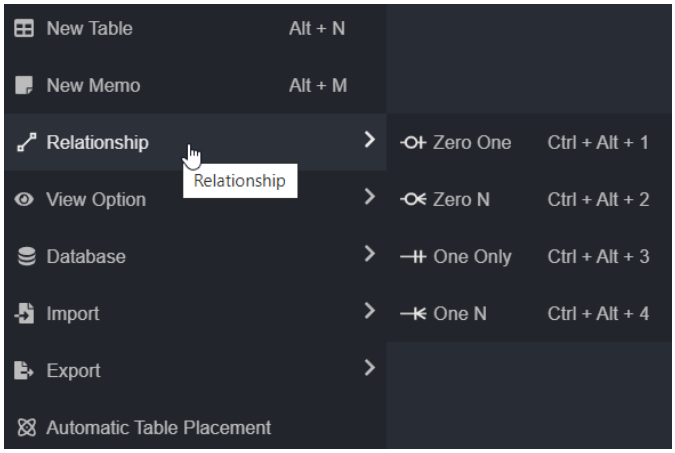

Figure 3.7: vuerd - Supported notation

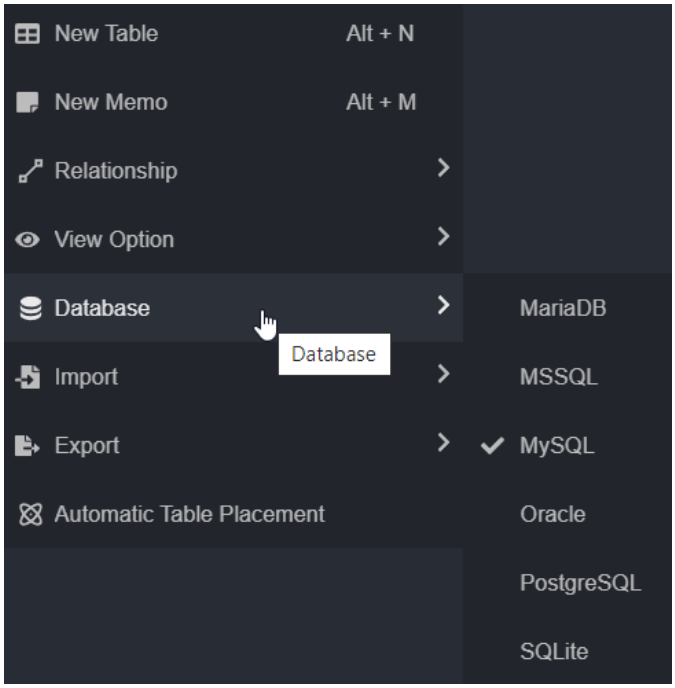

Figure 3.8: vuerd - Supported RDBMS vendors

```
CREATE TABLE activity
(
  id           INT           NOT NULL,
  AvAtAr_id    INT           NOT NULL,
  article_id   INT           NOT NULL,
  content_id   INT           NOT NULL,
  version      INT2          NOT NULL,
  date_created DATETIME      NOT NULL,
  last_updated DATETIME      NOT NULL,
  point        INT           NOT NULL,
  point_type   VARCHAR(255) NOT NULL,
  type         VARCHAR(255) NOT NULL,
  PRIMARY KEY (id)
);
```

Figure 3.9: vuerd - Result of the SQL export - table

```
ALTER TABLE activity
  ADD CONSTRAINT FK_avatar_TO_activity
    FOREIGN KEY (AvAtAr_id)
    REFERENCES avatar (id);
```

Figure 3.10: vuerd - Result of the SQL export - foreign key

CHAPTER 4

Import and export in vendor-specific formats

In this chapter we will focus on the extension of the BIGER modeling tool. We will start with an overview of the architecture and the general approach, that was chosen for the implementation. Then follows a list of specific features, that were implemented and those, that are out of scope for this work.

4.1 Architecture

The existing BIGER implementation only provided limited SQL export and no import functionality. To support provider-specific SQL output, the Xtend implementation was transformed into a regular Java format. Because of the similar input format for all chosen vendors, most features can be implemented in a generic generator class. Actual vendor-specifics can be implemented in sub-classes, see Figure 4.1. This transformation to regular Java code gives the developer more freedom in the usage of features from new Java versions. Additionally, it makes the code accessible to a broader audience of developers, as no deep knowledge of the Xtend framework is needed. Recently the future maintenance of the Xtend framework became questionable: "Briefly: The future maintenance of Xtext & especially Xtend is at risk."[1]

[1] https://eclipse.dev/Xtext/xtend/releasenotes.html#/releasenotes/2020/06/02/version-2-22-0
last accessed 04.05.2023

For the new import functionality, different options were considered:

- external library: JSQLParser
- external library: Eclipse SQL Query Parser
- custom Xtext grammar
- regular Java implementation

The external libraries were not able to process the output of the different vendors directly. Some keywords or new features are not known to the libraries, other features are too different across the vendors. The same reasons apply for any explicitly defined grammar. To be most flexible and because of the reasons already mentioned with the SQL export functionality, a regular Java implementation using Regular Expressions was chosen. Development and testing was supported by Regex101[2].

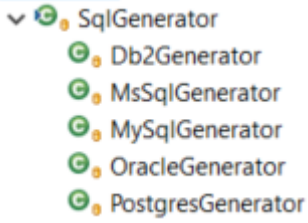

Figure 4.1: Type hierarchy of SQL generators

4.1.1 SQL export

The workflow of the SQL generator was in general not changed from the existing implementation. It receives the parsed Xtext model of the ER diagram as input, the output is a new file with .sql extension.

First the strong entities are processed, they have attributes and a primary key, but no foreign keys. In the next step, the weak relationships are processed. They have additionally a foreign key to other strong or weak entities. At last, the strong relationships are processed. These relationships can have up to three related entities.

[2] `https://regex101.com/`
last accessed 04.05.2023

4.1.2 SQL import

The SQL import is based on Regular Expressions (RegEx) to flexibly parse different vendor output formats, while being extendable to new vendors or features. Common parts are generalized to keep the expressions understandable, see Listing 4.1. Since some vendors split the table and key definitions, the importer has to be able to parse both, "create" and "alter" statements. Some format differences are not easily modeled in RegEx, e.g. line breaks because of different variants of line endings, so a simple pre-processing is done to bring it into a similar form, see Listing 4.2. Input for the importer is a .sql file, at least containing Data Definition Language (DDL) statements, output is a new file with .erd extension.

The first step is to search for "alter" statements and store the found key definitions for later use. In the next step, the actual "create" statements are processed. Either the previously stored key definitions are used or the keys are defined in the "create" statement itself. At last, the relationships are generated, where weak relationships are generated with default cardinalities.

```
private static final String SIZE_BASE_PATTERN = "\\d
    ↪ +(?:,\\s*\\d+)?";
private static final String SIZE_PATTERN = "\\(((?:" +
    ↪ SIZE_BASE_PATTERN + ")|\\*)";
private static final String ATTRIBUTE_PATTERN = "\\s
    ↪ *([^\\)\\s,]*)(?: (.*\\([\\d\\*]+.*?\\)|[^,\\s]+))
    ↪ ?[^,\\)]*,?\\s*(?:--(.*))?";

private static final String FOREIGN_KEY_BASE_PATTERN = "
    ↪ FOREIGN KEY\\s*\\((.*?)\\)\\s*REFERENCES (?:.+?\\.)
    ↪ ?(\\S+)\\s*\\((.*?)\\)";
private static final String FOREIGN_KEY_PATTERN = ".*?" +
    ↪  FOREIGN_KEY_BASE_PATTERN + "([^,]*?)";

private static final String PRIMARY_KEY_BASE_PATTERN = "
    ↪ PRIMARY KEY(?: CLUSTERED)?\\s*\\((.*?)\\)";
private static final String PRIMARY_KEY_PATTERN = ".*" +
    ↪ PRIMARY_KEY_BASE_PATTERN + "(?:WITH.+?\\(.*?\\))
    ↪ ?[^,\\)]*?";
private static final String UNIQUE_KEY_PATTERN = ".*
    ↪ UNIQUE KEY[^,\\)]*?";
```

Listing 4.1: Regular Expressions used for SQL import

```
    private String preprocessSql(String text) {
        // Oracle: insert line break
        text = text.replace("(\t", "(\r\n\t");
        // MsSql: combine primary key and foreign key
           ↪ clauses into a single line
        Pattern pPrimaryKey = Pattern.compile(
           ↪ PRIMARY_KEY_BASE_PATTERN, Pattern.
           ↪ CASE_INSENSITIVE | Pattern.DOTALL);
        Matcher mPrimaryKey = pPrimaryKey.matcher(text);
        while (mPrimaryKey.find()) {
            String originalText = mPrimaryKey.group();
            String replacedText = removeNewlines(
               ↪ originalText);
            text = text.replace(originalText,
               ↪ replacedText);
        }
        Pattern pForeignKey = Pattern.compile(
           ↪ FOREIGN_KEY_BASE_PATTERN, Pattern.
           ↪ CASE_INSENSITIVE | Pattern.DOTALL);
        Matcher mForeignKey = pForeignKey.matcher(text);
        while (mForeignKey.find()) {
            String originalText = mForeignKey.group();
            String replacedText = removeNewlines(
               ↪ originalText);
            text = text.replace(originalText,
               ↪ replacedText);
        }
        return removeNewlines(text, REPLACE_NEWLINES);
    }
```

Listing 4.2: Pre-process input before using Regular Expressions

4.2 Manual tests

Manual testing is necessary to find errors and misconceptions in the initial implementation for each vendor. Using an appropriate architecture, automated tests should be enough to find regressions in the existing code. Further manual tests are then only necessary for new features and additional vendors. Table 4.1 shows an overview of the tools and version used for testing the SQL import and export functionalities for the different vendors.

Three main models were used for testing, which cover most of the necessary edge cases for the implemented features:

- university.erd: This is the biggest model, containing a realistic example of a complete database. It covers strong and weak tables and relationships, a three-way relationship and various data types. For the full ER diagram model, see file A.1 in Appendix A.

- relationships.erd: The model is mainly used to test relationships. All attributes in this model have the same name, which is useful for testing the de-duplication functionality. For the full ER diagram model, see file A.2 in Appendix A.

- basic.erd: This is a very basic example, covering only two tables and a relationship. Such a simple model is useful for verifying the basic understanding of the requirements at the beginning of the implementation phase. For the full ER diagram model, see file A.3 in Appendix A.

	Oracle	MySQL	MS SQL	PostgreSQL	Db2
DB Version	21c	5.7	15.0	15.1	11.5
Export Tool	SQL Developer	MySQL Workbench	SQL Server Management Studio	pg_dump	db2look
Version	22.2.1	6.3	15.0	15.1	11.5
Import Tool	SQL Developer	MySQL Workbench	SQL Server Management Studio	pgAdmin 4	IBM Db2 Data Management Console
Version	22.2.1	6.3	15.0	6.15	3.1.9

Table 4.1: Comparison of tools used for testing.

4.3 Unit tests

Automated tests are important to check if the implementation follows the specification, but even more important to detect regressions after adding new features and fixing errors. Especially for the work with regular expressions in the import process, it was essential to always test the correct parsing, when adapting them for additional vendor formats.

The implementation of the unit tests was done with adaptability and extendability in mind. For each test, an input file and an expected output file has to be provided, see Figure 4.2. The input files are processed in a loop, which can be easily extended for additional tests. For each test, the actual output is compared to the expected output.

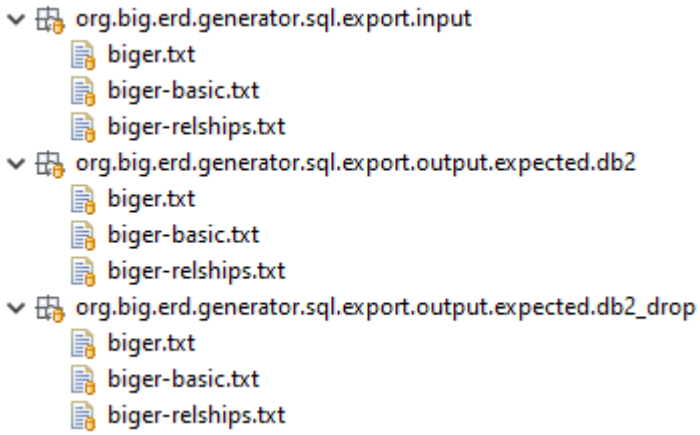

Figure 4.2: Input and expected output for all supported notations and vendors

4.4 Encountered challenges

4.4.1 Type mapping

All of the chosen RDBMS understand the basic "create" statement, so the SQL generator output is already compatible. The only prominent difference across all vendors are the data types. Although all vendors share a common concept of numerical, textual, binary and date/time data, the actual implementations use different names and size categories[3].

To be able to generate valid SQL code for all vendors, an explicit mapping of conceptual data category and vendor-specific data types has to be implemented. There could be a definitive mapping between all possible data types of all vendors. For bigER we chose a simpler approach, the relevant vendor-specific data types are only associated with a common data category, see Listing 4.3. When generating code for a RDBMS, that does not know that data type, a mapping has to be performed. The biggest data type of the corresponding data category is used as a replacement for the unknown type, see Listing 4.4. This way the replaced type is at least as big as the user defined data type, in some cases unnecessarily big.

The mapping is easily extendable by new vendors, data types and even additional data categories. In case no type is provided by the user, a default of VARCHAR(255) is assumed. In the abstract ER diagram, it is no problem to omit the data type. For the technical model, a data type is required.

[3] https://en.wikibooks.org/wiki/SQL_Dialects_Reference/Print_version last accessed 15.12.2022

```
public static final List<String> INTEGER_TYPES = Arrays.
    ↪ asList("BIGINT", "INT", "SMALLINT", "TINYINT");

public static final List<String> FLOAT_TYPES = Arrays.
    ↪ asList("FLOAT", "REAL");

public static final List<String> DECIMAL_TYPES = Arrays.
    ↪ asList("DECIMAL", "NUMERIC");

public static final List<String> ALL_NUMERIC_TYPES =
            Stream.concat(DECIMAL_TYPES.stream(), Stream.
                ↪ concat(FLOAT_TYPES.stream(),
                ↪ INTEGER_TYPES.stream()))
          .distinct()
          .collect(Collectors.toList());

public static final List<String> ALL_TYPES =
            Stream.concat(ALL_NUMERIC_TYPES.stream(),
                ↪ Stream.concat(ALL_CHARACTER_TYPES.stream
                ↪ (), Stream.concat(ALL_DATE_TYPES.stream
                ↪ (), Stream.concat(ALL_BINARY_TYPES.
                ↪ stream(), ALL_BOOLEAN_TYPES.stream()))))
          .distinct()
          .collect(Collectors.toList());
```

Listing 4.3: Mapping of Microsoft SQL data types to common data categories

```
        @Override
        protected String mapDataType(String type) {
                String upperType = type.toUpperCase();
                if (ALL_TYPES.contains(upperType)) {
                        return type;
                }

                // numeric types
                if (DataTypes.getAllIntegerTypes().contains(
                    ↪ upperType)) {
                        return INTEGER_TYPES.get(0);
                }
                if (DataTypes.getAllFloatTypes().contains(upperType
                    ↪ )) {
                        return FLOAT_TYPES.get(0);
                }
                if (DataTypes.getAllDecimalTypes().contains(
                    ↪ upperType)) {
                        return DECIMAL_TYPES.get(0);
                }
                if (DataTypes.getAllNumericTypes().contains(
                    ↪ upperType)) {
                        return ALL_NUMERIC_TYPES.get(0);
                }

                return null;
        }
}
```

Listing 4.4: Mapping of Microsoft SQL data types to common data categories

4.4.2 Support for multiple notations

In an earlier work[8], the support for multiple notations in the bigER ER model editor was added. To import SQL table definitions into these notations, the generic import process is extended by specific implementations, see Figure 4.3.

The main difference between the notations in the import process is the determination of cardinalities. Across all notations, the most important parts to make this determination, are mandatoriness and the multiplicity of the relationships, see Listing 4.5 for the cardinality determination of the Min-Max notation.

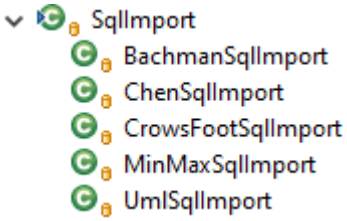

Figure 4.3: Type hierarchy of SQL importers

```
protected String getCardinality(boolean isMandatory,
    ↪ boolean isSingle, int countMultiple) {
        return (isMandatory ? "1" : "0") + "," + (isSingle
            ↪ ? "1" : "*");
}
```

Listing 4.5: Cardinality determination of the Min-Max notation

4.4.3 Optional attributes

The BIGER ER model language allows to specify, which attributes are optional. Depending on this setting, the generators adds the constraint to the produces SQL statements, see Listing 4.6. On the import side, the attribute is set optional, if the constraint is not recognized from the provided SQL statements.

```
            if (attribute.getType() != AttributeType.OPTIONAL)
                ↪ {
                    mappedType = mappedType + " " +
                        ↪ GeneratorUtils.NOT_NULL;
            }
```

Listing 4.6: Adding NOT NULL constraint

4.4.4 Multiple attributes in keys

The existing implementation of the SQL generator only supported one primary key attribute. As this was not a technical limitation, it was trivial to extend for multiple attributes, see Listing 4.7.

```
private Map<String, Attribute> primaryKey(final Map<
    ↪ String, Attribute> attributes) {
        Map<String, Attribute> key = new LinkedHashMap<>();
        for (final String name : attributes.keySet()) {
                Attribute attribute = attributes.get(name);
                if (attribute.getType() == AttributeType.KEY)
                    ↪ {
                        key.put(name, attribute);
                }
        }
        return key;
}
```

Listing 4.7: Multiple key attributes are supported

4.4.5 Numerical precision

In the existing implementation it was possible to constrain the length/scale of numerical and textual attributes. Additionally, it is now possible to define the precision for numerical values.

Only if the size and precision values greater than 0 in the ER diagram, they are generated into the SQL code as well, see Listing 4.8. In this case the responsibility of using the feature correctly lies with the user. The application does not need to have knowledge about the different data types and whether a precision value is allowed and reasonable.

```
protected String transformDataType(Attribute attribute,
    ↪ String mappedType, int size, Integer precision,
    ↪ StringBuilder comment) {
        if (size > 0) {
                String strPrecision = "";
                if (precision != null && precision > 0) {
                        strPrecision = ", " + precision;
                }
                return mappedType + "(" + size + strPrecision
                    ↪ + ")";
        }
        return mappedType;
}
```

Listing 4.8: Size and precision have to be greater than 0

4.4.6 Drop statements

While testing data models on real databases, it can be useful to cleanup the database between tests. For this case, additionally to the creation statements of the database tables, the generation of the according drop statements for the created tables (in reverse order) was implemented, see Listing 4.9.

```
private String toTable(final Entity entity, boolean drop)
    ↪ {
        StringConcatenation tableContent = new
            ↪ StringConcatenation();
        startTable(tableContent, entity.getName(), drop);
        if (!drop) {
              Set<String> usedNames = new HashSet<>();
              Map<String, Attribute> attributeMap =
                  ↪ deduplicateAttributes(entity.
                  ↪ getAttributes(), usedNames);

              addAttributes(tableContent, attributeMap);

              addPrimaryKeys(tableContent, entity.getName(),
                  ↪  Arrays.asList(this.primaryKey(
                  ↪ attributeMap)));
        }
        endTable(tableContent, drop);
        return tableContent.toString();
}
```

Listing 4.9: Depending on the use-case different SQL code is produced

4.4.7 Transitive keys

As soon as there are multiple levels of weak relationships, it can happen that a table will receive key attributes from tables not directly related. The Xtext model does not provide the model in a way, that would make it possible to resolve these indirect relationships. Therefore the needed attributes have to be cached in the generator. For each table, the full list of key attributes is stored separately, including the inherited ones. Processing another table that is in a weak relationship, will use this stored list of key attributes instead of the list from the model. The current assumption is that the referenced table will already be processed at the time it is needed, otherwise there will be an error, see Listing 4.10.

```
private Map<String, Attribute> effectivePrimaryKey(final
    ↪ Entity entity) {
        String name = entity.getName();
        if (!effectivePrimaryKeys.containsKey(name)) {
              throw new IllegalArgumentException("Entity" +
                  ↪  name + " not yet processed.");
        }
        return effectivePrimaryKeys.get(name);
}
```

Listing 4.10: Use the cached list of key attributes or raise an error

4.4.8 Attribute de-duplication

In cases where the attributes of multiple tables have to be combined, e.g. foreign keys, it has to be guaranteed, that all names are unique per table in the technical model.

To solve this problem, each attribute is checked against all attributes already processed. In case the name is already used, a number is added and increased until a unique name is found, see Listing 4.11. The new names have to be used everywhere, where the attribute is referenced.

```
public static String findUniqueName(String nameOriginal,
    ↪ Set<String> usedNames) {
        String name = nameOriginal;
        int i = 2;
        while (!usedNames.add(name)) {
              name = nameOriginal + i;
              i++;
        }
        return name;
}
```

Listing 4.11: Suffix number is increased until unique name is found

4.4.9 Weak relationship names

Weak relationships have a name in the ER diagram, but don't have a corresponding table in the technical model. When importing a technical model into an ER model, a new name has to be artificially created.

The name is composed of the names of the related tables. To ensure unique names, the same logic as in the previous section is used.

4.4.10 De-quoting

The outputs of different RDBMS have different formats, how the names of table, attributes and data types are quoted. These quotes are removed before populating the ER diagram, see Listing 4.12.

```
private String deQuote(String str) {
      str = deQuote(str, "'");
      str = deQuote(str, "\"");
      str = deQuote(str, "`");
      str = deQuote(str, "[", "]");
      return str;
}
private String deQuote(String str, String quoteChar) {
      return deQuote(str, quoteChar, null);
}

private String deQuote(String str, String quoteChar,
   ↪ String differentEndChar) {
      if (str != null && str.length() > 1) {
            if (differentEndChar == null) {
                  if (str.startsWith(quoteChar) && str.
                     ↪ endsWith(quoteChar)) {
                        str = str.substring(1, str.length
                           ↪ () - 1);
                  }
            } else {
                  if (str.startsWith(quoteChar) && str.
                     ↪ endsWith(differentEndChar)) {
                        str = str.substring(1, str.length
                           ↪ () - 1);
                  }
            }
      }
      return str;
}
```

Listing 4.12: Removing all possible quote characters

4.4.11 Data type post-processing

Some data types have spaces in their name, but this is not allowed by the BIGER DSL, e.g. character varying. The spaces are simply replaced by an underscore to make it a valid value for the ER model, see Listing 4.13 for the conversion at import and listing 4.14 for the conversion at export.

```
// ER model does not support spaces in data types
private String replaceSpaces(String value) {
      return value.replace(" ", "_");
}
```

Listing 4.13: Removing spaces from data types at import

```
// ER model does not support spaces in data types
private String replaceUnderscores(String value) {
      return value.replace("_", " ");
}
```

Listing 4.14: Removing spaces from data types at export

4.4.12 Comments

In the bigER DSL it is possible to put comments in the ER diagram, but they are not part of the model. For that reason these comments are not part of the technical model either. Other features of the generator, which have to modify the user input to generate valid code, are producing comments in the technical model to notify the user about the changes, see Listing 4.15.

On the import side, comments on attribute level can be parsed and are included in the new ER diagram. Comments on other database objects, e.g. tables, are not supported.

```
CREATE TABLE Entity1 (
      id VARCHAR(255) NOT NULL, -- added default type; added
         ↪ NULL constraint
      PRIMARY KEY (id)
);
```

Listing 4.15: Comments about changes while generating SQL code

CHAPTER 5

Conclusion

This chapter contains an overview of the observations made during development of the described features. Afterwards a short summary is given about the whole project, as well as an outlook on potential future work.

5.1 Observations

The decision to try using pure Java generators and importers instead of heavily utilizing the Xtend framework, proved useful, as all expected benefits were reached and no unexpected disadvantages were observed. The modular framework and the technologies used in BIGER enabled independent and flexible development of features in an iterative approach.

The primary goal of supporting a mapping of multiple input formats to multiple output formats in both directions of the ER modeling process could be reached, by accepting trade-offs in specific feature functionalities. Where possible, features were implemented to support the user as much as possible, while being aware and notifying about potential adaptations needed because of shortcomings in the automated mapping.

Due to the high standardization of both the ER language provided by BIGER and the accepted SQL language across the chosen vendors, almost all of the functionality could be implemented in a generic approach with minimal adaptations for specific vendor implementations. For the SQL export, the biggest differences are the column data types. The solution provided in this work covers the most prominent types and an attempt of a mapping between them. As a continued maintenance is expected to support additional vendors, future changes for specific vendors and compatibility between the formats, the architectural design focused on minimizing the effort of maintenance work and extension.

Although the output formats across the vendors are different from the common accepted language, the use of Regular Expressions proved to be a flexible form of abstraction to hide vendor specifics. Problems not efficiently covered by RegEx patterns, could be avoided by pre-processing the input. Importing SQL into an ER diagram does not have a unique possible mapping, so the produced cardinalities are estimates that potentially have to be adapted by the user.

5.2 Summary

In summary, the bigER tool was further enhanced by two important features. The import and export in multiple formats and notations is expected to enable even more users to design, maintain and migrate their models and databases with this tool. By utilizing a modular approach, not only the cross-product of notations and RDBMS vendors is supported by design, but by transitivity the cross-product of notations and other notations, and vendors and other vendors as well. Each added vendor or notation will increase this list non-linearly.

5.3 Outlook

In comparison to other available tools, bigER offers the richest feature set to support working with multiple ER notations and RDBMS vendors. The modular and extensible architecture ensures maintainability and up-to-date support for the future. Extending the list of supported formats is possible, as well as offering more features, according to the community's needs. Converting ER diagrams to non-RDBMS systems will further enhance the capabilities of bigER.

During this work, a lot of ideas were discussed and implemented, but some were discarded after analyzing the implications and advantages for the users. In this section we cover the cases, where an implementation does not benefit the users or the stability of the code generation. Some of the ideas may get re-visited in later works.

5.3.1 Case conversion

The ER model is case-insensitive in general, but the various RDBMS implementations can have a preferred casing. The same attribute could be generated into different forms, depending on the vendor.

After some discussion and weighing of the implications for the user's workflow, it was decided to not implement this conversion.

5.3.2 Sorting attributes and tables

The idea to sort the generator output, e.g. alphabetically, was quickly discarded, as it would not reflect dependencies between tables or relationships. The existing order is kept for tables, which reflects dependencies between strong and weak entities. In all other cases the given order by the user is unchanged, e.g. for attributes.

5.3.3 Default values

Another potential extension for the ER model would have been to add the possibility to define a default value for the attributes. This would have supplemented the existing possibility to define an attribute mandatory.

This feature was deemed too technical for the ER model and was not implemented. The same is true for automatically generated values for key attributes.

5.3.4 Keyword "multi-valued"

The bigER DSL already defines a "multi-valued" keyword, that is intended to model attributes, that can have multiple values, e.g. addresses, phone numbers. The SQL export would need to be enhanced to create artificial tables and relationships for the multi-valued attributes. The SQL import is probably more complicated, or would skip this feature at all.

APPENDIX A

ER diagrams

A.1 university.erd

```
// ER Model
erdiagram University

// Options
notation=default

// Entities
entity Student {
   id: INT key
   name: VARCHAR(255)
   birthday: DATE
   age: SMALLINT
}
entity Course {
   course_nr: INT key
   course_name: VARCHAR(100)
   credits: SMALLINT
}
weak entity Lecture {
   title: VARCHAR(255) partial-key
}
entity Instructor {
   instructor_id: INT key
   name: VARCHAR(255)
}
```

```
entity Department {
   dept_nr: INT key
   name: VARCHAR(100)
   abbreviation: CHAR(5)
}
weak entity Room {
   room_nr: INT partial-key
}
entity Building {
   building_id: CHAR(8) key
   address: VARCHAR(255)
}

// Relationships
relationship Exam {
   Student[1] -> Course[N] -> Instructor[N]
   points: DOUBLE
}
weak relationship has {
   Room[N] -> Building[1]
}
relationship Office {
   Room[1] -> Instructor[1]
}
relationship Work {
   Instructor[N] -> Department[1]
}
weak relationship include {
   Course[1] -> Lecture[N]
}
relationship Location {
   Building[N] -> Department[1]
}
```

A.2 relationships.erd

```
erdiagram Model

notation=default

entity Entity1 { id key }
entity Entity2 { id key }
```

```
entity Entity3 { id key }
entity Entity4 { id key }
entity Entity5 { id key }
entity Entity6 { id key }
entity Entity7 { id key }
entity Entity8 { id key }

relationship Rel1 {
   // zero-or-one -> one-and-only-one
   Entity1[0..1] -> Entity2[1..1]
}
relationship Rel2 {
   // [1..1] is the same as [1] (one)
   Entity3[1..1] -> Entity4[1]
}
relationship Rel3 {
   // zero-or-more -> one-or-more
   Entity5[0..N] -> Entity6[1..N]
}
relationship Rel4 {
   // [1..N] is the same as [N] (many)
   Entity7[1..N] -> Entity8[N]
}
```

A.3 basic.erd

```
erdiagram BasicExample // ER model

// entities
entity Customer {
   id: int key
   name: string
}
entity Order {
   order_number: int key
   price: double
}

// one-to-many relationship
relationship Places {
   Customer[1] -> Order[N]
}
```

APPENDIX B

RDBMS output

B.1 db2look.txt

```
-- Diese CLP-Datei wurde mit DB2LOOK Version "11.5" erstellt.
-- Zeitmarke: 10.12.2022 16:09:58
-- Datenbankname: BIGER
-- Datenbankmanagerversion: DB2/NT64 Version 11.5.8.0
-- Codepage der Datenbank: 1208
-- Sortierfolge fr Datenbank lautet: SYSTEM_1252
-- Alternative Sortierfolge (alt_collate): null
-- VARCHAR2-Kompatibilitt (varchar2_compat): OFF


CONNECT TO BIGER;

------------------------------------------------
-- DDL-Anweisungen fr Schemata
------------------------------------------------

-- Wenn die folgende DDL ausgefhrt wird, wird explizit ein
   ↪ Schema in der
-- neuen Datenbank erstellt, das einem implizit erstellten
   ↪ Schema
-- in der ursprnglichen Datenbank entspricht.

CREATE SCHEMA "DB2ADMIN";
```

```
------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."STUDENT"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."STUDENT" (
              "ID" BIGINT NOT NULL ,
              "NAME" VARCHAR(255 OCTETS) ,
              "BIRTHDAY" DATE ,
              "AGE" SMALLINT )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ STUDENT"

ALTER TABLE "DB2ADMIN"."STUDENT"
      ADD PRIMARY KEY
            ("ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."COURSE"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."COURSE" (
              "COURSE_NR" BIGINT NOT NULL ,
              "COURSE_NAME" VARCHAR(100 OCTETS) ,
              "CREDITS" SMALLINT )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ COURSE"

ALTER TABLE "DB2ADMIN"."COURSE"
      ADD PRIMARY KEY
```

```
            ("COURSE_NR")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."INSTRUCTOR"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."INSTRUCTOR" (
              "INSTRUCTOR_ID" BIGINT NOT NULL ,
              "NAME" VARCHAR(255 OCTETS) )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ INSTRUCTOR"

ALTER TABLE "DB2ADMIN"."INSTRUCTOR"
      ADD PRIMARY KEY
            ("INSTRUCTOR_ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."DEPARTMENT"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."DEPARTMENT" (
              "DEPT_NR" BIGINT NOT NULL ,
              "NAME" VARCHAR(100 OCTETS) ,
              "ABBREVIATION" CHAR(5 OCTETS) )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ DEPARTMENT"
```

```
ALTER TABLE "DB2ADMIN"."DEPARTMENT"
      ADD PRIMARY KEY
            ("DEPT_NR")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."BUILDING"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."BUILDING" (
             "BUILDING_ID" CHAR(8 OCTETS) NOT NULL ,
             "ADDRESS" VARCHAR(255 OCTETS) )
            IN "USERSPACE1"
            ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ BUILDING"

ALTER TABLE "DB2ADMIN"."BUILDING"
      ADD PRIMARY KEY
            ("BUILDING_ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."ROOM"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."ROOM" (
             "ROOM_NR" BIGINT NOT NULL ,
             "BUILDING_ID" CHAR(8 OCTETS) NOT NULL )
            IN "USERSPACE1"
            ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."ROOM
   ↪ "
```

```
ALTER TABLE "DB2ADMIN"."ROOM"
      ADD PRIMARY KEY
            ("ROOM_NR",
             "BUILDING_ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."LECTURE"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."LECTURE" (
             "TITLE" VARCHAR(255 OCTETS) NOT NULL ,
             "COURSE_NR" BIGINT NOT NULL )
            IN "USERSPACE1"
            ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ LECTURE"

ALTER TABLE "DB2ADMIN"."LECTURE"
      ADD PRIMARY KEY
            ("TITLE",
             "COURSE_NR")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."EXAM"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."EXAM" (
             "ID" BIGINT NOT NULL ,
             "COURSE_NR" BIGINT NOT NULL ,
             "INSTRUCTOR_ID" BIGINT NOT NULL ,
             "POINTS" DOUBLE )
            IN "USERSPACE1"
```

```
            ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."EXAM
   ↪ "

ALTER TABLE "DB2ADMIN"."EXAM"
      ADD PRIMARY KEY
            ("ID",
             "COURSE_NR",
             "INSTRUCTOR_ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."OFFICE"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."OFFICE" (
             "ROOM_NR" BIGINT NOT NULL ,
             "BUILDING_ID" CHAR(8 OCTETS) NOT NULL ,
             "INSTRUCTOR_ID" BIGINT NOT NULL )
            IN "USERSPACE1"
            ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ OFFICE"

ALTER TABLE "DB2ADMIN"."OFFICE"
      ADD PRIMARY KEY
            ("ROOM_NR",
             "BUILDING_ID",
             "INSTRUCTOR_ID")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."WORK"
------------------------------------------------
```

```
CREATE TABLE "DB2ADMIN"."WORK" (
              "INSTRUCTOR_ID" BIGINT NOT NULL ,
              "DEPT_NR" BIGINT NOT NULL )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."WORK
   ↪ "

ALTER TABLE "DB2ADMIN"."WORK"
      ADD PRIMARY KEY
            ("INSTRUCTOR_ID",
             "DEPT_NR")
      ENFORCED;



------------------------------------------------
-- DDL-Anweisungen fr Tabelle "DB2ADMIN"."LOCATION"
------------------------------------------------


CREATE TABLE "DB2ADMIN"."LOCATION" (
              "BUILDING_ID" CHAR(8 OCTETS) NOT NULL ,
              "DEPT_NR" BIGINT NOT NULL )
             IN "USERSPACE1"
             ORGANIZE BY ROW;


-- DDL-Anweisungen fr Primrschlssel fr Tabelle "DB2ADMIN"."
   ↪ LOCATION"

ALTER TABLE "DB2ADMIN"."LOCATION"
      ADD PRIMARY KEY
            ("BUILDING_ID",
             "DEPT_NR")
      ENFORCED;
```

```
-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."ROOM
   ↪ "

ALTER TABLE "DB2ADMIN"."ROOM"
      ADD CONSTRAINT "SQL221210155240620" FOREIGN KEY
            ("BUILDING_ID")
      REFERENCES "DB2ADMIN"."BUILDING"
            ("BUILDING_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."
   ↪ LECTURE"

ALTER TABLE "DB2ADMIN"."LECTURE"
      ADD CONSTRAINT "SQL221210155240660" FOREIGN KEY
            ("COURSE_NR")
      REFERENCES "DB2ADMIN"."COURSE"
            ("COURSE_NR")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."EXAM
   ↪ "

ALTER TABLE "DB2ADMIN"."EXAM"
      ADD CONSTRAINT "SQL221210155240730" FOREIGN KEY
            ("ID")
      REFERENCES "DB2ADMIN"."STUDENT"
            ("ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2ADMIN"."EXAM"
      ADD CONSTRAINT "SQL221210155240740" FOREIGN KEY
            ("COURSE_NR")
      REFERENCES "DB2ADMIN"."COURSE"
```

```
            ("COURSE_NR")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2ADMIN"."EXAM"
      ADD CONSTRAINT "SQL221210155240750" FOREIGN KEY
            ("INSTRUCTOR_ID")
      REFERENCES "DB2ADMIN"."INSTRUCTOR"
            ("INSTRUCTOR_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."
   ↪ OFFICE"

ALTER TABLE "DB2ADMIN"."OFFICE"
      ADD CONSTRAINT "SQL221210155240800" FOREIGN KEY
            ("ROOM_NR",
             "BUILDING_ID")
      REFERENCES "DB2ADMIN"."ROOM"
            ("ROOM_NR",
             "BUILDING_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2ADMIN"."OFFICE"
      ADD CONSTRAINT "SQL221210155240810" FOREIGN KEY
            ("INSTRUCTOR_ID")
      REFERENCES "DB2ADMIN"."INSTRUCTOR"
            ("INSTRUCTOR_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."WORK
   ↪ "
```

```
ALTER TABLE "DB2ADMIN"."WORK"
      ADD CONSTRAINT "SQL221210155240850" FOREIGN KEY
            ("INSTRUCTOR_ID")
      REFERENCES "DB2ADMIN"."INSTRUCTOR"
            ("INSTRUCTOR_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2ADMIN"."WORK"
      ADD CONSTRAINT "SQL221210155240860" FOREIGN KEY
            ("DEPT_NR")
      REFERENCES "DB2ADMIN"."DEPARTMENT"
            ("DEPT_NR")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

-- DDL-Anweisungen fr Fremdschlssel fr Tabelle "DB2ADMIN"."
   ↪ LOCATION"

ALTER TABLE "DB2ADMIN"."LOCATION"
      ADD CONSTRAINT "SQL221210155240900" FOREIGN KEY
            ("BUILDING_ID")
      REFERENCES "DB2ADMIN"."BUILDING"
            ("BUILDING_ID")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;

ALTER TABLE "DB2ADMIN"."LOCATION"
      ADD CONSTRAINT "SQL221210155240910" FOREIGN KEY
            ("DEPT_NR")
      REFERENCES "DB2ADMIN"."DEPARTMENT"
            ("DEPT_NR")
      ON DELETE CASCADE
      ON UPDATE NO ACTION
      ENFORCED
      ENABLE QUERY OPTIMIZATION;
```

```
COMMIT WORK;

CONNECT RESET;

TERMINATE;
```

B.2 mssql.txt

```
USE [master]
GO
/****** Object: User [##MS_PolicyEventProcessingLogin##] Script
   ↪  Date: 08.12.2022 17:01:25 ******/
CREATE USER [##MS_PolicyEventProcessingLogin##] FOR LOGIN [##
   ↪ MS_PolicyEventProcessingLogin##] WITH DEFAULT_SCHEMA=[dbo]
GO
/****** Object: User [##MS_AgentSigningCertificate##] Script
   ↪ Date: 08.12.2022 17:01:25 ******/
CREATE USER [##MS_AgentSigningCertificate##] FOR LOGIN [##
   ↪ MS_AgentSigningCertificate##]
GO
/****** Object: Table [dbo].[Building] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Building](
      [building_id] [char](8) NOT NULL,
      [address] [varchar](255) NULL,
PRIMARY KEY CLUSTERED
(
      [building_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
```

```
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Course] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Course](
      [course_nr] [int] NOT NULL,
      [course_name] [varchar](100) NULL,
      [credits] [smallint] NULL,
PRIMARY KEY CLUSTERED
(
      [course_nr] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Department] Script Date:
   ↪ 08.12.2022 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Department](
      [dept_nr] [int] NOT NULL,
      [name] [varchar](100) NULL,
      [abbreviation] [char](5) NULL,
PRIMARY KEY CLUSTERED
(
      [dept_nr] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
```

```
/****** Object: Table [dbo].[Exam] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Exam](
      [id] [int] NOT NULL,
      [course_nr] [int] NOT NULL,
      [instructor_id] [int] NOT NULL,
      [points] [decimal](18, 0) NULL,
PRIMARY KEY CLUSTERED
(
      [id] ASC,
      [course_nr] ASC,
      [instructor_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Instructor] Script Date:
   ↪ 08.12.2022 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Instructor](
      [instructor_id] [int] NOT NULL,
      [name] [varchar](255) NULL,
PRIMARY KEY CLUSTERED
(
      [instructor_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Lecture] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
```

```
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Lecture](
      [title] [varchar](255) NOT NULL,
      [course_nr] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [title] ASC,
      [course_nr] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Location] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Location](
      [building_id] [char](8) NOT NULL,
      [dept_nr] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [building_id] ASC,
      [dept_nr] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Office] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Office](
      [room_nr] [int] NOT NULL,
```

```
      [building_id] [char](8) NOT NULL,
      [instructor_id] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [room_nr] ASC,
      [building_id] ASC,
      [instructor_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Room] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Room](
      [room_nr] [int] NOT NULL,
      [building_id] [char](8) NOT NULL,
PRIMARY KEY CLUSTERED
(
      [room_nr] ASC,
      [building_id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Student] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Student](
      [id] [int] NOT NULL,
      [name] [varchar](255) NULL,
      [birthday] [date] NULL,
      [age] [smallint] NULL,
```

```
PRIMARY KEY CLUSTERED
(
      [id] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
/****** Object: Table [dbo].[Work] Script Date: 08.12.2022
   ↪ 17:01:25 ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TABLE [dbo].[Work](
      [instructor_id] [int] NOT NULL,
      [dept_nr] [int] NOT NULL,
PRIMARY KEY CLUSTERED
(
      [instructor_id] ASC,
      [dept_nr] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF,
   ↪ IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
   ↪ ALLOW_PAGE_LOCKS = ON, OPTIMIZE_FOR_SEQUENTIAL_KEY = OFF)
   ↪ ON [PRIMARY]
) ON [PRIMARY]
GO
ALTER TABLE [dbo].[Exam] WITH CHECK ADD FOREIGN KEY([course_nr
   ↪ ])
REFERENCES [dbo].[Course] ([course_nr])
GO
ALTER TABLE [dbo].[Exam] WITH CHECK ADD FOREIGN KEY([id])
REFERENCES [dbo].[Student] ([id])
GO
ALTER TABLE [dbo].[Exam] WITH CHECK ADD FOREIGN KEY([
   ↪ instructor_id])
REFERENCES [dbo].[Instructor] ([instructor_id])
GO
ALTER TABLE [dbo].[Lecture] WITH CHECK ADD FOREIGN KEY([
   ↪ course_nr])
REFERENCES [dbo].[Course] ([course_nr])
ON DELETE CASCADE
```

```
GO
ALTER TABLE [dbo].[Location] WITH CHECK ADD FOREIGN KEY([
   ↪ building_id])
REFERENCES [dbo].[Building] ([building_id])
GO
ALTER TABLE [dbo].[Location] WITH CHECK ADD FOREIGN KEY([
   ↪ dept_nr])
REFERENCES [dbo].[Department] ([dept_nr])
GO
ALTER TABLE [dbo].[Office] WITH CHECK ADD FOREIGN KEY([room_nr],
   ↪  [building_id])
REFERENCES [dbo].[Room] ([room_nr], [building_id])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Office] WITH CHECK ADD FOREIGN KEY([
   ↪ instructor_id])
REFERENCES [dbo].[Instructor] ([instructor_id])
GO
ALTER TABLE [dbo].[Room] WITH CHECK ADD FOREIGN KEY([
   ↪ building_id])
REFERENCES [dbo].[Building] ([building_id])
ON DELETE CASCADE
GO
ALTER TABLE [dbo].[Work] WITH CHECK ADD FOREIGN KEY([dept_nr])
REFERENCES [dbo].[Department] ([dept_nr])
GO
ALTER TABLE [dbo].[Work] WITH CHECK ADD FOREIGN KEY([
   ↪ instructor_id])
REFERENCES [dbo].[Instructor] ([instructor_id])
GO
```

B.3 mysql.txt

```
CREATE TABLE `Building` (
  `building_id` char(8) NOT NULL,
  `address` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`building_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Department` (
  `dept_nr` int(11) NOT NULL,
  `name` varchar(100) DEFAULT NULL,
```

```
  `abbreviation` char(5) DEFAULT NULL,
  PRIMARY KEY (`dept_nr`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Course` (
  `course_nr` int(11) NOT NULL,
  `course_name` varchar(100) DEFAULT NULL,
  `credits` smallint(6) DEFAULT NULL,
  PRIMARY KEY (`course_nr`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Exam` (
  `id` int(11) NOT NULL,
  `course_nr` int(11) NOT NULL,
  `instructor_id` int(11) NOT NULL,
  `points` double DEFAULT NULL,
  PRIMARY KEY (`id`,`course_nr`,`instructor_id`),
  CONSTRAINT `Exam_ibfk_1` FOREIGN KEY (`id`) REFERENCES `
    ↪ Student` (`id`) ON DELETE CASCADE,
  CONSTRAINT `Exam_ibfk_2` FOREIGN KEY (`course_nr`) REFERENCES
    ↪ `Course` (`course_nr`) ON DELETE CASCADE,
  CONSTRAINT `Exam_ibfk_3` FOREIGN KEY (`instructor_id`)
    ↪ REFERENCES `Instructor` (`instructor_id`) ON DELETE
    ↪ CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Lecture` (
  `title` varchar(255) NOT NULL,
  `course_nr` int(11) NOT NULL,
  PRIMARY KEY (`title`,`course_nr`),
  CONSTRAINT `Lecture_ibfk_1` FOREIGN KEY (`course_nr`)
    ↪ REFERENCES `Course` (`course_nr`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Instructor` (
  `instructor_id` int(11) NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  PRIMARY KEY (`instructor_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Location` (
  `building_id` char(8) NOT NULL,
  `dept_nr` int(11) NOT NULL,
```

```
  PRIMARY KEY (`building_id`,`dept_nr`),
  CONSTRAINT `Location_ibfk_1` FOREIGN KEY (`building_id`)
     ↪ REFERENCES `Building` (`building_id`) ON DELETE CASCADE,
  CONSTRAINT `Location_ibfk_2` FOREIGN KEY (`dept_nr`)
     ↪ REFERENCES `Department` (`dept_nr`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Office` (
  `room_nr` int(11) NOT NULL,
  `building_id` char(8) NOT NULL,
  `instructor_id` int(11) NOT NULL,
  PRIMARY KEY (`room_nr`,`building_id`,`instructor_id`),
  CONSTRAINT `Office_ibfk_1` FOREIGN KEY (`room_nr`, `
     ↪ building_id`) REFERENCES `Room` (`room_nr`, `building_id
     ↪ `) ON DELETE CASCADE,
  CONSTRAINT `Office_ibfk_2` FOREIGN KEY (`instructor_id`)
     ↪ REFERENCES `Instructor` (`instructor_id`) ON DELETE
     ↪ CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Student` (
  `id` int(11) NOT NULL,
  `name` varchar(255) DEFAULT NULL,
  `birthday` date DEFAULT NULL,
  `age` smallint(6) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Room` (
  `room_nr` int(11) NOT NULL,
  `building_id` char(8) NOT NULL,
  PRIMARY KEY (`room_nr`,`building_id`),
  CONSTRAINT `Room_ibfk_1` FOREIGN KEY (`building_id`)
     ↪ REFERENCES `Building` (`building_id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;

CREATE TABLE `Work` (
  `instructor_id` int(11) NOT NULL,
  `dept_nr` int(11) NOT NULL,
  PRIMARY KEY (`instructor_id`,`dept_nr`),
  CONSTRAINT `Work_ibfk_1` FOREIGN KEY (`instructor_id`)
     ↪ REFERENCES `Instructor` (`instructor_id`) ON DELETE
     ↪ CASCADE,
```

```
  CONSTRAINT `Work_ibfk_2` FOREIGN KEY (`dept_nr`) REFERENCES `
     ↪ Department` (`dept_nr`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

B.4 oracle.txt

```
--------------------------------------------------------
-- DDL for Table BUILDING
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."BUILDING"
   ( "BUILDING_ID" CHAR(8 BYTE),
       "ADDRESS" VARCHAR2(255 BYTE)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table COURSE
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."COURSE"
   ( "COURSE_NR" NUMBER(*,0),
       "COURSE_NAME" VARCHAR2(100 BYTE),
       "CREDITS" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table DEPARTMENT
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."DEPARTMENT"
```

```
  ( "DEPT_NR" NUMBER(*,0),
      "NAME" VARCHAR2(100 BYTE),
      "ABBREVIATION" CHAR(5 BYTE)
  ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    ↪ 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
    ↪ DEFAULT)
 TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table EXAM
--------------------------------------------------------

 CREATE TABLE "SYSTEM"."EXAM"
  ( "ID" NUMBER(*,0),
      "COURSE_NR" NUMBER(*,0),
      "INSTRUCTOR_ID" NUMBER(*,0),
      "POINTS" BINARY_DOUBLE
  ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    ↪ 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
    ↪ DEFAULT)
 TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table INSTRUCTOR
--------------------------------------------------------

 CREATE TABLE "SYSTEM"."INSTRUCTOR"
  ( "INSTRUCTOR_ID" NUMBER(*,0),
      "NAME" VARCHAR2(255 BYTE)
  ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
 STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
    ↪ 2147483645
 PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
 BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
    ↪ DEFAULT)
 TABLESPACE "SYSTEM" ;
```

```
--------------------------------------------------------
-- DDL for Table LECTURE
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."LECTURE"
   ( "TITLE" VARCHAR2(255 BYTE),
       "COURSE_NR" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table LOCATION
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."LOCATION"
   ( "BUILDING_ID" CHAR(8 BYTE),
       "DEPT_NR" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table OFFICE
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."OFFICE"
   ( "ROOM_NR" NUMBER(*,0),
       "BUILDING_ID" CHAR(8 BYTE),
       "INSTRUCTOR_ID" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
```

```
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table ROOM
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."ROOM"
   ( "ROOM_NR" NUMBER(*,0),
       "BUILDING_ID" CHAR(8 BYTE)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table STUDENT
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."STUDENT"
   ( "ID" NUMBER(*,0),
       "NAME" VARCHAR2(255 BYTE),
       "BIRTHDAY" DATE,
       "AGE" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Table WORK
--------------------------------------------------------

  CREATE TABLE "SYSTEM"."WORK"
   ( "INSTRUCTOR_ID" NUMBER(*,0),
       "DEPT_NR" NUMBER(*,0)
   ) PCTFREE 10 PCTUSED 40 INITRANS 1 MAXTRANS 255
```

```
 NOCOMPRESS LOGGING
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
-----------------------------------------------------
-- DDL for Index SYS_C008406
-----------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008406" ON "SYSTEM"."
     ↪ BUILDING" ("BUILDING_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
-----------------------------------------------------
-- DDL for Index SYS_C008403
-----------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008403" ON "SYSTEM"."COURSE
     ↪ " ("COURSE_NR")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
-----------------------------------------------------
-- DDL for Index SYS_C008405
-----------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008405" ON "SYSTEM"."
     ↪ DEPARTMENT" ("DEPT_NR")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
```

```
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008411
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008411" ON "SYSTEM"."EXAM"
     ↪ ("ID", "COURSE_NR", "INSTRUCTOR_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008404
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008404" ON "SYSTEM"."
     ↪ INSTRUCTOR" ("INSTRUCTOR_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008409
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008409" ON "SYSTEM"."
     ↪ LECTURE" ("TITLE", "COURSE_NR")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
```

```
-- DDL for Index SYS_C008421
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008421" ON "SYSTEM"."
     ↪ LOCATION" ("BUILDING_ID", "DEPT_NR")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008415
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008415" ON "SYSTEM"."OFFICE
     ↪ " ("ROOM_NR", "BUILDING_ID", "INSTRUCTOR_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008407
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008407" ON "SYSTEM"."ROOM"
     ↪ ("ROOM_NR", "BUILDING_ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008402
--------------------------------------------------------
```

```
  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008402" ON "SYSTEM"."
     ↪ STUDENT" ("ID")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- DDL for Index SYS_C008418
--------------------------------------------------------

  CREATE UNIQUE INDEX "SYSTEM"."SYS_C008418" ON "SYSTEM"."WORK"
     ↪ ("INSTRUCTOR_ID", "DEPT_NR")
  PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ;
--------------------------------------------------------
-- Constraints for Table BUILDING
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."BUILDING" ADD PRIMARY KEY ("BUILDING_ID
     ↪ ")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table COURSE
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."COURSE" ADD PRIMARY KEY ("COURSE_NR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
```

```
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table DEPARTMENT
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."DEPARTMENT" ADD PRIMARY KEY ("DEPT_NR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table EXAM
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."EXAM" ADD PRIMARY KEY ("ID", "COURSE_NR
     ↪ ", "INSTRUCTOR_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table INSTRUCTOR
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."INSTRUCTOR" ADD PRIMARY KEY ("
     ↪ INSTRUCTOR_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
```

```
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table LECTURE
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."LECTURE" ADD PRIMARY KEY ("TITLE", "
     ↪ COURSE_NR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table LOCATION
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."LOCATION" ADD PRIMARY KEY ("BUILDING_ID
     ↪ ", "DEPT_NR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table OFFICE
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."OFFICE" ADD PRIMARY KEY ("ROOM_NR", "
     ↪ BUILDING_ID", "INSTRUCTOR_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
```

```
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table ROOM
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."ROOM" ADD PRIMARY KEY ("ROOM_NR", "
     ↪ BUILDING_ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table STUDENT
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."STUDENT" ADD PRIMARY KEY ("ID")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Constraints for Table WORK
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."WORK" ADD PRIMARY KEY ("INSTRUCTOR_ID",
     ↪ "DEPT_NR")
  USING INDEX PCTFREE 10 INITRANS 2 MAXTRANS 255 COMPUTE
     ↪ STATISTICS
  STORAGE(INITIAL 65536 NEXT 1048576 MINEXTENTS 1 MAXEXTENTS
     ↪ 2147483645
  PCTINCREASE 0 FREELISTS 1 FREELIST GROUPS 1
  BUFFER_POOL DEFAULT FLASH_CACHE DEFAULT CELL_FLASH_CACHE
     ↪ DEFAULT)
```

```
  TABLESPACE "SYSTEM" ENABLE;
--------------------------------------------------------
-- Ref Constraints for Table EXAM
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."EXAM" ADD FOREIGN KEY ("ID")
        REFERENCES "SYSTEM"."STUDENT" ("ID") ON DELETE CASCADE
           ↪ ENABLE;
  ALTER TABLE "SYSTEM"."EXAM" ADD FOREIGN KEY ("COURSE_NR")
        REFERENCES "SYSTEM"."COURSE" ("COURSE_NR") ON DELETE
           ↪ CASCADE ENABLE;
  ALTER TABLE "SYSTEM"."EXAM" ADD FOREIGN KEY ("INSTRUCTOR_ID")
        REFERENCES "SYSTEM"."INSTRUCTOR" ("INSTRUCTOR_ID") ON
           ↪ DELETE CASCADE ENABLE;
--------------------------------------------------------
-- Ref Constraints for Table LECTURE
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."LECTURE" ADD FOREIGN KEY ("COURSE_NR")
        REFERENCES "SYSTEM"."COURSE" ("COURSE_NR") ON DELETE
           ↪ CASCADE ENABLE;
--------------------------------------------------------
-- Ref Constraints for Table LOCATION
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."LOCATION" ADD FOREIGN KEY ("BUILDING_ID
     ↪ ")
        REFERENCES "SYSTEM"."BUILDING" ("BUILDING_ID") ON DELETE
           ↪  CASCADE ENABLE;
  ALTER TABLE "SYSTEM"."LOCATION" ADD FOREIGN KEY ("DEPT_NR")
        REFERENCES "SYSTEM"."DEPARTMENT" ("DEPT_NR") ON DELETE
           ↪ CASCADE ENABLE;
--------------------------------------------------------
-- Ref Constraints for Table OFFICE
--------------------------------------------------------

  ALTER TABLE "SYSTEM"."OFFICE" ADD FOREIGN KEY ("ROOM_NR", "
     ↪ BUILDING_ID")
        REFERENCES "SYSTEM"."ROOM" ("ROOM_NR", "BUILDING_ID") ON
           ↪  DELETE CASCADE ENABLE;
  ALTER TABLE "SYSTEM"."OFFICE" ADD FOREIGN KEY ("INSTRUCTOR_ID
     ↪ ")
```

```
        REFERENCES "SYSTEM"."INSTRUCTOR" ("INSTRUCTOR_ID") ON
           ↪ DELETE CASCADE ENABLE;
-----------------------------------------------------------
-- Ref Constraints for Table ROOM
-----------------------------------------------------------

  ALTER TABLE "SYSTEM"."ROOM" ADD FOREIGN KEY ("BUILDING_ID")
        REFERENCES "SYSTEM"."BUILDING" ("BUILDING_ID") ON DELETE
           ↪  CASCADE ENABLE;
-----------------------------------------------------------
-- Ref Constraints for Table WORK
-----------------------------------------------------------

  ALTER TABLE "SYSTEM"."WORK" ADD FOREIGN KEY ("INSTRUCTOR_ID")
        REFERENCES "SYSTEM"."INSTRUCTOR" ("INSTRUCTOR_ID") ON
           ↪ DELETE CASCADE ENABLE;
  ALTER TABLE "SYSTEM"."WORK" ADD FOREIGN KEY ("DEPT_NR")
        REFERENCES "SYSTEM"."DEPARTMENT" ("DEPT_NR") ON DELETE
           ↪ CASCADE ENABLE;
```

B.5 pgadmin.txt

```
-- Table: public.exam

-- DROP TABLE IF EXISTS public.exam;

CREATE TABLE IF NOT EXISTS public.exam
(
    id integer NOT NULL,
    course_nr integer NOT NULL,
    instructor_id integer NOT NULL,
    points double precision,
    CONSTRAINT exam_pkey PRIMARY KEY (id, course_nr,
       ↪ instructor_id),
    CONSTRAINT exam_course_nr_fkey FOREIGN KEY (course_nr)
       REFERENCES public.course (course_nr) MATCH SIMPLE
       ON UPDATE NO ACTION
       ON DELETE NO ACTION,
    CONSTRAINT exam_id_fkey FOREIGN KEY (id)
       REFERENCES public.student (id) MATCH SIMPLE
       ON UPDATE NO ACTION
       ON DELETE NO ACTION,
```

```
    CONSTRAINT exam_instructor_id_fkey FOREIGN KEY (
        ↪ instructor_id)
        REFERENCES public.instructor (instructor_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.exam
    OWNER to postgres;

-- Table: public.room

-- DROP TABLE IF EXISTS public.room;

CREATE TABLE IF NOT EXISTS public.room
(
    room_nr integer NOT NULL,
    building_id character(8) COLLATE pg_catalog."default" NOT
        ↪ NULL,
    CONSTRAINT room_pkey PRIMARY KEY (room_nr, building_id),
    CONSTRAINT room_building_id_fkey FOREIGN KEY (building_id)
        REFERENCES public.building (building_id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE CASCADE
)

TABLESPACE pg_default;

ALTER TABLE IF EXISTS public.room
    OWNER to postgres;
```

B.6 pgdump.txt

```
--
-- PostgreSQL database dump
--

-- Dumped from database version 15.1
-- Dumped by pg_dump version 15.1
```

```
SET statement_timeout = 0;
SET lock_timeout = 0;
SET idle_in_transaction_session_timeout = 0;
SET client_encoding = 'UTF8';
SET standard_conforming_strings = on;
SELECT pg_catalog.set_config('search_path', '', false);
SET check_function_bodies = false;
SET xmloption = content;
SET client_min_messages = warning;
SET row_security = off;

--
-- Name: adminpack; Type: EXTENSION; Schema: -; Owner: -
--

CREATE EXTENSION IF NOT EXISTS adminpack WITH SCHEMA pg_catalog
    ↪ ;


--
-- Name: EXTENSION adminpack; Type: COMMENT; Schema: -; Owner:
--

COMMENT ON EXTENSION adminpack IS 'administrative functions for
    ↪  PostgreSQL';


SET default_tablespace = '';

SET default_table_access_method = heap;

--
-- Name: building; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.building (
    building_id character(8) NOT NULL,
    address character varying(255)
);


ALTER TABLE public.building OWNER TO postgres;
```

```
--
-- Name: course; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.course (
   course_nr integer NOT NULL,
   course_name character varying(100),
   credits smallint
);


ALTER TABLE public.course OWNER TO postgres;

--
-- Name: department; Type: TABLE; Schema: public; Owner:
   ↪ postgres
--

CREATE TABLE public.department (
   dept_nr integer NOT NULL,
   name character varying(100),
   abbreviation character(5)
);


ALTER TABLE public.department OWNER TO postgres;

--
-- Name: exam; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.exam (
   id integer NOT NULL,
   course_nr integer NOT NULL,
   instructor_id integer NOT NULL,
   points double precision
);


ALTER TABLE public.exam OWNER TO postgres;

--
```

```
-- Name: instructor; Type: TABLE; Schema: public; Owner:
   ↪ postgres
--

CREATE TABLE public.instructor (
   instructor_id integer NOT NULL,
   name character varying(255)
);


ALTER TABLE public.instructor OWNER TO postgres;

--
-- Name: lecture; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.lecture (
   title character varying(255) NOT NULL,
   course_nr integer NOT NULL
);


ALTER TABLE public.lecture OWNER TO postgres;

--
-- Name: location; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.location (
   building_id character(8) NOT NULL,
   dept_nr integer NOT NULL
);


ALTER TABLE public.location OWNER TO postgres;

--
-- Name: office; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.office (
   building_id character(8) NOT NULL,
   room_nr integer NOT NULL,
```

```
   instructor_id integer NOT NULL
);


ALTER TABLE public.office OWNER TO postgres;

--
-- Name: room; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.room (
   room_nr integer NOT NULL,
   building_id character(8) NOT NULL
);


ALTER TABLE public.room OWNER TO postgres;

--
-- Name: student; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.student (
   id integer NOT NULL,
   name character varying(255),
   birthday date,
   age smallint
);


ALTER TABLE public.student OWNER TO postgres;

--
-- Name: work; Type: TABLE; Schema: public; Owner: postgres
--

CREATE TABLE public.work (
   instructor_id integer NOT NULL,
   dept_nr integer NOT NULL
);


ALTER TABLE public.work OWNER TO postgres;
```

```
--
-- Name: building building_pkey; Type: CONSTRAINT; Schema:
   ↪ public; Owner: postgres
--

ALTER TABLE ONLY public.building
   ADD CONSTRAINT building_pkey PRIMARY KEY (building_id);


--
-- Name: course course_pkey; Type: CONSTRAINT; Schema: public;
   ↪ Owner: postgres
--

ALTER TABLE ONLY public.course
   ADD CONSTRAINT course_pkey PRIMARY KEY (course_nr);


--
-- Name: department department_pkey; Type: CONSTRAINT; Schema:
   ↪ public; Owner: postgres
--

ALTER TABLE ONLY public.department
   ADD CONSTRAINT department_pkey PRIMARY KEY (dept_nr);


--
-- Name: exam exam_pkey; Type: CONSTRAINT; Schema: public;
   ↪ Owner: postgres
--

ALTER TABLE ONLY public.exam
   ADD CONSTRAINT exam_pkey PRIMARY KEY (id, course_nr,
      ↪ instructor_id);


--
-- Name: instructor instructor_pkey; Type: CONSTRAINT; Schema:
   ↪ public; Owner: postgres
--
```

```
ALTER TABLE ONLY public.instructor
   ADD CONSTRAINT instructor_pkey PRIMARY KEY (instructor_id);


--
-- Name: lecture lecture_pkey; Type: CONSTRAINT; Schema: public
   ↪ ; Owner: postgres
--

ALTER TABLE ONLY public.lecture
   ADD CONSTRAINT lecture_pkey PRIMARY KEY (title, course_nr);


--
-- Name: location location_pkey; Type: CONSTRAINT; Schema:
   ↪ public; Owner: postgres
--

ALTER TABLE ONLY public.location
   ADD CONSTRAINT location_pkey PRIMARY KEY (building_id,
      ↪ dept_nr);


--
-- Name: office office_pkey; Type: CONSTRAINT; Schema: public;
   ↪ Owner: postgres
--

ALTER TABLE ONLY public.office
   ADD CONSTRAINT office_pkey PRIMARY KEY (room_nr,
      ↪ instructor_id, building_id);


--
-- Name: room room_pkey; Type: CONSTRAINT; Schema: public;
   ↪ Owner: postgres
--

ALTER TABLE ONLY public.room
   ADD CONSTRAINT room_pkey PRIMARY KEY (room_nr, building_id);


--
```

```
-- Name: student student_pkey; Type: CONSTRAINT; Schema: public
   ↪ ; Owner: postgres
--

ALTER TABLE ONLY public.student
   ADD CONSTRAINT student_pkey PRIMARY KEY (id);


--
-- Name: work work_pkey; Type: CONSTRAINT; Schema: public;
   ↪ Owner: postgres
--

ALTER TABLE ONLY public.work
   ADD CONSTRAINT work_pkey PRIMARY KEY (instructor_id, dept_nr
      ↪ );


--
-- Name: exam exam_course_nr_fkey; Type: FK CONSTRAINT; Schema:
   ↪  public; Owner: postgres
--

ALTER TABLE ONLY public.exam
   ADD CONSTRAINT exam_course_nr_fkey FOREIGN KEY (course_nr)
      ↪ REFERENCES public.course(course_nr);


--
-- Name: exam exam_id_fkey; Type: FK CONSTRAINT; Schema: public
   ↪ ; Owner: postgres
--

ALTER TABLE ONLY public.exam
   ADD CONSTRAINT exam_id_fkey FOREIGN KEY (id) REFERENCES
      ↪ public.student(id);


--
-- Name: exam exam_instructor_id_fkey; Type: FK CONSTRAINT;
   ↪ Schema: public; Owner: postgres
--
```

```
ALTER TABLE ONLY public.exam
   ADD CONSTRAINT exam_instructor_id_fkey FOREIGN KEY (
      ↪ instructor_id) REFERENCES public.instructor(
      ↪ instructor_id);


--
-- Name: lecture lecture_course_nr_fkey; Type: FK CONSTRAINT;
   ↪ Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.lecture
   ADD CONSTRAINT lecture_course_nr_fkey FOREIGN KEY (course_nr
      ↪ ) REFERENCES public.course(course_nr) ON DELETE CASCADE
      ↪ ;


--
-- Name: location location_building_id_fkey; Type: FK
   ↪ CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.location
   ADD CONSTRAINT location_building_id_fkey FOREIGN KEY (
      ↪ building_id) REFERENCES public.building(building_id);


--
-- Name: location location_dept_nr_fkey; Type: FK CONSTRAINT;
   ↪ Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.location
   ADD CONSTRAINT location_dept_nr_fkey FOREIGN KEY (dept_nr)
      ↪ REFERENCES public.department(dept_nr);


--
-- Name: office office_building_id_room_nr_fkey; Type: FK
   ↪ CONSTRAINT; Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.office
```

```
   ADD CONSTRAINT office_building_id_room_nr_fkey FOREIGN KEY (
      ↪ building_id, room_nr) REFERENCES public.room(
      ↪ building_id, room_nr) ON DELETE CASCADE;


--
-- Name: office office_instructor_id_fkey; Type: FK CONSTRAINT;
   ↪  Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.office
   ADD CONSTRAINT office_instructor_id_fkey FOREIGN KEY (
      ↪ instructor_id) REFERENCES public.instructor(
      ↪ instructor_id);


--
-- Name: room room_building_id_fkey; Type: FK CONSTRAINT;
   ↪ Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.room
   ADD CONSTRAINT room_building_id_fkey FOREIGN KEY (
      ↪ building_id) REFERENCES public.building(building_id) ON
      ↪  DELETE CASCADE;


--
-- Name: work work_dept_nr_fkey; Type: FK CONSTRAINT; Schema:
   ↪ public; Owner: postgres
--

ALTER TABLE ONLY public.work
   ADD CONSTRAINT work_dept_nr_fkey FOREIGN KEY (dept_nr)
      ↪ REFERENCES public.department(dept_nr);


--
-- Name: work work_instructor_id_fkey; Type: FK CONSTRAINT;
   ↪ Schema: public; Owner: postgres
--

ALTER TABLE ONLY public.work
```

```
   ADD CONSTRAINT work_instructor_id_fkey FOREIGN KEY (
      ↪ instructor_id) REFERENCES public.instructor(
      ↪ instructor_id);


--
-- PostgreSQL database dump complete
--
```

List of Figures

List of Tables

Bibliography

[1] Lorenzo Bettini. *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[2] Dominik Bork and Philip Langer. Catchword: Language server protocol - an introduction to the protocol, its use, and adoption for web modeling tools. *Enterprise Modelling and Information Systems Architectures - International Journal of Conceptual Modeling*, 18(9):1–16, 2023.

[3] Peter Pin-Shan Chen. The entity-relationship model—toward a unified view of data. *ACM transactions on database systems (TODS)*, 1(1):9–36, 1976.

[4] Andreas Gadatsch. *Einführung in die Datenmodellierung*, pages 1–7. Springer Fachmedien Wiesbaden, Wiesbaden, 2017.

[5] Philipp-Lorenz Glaser. Developing sprotty-based modeling tools for vs code, 2022.

[6] Philipp-Lorenz Glaser and Dominik Bork. The biger tool - hybrid textual and graphical modeling of entity relationships in VS code. In *25th International Enterprise Distributed Object Computing Workshop, EDOC Workshop 2021, Gold Coast, Australia, October 25-29, 2021*, pages 337–340. IEEE, 2021.

[7] Philipp-Lorenz Glaser, Georg Hammerschmied, Vladyslav Hnatiuk, and Dominik Bork. The biger modeling tool. In Sebastian Link, Iris Reinhartz-Berger, Jelena Zdravkovic, Dominik Bork, and Srinath Srinivasa, editors, *Proceedings of the ER Forum and PhD Symposium 2022 co-located with 41st International Conference on Conceptual Modeling (ER 2022), Virtual Event, Hyderabad, India, October 17, 2022*, volume 3211 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2022.

[8] Georg Hammerschmied. *Multi-Notation Support for a Hybrid VS Code Modeling Tool*. PhD thesis, Technische Universität Wien, 2022.

[9] Eric Laquer. *Defining Data-Driven Software Development*. O'Reilly Media, Incorporated, 2017.

[10] Jan Paredaens, Paul De Bra, Marc Gyssens, and Dirk Van Gucht. *The structure of the relational database model*, volume 17. Springer Science & Business Media, 2012.

[11] I Puja, Patrizia Poscic, and Danijela Jaksic. Overview and comparison of several elational database modelling metodologies and notations. In *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1641–1646. IEEE, 2019.

[12] Eberhard Stickel. *Das Entity-Relationship-Modell*, pages 74–98. Gabler Verlag, Wiesbaden, 1991.