Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of System Engineering and Informatics

Bachelor Thesis

Data integrity in relational database processing

Tran Thu Phuong

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Thu Phuong Tran

Informatics

Thesis title

Data integrity in relational database processing

Objectives of thesis

This bachelor thesis is focused on the issue of data integrity in relational database processing. Its aim is to define the theoretical principles of database, relational databases, especially in data integrity, to map the current situation of this problem, to design, to check and to demonstrate possible usage of the integrity constraints. The work uses SQL language, especialy MySQL to make a project in practical part to demonstrate the necessity of using integrity constraints in database processing.

Methodology

First use the general understanding of “Integrity data in relational database processing” in the literature review of this study by referring to the developer’s guide. Then perform the study and analysis of available information sources and existing solutions in the given field. Follow the book sources written below.

About exploring “Integrity data in relational database processing”, be focused on specific features of the MySQL database management system. In the practical part, create a project using the same MySQL environment.

The proposed extent of the thesis

30 – 60 pages

Keywords

Database, Data integrity, Relational database, SQL

Recommended information sources

Database System Concepts-Sixth edition – 2012 (Abraham Silberschatz).
Fundamentals of database systems – Sixth edition-2011 (Ramez Elmasri, Shamkant B. Navathe).
Learning SQL – Second edition – 2009 (Alan Beaulieu).
Sam's Teach Yourself MySQL in 21 days – 2001 (Mark Maslakowski)

Expected date of thesis defence

2018/19 SS – FEM

The Bachelor Thesis Supervisor

doc. Ing. Vojtěch Merunka, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 31. 1. 2019

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 31. 1. 2019

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 10. 03. 2019

Declaration

I declare that I have worked on my bachelor thesis titled "Data integrity in relational database processing" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 15.03.2019 _____

Acknowledgement

I would like to thank my parents who always stand beside me, support and give me the power to pursue my dream.

I also would like to thank my supervisor Merunka Vojtěch doc.Ing.,Ph.D who gave me great help and good advice throughout the course of this thesis.

Data integrity in relational database processing

Abstract

Data integrity is a topical, necessary and comprehensive issue and is being used for years in applications, databases, and projects.

This bachelor thesis is focused on the issue of data integrity in relational database processing. Its aim is to define the theoretical principles of database, relational database, especially in data integrity, to map the current situation of this problem, to design, to check and to demonstrate possible usage of the integrity constraints. The work uses SQL, especially MySQL to make a project in practical part to demonstrate the necessity of using integrity constraints in database processing on the example of university dormitories. The solution is using relational database, SQL queries, functions and it satisfies the requirements for the evidence of dormitory building, employees, rooms, students, bookings and payments. The solution demonstrates necessity of using integrity constraints in relational database to achieve the data accuracy and consistency.

Keywords: database, data integrity, relational database, SQL

Integrita dat ve relačne databázovem zpracování

Abstrakt

Bakalářská práce je zaměřena na problematiku integrity dat při zpracování relačních databází. Jeho cílem je definovat teoretické principy databáze, relační databáze, zejména v oblasti integrity dat, zmapovat současnou situaci tohoto problému, navrhnout, zkontrolovat a prokázat možné využití omezení integrity. Práce využívá SQL, zejména MySQL, k vytvoření projektu v praktické části, který demonstruje nutnost použití integračních omezení při zpracování databází na příkladu univerzitních kolejí. Řešení využívá relační databázi, SQL dotazy, funkce a splňuje požadavky na evidenci kolejní budovy, zaměstnanců, místností, studentů, rezervací a plateb. Řešení demonstruje nutnost použití omezení integrity v relační databázi pro dosažení přesnosti a konzistence dat.

Klíčová slova: Datová základna, datová integrita, relační databázová technologie, SQL

Table of content

List of figures:

List of tables:

1. Introduction:

Data has become one of the most valuable assets of any company and many would claim that it is changing the face of the world. Data can help companies boost their revenue, build more efficiently, target their audience with personalized ads, or even in some cases find a cure to a disease. The better data a company has, the more successful it is likely to become. This is where data integrity becomes key.

Data integrity is a topical, necessary and comprehensive issue and is being used for years in applications, databases, and projects. Nowadays, it is again one of the topics discussed, especially in times of increasing data on the Internet. These data should be protected against misuse of the correct integrity of the data, thus facilitating the combination of other devices with the protection of the data. It is often considered a part of data processing and is completely underestimated. From the point of view of databases, integrity is the key component of the design of a good database and it is not possible to reliably process and store data without it.

The issue of integrity constraints is current in databases of all kinds. A high-quality data base is needed nowadays for all projects that work with some data and for all businesses using data-processing software. Relevant and correct data makes it easy to search, especially to search for a larger amount of data, giving it a good value. Even the administrator can insert bad data into the system and thus correctly design into- grated constraints by doing what's going to work and avoiding damage to inappropriate data storage. Data Integrity works both in the database and in an application that is superstructure over a given database.

This bachelor thesis focuses on the issue of integrity constraints in database processing. It analyzes the integrity, reference, and the use of trigger and business rules. It also describes the use of individual restrictions and their benefits. MySQL uses SQL to implement integrity constraints, and MySQL has been used to preserve the database. The work is divided into two main sections, the theoretical part and the practical part.

This thesis document has 2 parts, the first part is theoretical part which defines the concepts and theoretical principles of the relational database, especially the problems and principles connected with the data integrity and their utilization. It deals with the transaction processing of the database; its main components are related. It describes the individual relationships in the database, the basic principles of creation and design of the database. It analyzes the SQL query language, individual integrity types, and their use in SQL together with trigger and business rules. It also describes the current state of the issue. The practical part demonstrates the use of integrity constraints using the SQL query language, which is used to create individual tables and scripts.

2. Aim of work and methodology:

2.1 Aim of work:

This bachelor thesis is focused on the issue of data integrity in relational database processing. Its aim is to define the theoretical principles of database, relational database, especially in data integrity, to map the current situation of this problem, to design, to check and to demonstrate possible usage of the integrity constraints.

2.2 Methodology:

This section briefly explains how this thesis was organized and explains the data source. The general understanding of "Integrity data in relational database processing" in the literature review of this study by referring the developers guide. The methodology used in this bachelor thesis was based on the study and analysis of available information sources and existing solutions in the given field. The main resources that I used in this thesis is 4 handbooks: Database System Concepts-Sixth edition (Abraham Silberschatz), Fundamental of database systems – Sixth edition (Ramez Elmasri, Shamkant B. Navathe), Learning SQL – Second edition (Alan Beaulieu), Sam's Teach Yourself MySQL in 21 days (Mark Maslakowski)

The thesis is about exploring "Integrity data in relational database processing" and I choose MySQL database management system for creating a project in the practical part because MySQL are the most popular open source database, extremely simple to download and install.

3. Database:

Database today are essential to every business. It is fair to say that databases play a critical role in almost all areas where computers are used, including business, electronic commerce, engineering, medicine, genetics, law, education, and library science.

The power of databases comes from a body of knowledge and technology that has developed over several decades and is embodied in specialized software called a database management system or DBMS or database system. A DBMS is a powerful tool for creating and managing large amounts of data efficiently and allowing it to persist over long periods of time, safety. [1]

A database is a collection of related data. By data, we mean known facts that can be recorded and that have implicit meaning. For example, consider the names, telephone numbers, and addresses of the people you know. You may have recorded this data in an indexed address book

or you may have stored it on a hard drive, using a personal computer and software such as Microsoft Access or Excel. This collection of related data with an implicit meaning is a database. [1]

A database has the following implicit properties:

- A database represents some aspect of the real world, sometimes called the miniworld or the universe of discourse (UoD). Changes to the miniworld are reflected in the database.
- A database is a logically coherent collection of data with some inherent meaning. A random assortment of data cannot correctly be referred to as a database.
- A database is designed, built, and populated with data for a specific purpose. It has an intended group of users and some preconceived applications in which these users are interested. [2]

A database is stored as a file or a set of files on magnetic disk or tape, optical disk, or some other secondary storage device. The information in these files may be broken down into records, each of which consists of one or more fields. Fields are the basic units of data storage, and each field typically contains information pertaining to one aspect or attribute of the entity described by the database. Records are also organized into tables that include information about relationships between its various fields. Although database is applied loosely to any collection of information in computer files, a database in the strict sense provides cross-referencing capabilities. Using keywords and various sorting commands, users can rapidly search, rearrange, group, and select the fields in many records to retrieve or create reports on aggregates of data. [1] [2]

The information in many databases consists of natural-language texts of documents; number-oriented databases primarily contain information such as statistics, tables, financial data, and raw scientific and technical data. Small databases can be maintained on personal-computer systems and may be used by individuals at home. These and larger databases have become increasingly important in business life, in part because they are now commonly designed to be integrated with other office software, including spreadsheet programs. [1] [2]

Typical commercial database applications include airline reservations, production management functions, medical records in hospitals, and legal records of insurance companies. The largest databases are usually maintained by governmental agencies, business organizations, and universities. These databases may contain texts of such materials as abstracts, reports, legal statutes, wire services, newspapers and journals, encyclopedias, and catalogs of various kinds. Reference databases contain bibliographies or indexes that serve as guides to the location of information in books, periodicals, and other published literature. Thousands of these publicly accessible databases now exist, covering topics ranging from law, medicine, and engineering to news and current events, games, classified advertisements, and instructional courses. [1] [2]

4. Relational database:

This part introduces the relational model of data, covering basic concepts such as the structure of relational databases, database schemas, keys, schema diagrams, relational query languages, and

relational operations describe the SQL query language, which is the standard for commercial relational DBMS.

The concept of a relational database was first proposed by English computer scientist E. F. Codd in 1970. He invented the relational model for database management and theoretical basis for relational databases and relational database management systems while working for IBM. Codd also defined the 12 rules of constituted a relational database as well as the twelve laws of online analytical processing (a term he himself coined). [2]

The first commercial implementations of the relational model became available in the early 1980s, such as the SQL/DS system on the MVS operating system by IBM and the Oracle DBMS. Since then, the model has been implemented in a large number of commercial systems. Current popular relational DBMSs (RDBMSs) include DB2 and Informix Dynamic Server (from IBM), Oracle and Rdb (from Oracle), Sybase DBMS (from Sybase) and SQLServer and Access (from Microsoft). In addition, several open source systems, such as MySQL and PostgreSQL, are available. [2]

4.1 Relational:

Relational databases are found in many organizations, used as a way of storing data and the relationships between different data points. It is a set of tables that data can be accessed from or reconstructed in a multitude of ways without having to rearrange the database tables.

In a relational database, each relation is a set of tuples. Each tuple is a list of attributes, which represents a single item in the database. Each tuple ("row") in a relation ("table") shares the same attributes ("columns"). Each attribute has a well-defined data type (int, string, etc.), defined ahead of time—schema in a relational database is static. [3]

4.2 Relational data model:

Relational data model is the primary data model, which is used widely around the world for data storage and processing. This model is simple, and it has all the properties and capabilities required to process data with storage efficiency.

The relational model represents the database as a collection of relations. Informally, each relation resembles a table of values or, to some extent, a flat file of records. It is called a flat file because each record has a simple linear or flat structure. [2]

When a relation is thought of as a table of values, each row in the table represents a collection of related data values. A row represents a fact that typically corresponds to a real-world entity or relationship. The table name and column names are used to help to interpret the meaning of the values in each row. [2]

4.2.1 Domain:

A domain D is a set of atomic values. By atomic we mean that each value in the domain is indivisible as far as the formal relational model is concerned. A common method of specifying a domain is to specify a data type from which the data values forming the domain are drawn. It is also useful to specify a name for the domain, to help in interpreting its values. [2]. Some examples of domains follow:

- Names: The set of character strings that represent names of persons.
- Employee_ages. Possible ages of employees in a company; each must be an integer value between 15 and 80.
- Academic_department_names. The set of academic department names in a university, such as Computer Science, Economics, and Physics.

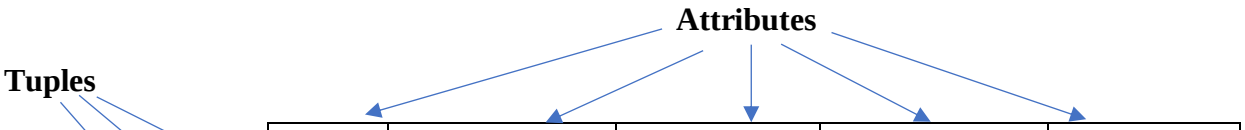
4.2.2 Table:

In relational data model, relations are saved in the format of Tables. This format stores the relation among entities. A table has rows and columns, where rows represents records and columns represent the attributes.

[3]

4.2.3 Tuple:

A single row of a table, which contains a single record for that relation is called a tuple. [3]



ID	Name	Date of birth	Phone number	Address
321	Dick Davidson	1995/02/01	000111222	Brno
456	Barbara Benson	1996/04/06	333444555	Plzen
879	Chung-cha Kim	1994/08/09	666777888	Praha

Table 1: student

4.2.4 Relation:

- Relation instance - A finite set of tuples in the relational database system represents relation instance. Relation instances do not have duplicate tuples.
- Relation schema - A relation schema describes the relation name (table name), attributes, and their names.
- Relation key - Each row has one or more attributes, known as relation key, which can identify the row in the relation (table) uniquely. [3]

Customer

cust_id	**fname**	**lname**
1	George	Blake
2	Sue	Smith

Table 2: customer

Account

account_id	**product_cd**	**cust_id**	**Balance $**
103	CHK	1	75
104	SAV	1	250
105	CHK	2	783
106	MM	2	500

Table 3: account

Product

product_cd	name
CHK	Checking
SAV	Savings
MM	Money market

Table 4: product

Transaction

txn_id	txn_type_cd	account_id	amount	date
978	DBT	103	100	2018-01-22
979	CDT	103	25	2018-02-05
980	DBT	104	250	2018-03-09
981	DBT	105	1000	2018-03-25
982	CDT	105	138	2018-04-02
983	CDT	105	77	2018-04-04
984	DBT	106	500	2018-03-27

Table 5: transaction

There are 4 tables representing 4 entities: customer, product, account and transaction. Looking across the top of customer table, you can see 3 columns: cust_id, fname (customer's first name) and lname(customer's last name). Looking down the side of the customer table, you can see two rows, one containing George Blake's data and one containing Sue Smith's data. The number of columns that a table may contain differs from server to server, but it is generally large enough not to be an issue. The number of rows that a table may contain is more matter of physical limits (i.e., how much disk drive space is available) and maintainability (i.e., how large a table can get before it becomes difficult to work with) than of database server limitations.

Each table in a relational database includes information that uniquely identifies a row in that table (primary key), along with additional information needed to describe the entity completely. Looking again at the customer table, the cust_id column holds a different number for each customer, George Blake, for example, can be uniquely identified by customer ID #1. No other customer will ever be assigned that identifier, and no other information is needed to locate George Blake's data in the customer table.

Every database server provides a mechanism for generating unique sets of numbers to use as values, so we won't need to worry about keeping track of what numbers have been assigned. [4]

While we might chose to use the combination of the *fname* and *lname* columns as the primary key (a primary key consisting of 2 or more columns is known as the compound key), there could easily be 2 or more people with the same first and last name that have account at the bank, Therefore we chose to include the *cust_id* column in the customer table specifically for use as the primary key column.

It may seen wasteful to store the same data many times, but the relational model is quite clear on what redundant data maybe stored. For example, it is proper for the *account table* to include a column for the unique identifier of the customer who open the account, but it is not proper to include the customer's first and last name in the *account table* as well. If a customer were to change her name, for example, you want to make sure that there is only one place in the database that hold customer's name; otherwise, the data may be changed in one place but not another, causing the data in the database to be unreliable. The proper place for this data is the *customer table*, and only the cust_id value should be included in the other tables. It is also not proper for a single column to contain multiple piece of information, such as a name column that contain both a person's last name and first name, or an address column that contains street, city, state, and zip code information. The process is refining a database design to ensure that each independent piece of information is only one place (except for foreign keys) is known as normalization.

The **null** value is a special value that signifies that the value is unknown or does not exist. For example, suppose as before that we include the attribute *phone number* in the *customer table*. It may be that an customer does not have a phone number at all, or that the telephone number is unlisted. We would then have to use the null value to signify that the value is unknown or does not exist

Getting back to the four tables, you may wonder how you would use this tables to find George Blake's transactions against his checking account. First, you would find George Blake's unique identifier in the *customer table*. Then you would find the row in the account table whose cust_id column contains George's unique identifier and whose *product_cd* columns matches the row in the *product table* whose *name* column equals "Checking". Finally, you would locate the rows in the *transaction table* whose *account_id* column matches the unique identifier from the *account* table. This may sound complicated, but you can do it in a single command called Query, using the SQL language, as you will see shortly. [4]

4.2.5 Some Terminology:

Entity: something of interest to the database user community.

(As the example before, entities of that database is customer, account, product, transaction.)

Column or attribute: an individual piece of data stored in a table.

Row or tuple: A set of columns that together completely describe an entity or some action on an entity. Also called a record.

Table: A set of rows, held either in memory (nonpersistent) or on permanent storage (persistent).

Result set: Another name for a nonpersistent table, generally the result of an SQL Query.

Primary key: One or more columns that can be use as the unique identifier for each row in a table.

Foreign key: one or more columns that can be used together to identify a single row in another table. [4]

4.3 Database normalization:

Normalization is the process of organizing data in a database. This includes creating tables and establishing relationships between those tables according to rules designed. The goal is to generate a set of relation schemas that allows us to store information without unnecessary redundancy, yet also allows us to retrieve information easily. The approach is to design schemas that are in an appropriate normal form. To determine whether a relation schema is in one of the desirable normal forms, we need additional information about the real-world enterprise that we are modeling with the database. [5]

Redundant data wastes disk space and creates maintenance problems. If data that exists in more than one place must be changed, the data must be changed in exactly the same way in all locations. A customer address change is much easier to implement if that data is stored only in the Customers table and nowhere else in the database. [6]

There are a few rules for database normalization. Each rule is called a "normal form." If the first rule is observed, the database is said to be in "first normal form." If the first three rules are observed, the database is considered to be in "third normal form." Although other levels of normalization are possible, third normal form is considered the highest level necessary for most applications. [6]

First Normal Form

- Eliminate repeating groups in individual tables.
- Create a separate table for each set of related data.
- Identify each set of related data with a primary key. [6]

Second Normal Form

- Create separate tables for sets of values that apply to multiple records.
- Relate these tables with a foreign key. [6]

Third Normal Form

- Eliminate fields that do not depend on the key. [6]

5. Query Language:

5.1 Query:

A query is a question, often expressed in a formal way. A database query can be either a select query or an action query. A select query is a data retrieval query, while an action query asks for additional operations on the data, such as insertion, updating or deletion. [7]

5.2 Query language:

A query language is a language in which a user requests information from the database, and Microsoft Structured Query Language (SQL) is the standard. Under the SQL query umbrella, there are several extensions of the language, including MySQL, Oracle SQL and NuoDB. Query languages for other types of databases, such as NoSQL databases and graph databases, include Cassandra Query Language (CQL), Neo4j's Cypher, Data Mining Extensions (DMX) and XQuery. [5]

Queries can accomplish a few different tasks. Primarily, queries are used to find specific data by filtering specific criteria. Queries can also calculate or summarize data, as well as automate data management tasks. Other queries include parameter, totals, crosstab, make table, append, update and delete. For example, a parameter query runs variations of a particular query, which prompts a user to insert a field value, and then it uses that value to create the criteria, while totals queries allow users to group and summarize data. [7]

5.3 SQL (Structure query language) :

There are several database query languages in use, either commercially or experimentally. In this part, we will talk the most widely used query language, SQL. Although we refer to the SQL language as a “query language,” it can do much more than just query a database. It can define the structure of the data, modify data in the database, and specify security constraints. It is not our intention to provide a complete users’ guide for SQL. Rather, we present SQL’s fundamental constructs and concepts. Individual implementations of SQL may differ in detail or may support only a subset of the full language.

5.3.1 Overview of the SQL Query Language:

IBM developed the original version of SQL, originally called Sequel, as part of the System R project in the early 1970s. The Sequel language has evolved since then, and its name has changed to SQL (Structured Query Language). Many products now support the SQL language. SQL has clearly established itself as the standard relational database language. [5]

In 1986, the American National Standards Institute (ANSI) and the International Organization for Standardization (ISO) published an SQL standard, called SQL-86. ANSI published an extended standard for SQL, SQL-89, in 1989. The next version of the standard was SQL-92 standard, followed by SQL:1999, SQL:2003, SQL:2006, and most recently SQL:2008. The bibliographic notes provide references to these standards. [5]

The SQL language has several parts:

- Data-definition language (DDL). The SQL DDL provides commands for defining relation schemas, deleting relations, and modifying relation schemas.
- Data-manipulation language (DML). The SQL DML provides the ability to query information from the database and to insert tuples into, delete tuples from, and modify tuples in the database.
- Integrity. The SQL DDL includes commands for specifying integrity constraints that the data stored in the database must satisfy. Updates that violate integrity constraints are disallowed.
- View definition. The SQL DDL includes commands for defining views
- Transaction control. SQL includes commands for specifying the beginning and ending of transactions.
- Embedded SQL and dynamic SQL. Embedded and dynamic SQL define how SQL statements can be embedded within general-purpose programming languages, such as C, C++, and Java.
- Authorization. The SQL DDL includes commands for specifying access rights to relations and views. [5]

5.3.2 SQL constraints:

SQL constraints are used to specify rules for the data in a table.

Constraints are used to limit the type of data that can go into a table. This ensures the accuracy and reliability of the data in the table. If there is any violation between the constraint and the data action, the action is aborted.

Constraints can be column level or table level. Column level constraints apply to a column, and table level constraints apply to the whole table.

The following constraints are commonly used in SQL:

- NOT NULL - Ensures that a column cannot have a NULL value
- UNIQUE - Ensures that all values in a column are different
- PRIMARY KEY - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table
- FOREIGN KEY - Uniquely identifies a row/record in another table
- CHECK - Ensures that all values in a column satisfies a specific condition
- DEFAULT - Sets a default value for a column when no value is specified
- INDEX - Used to create and retrieve data from the database very quickly

5.3.3 SQL Data Definition and Data Types:

The set of relations in a database must be specified to the system by means of a data-definition language (DDL). The SQL DDL allows specification of not only a set of relations, but also information about each relation, including:

• The schema for each relation.

• The types of values associated with each attribute.

• The integrity constraints.

• The set of indices to be maintained for each relation.

• The security and authorization information for each relation.

• The physical storage structure of each relation on disk. [5]

SQL uses the terms table, row, and column for the formal relational model terms relation, tuple, and attribute, respectively. We will use the corresponding terms interchangeably. The main SQL command for data definition is the CREATE statement, which can be used to create schemas, tables (relations), and domains (as well as other constructs such as views, assertions, and triggers) [2]

The **CREATE TABLE** command is used to specify a new relation by giving it a name and specifying its attributes and initial constraints. The attributes are specified first, and each attribute is given a name, a data type to specify its domain of values, and any attribute constraints, such as NOT NULL. The key, entity integrity, and referential integrity constraints can be specified within the CREATE TABLE statement after the attributes are declared, or they can be added later using the **ALTER TABLE** command [2]

Typically, the SQL schema in which the relations are declared is implicitly specified in the environment in which the CREATE TABLE statements are executed. Alternatively, we can explicitly attach the schema name to the relation name, separated by a period. For example, by writing **CREATE TABLE** COMPANY.EMPLOYEE ...

rather than **CREATE TABLE** EMPLOYEE ...

The SQL standard supports a variety of built-in types, including:

• ***char***(n): A fixed-length character string with user-specified length n. The full form, character, can be used instead.

• ***varchar***(n): A variable-length character string with user-specified maximum length n. The full form, character varying, is equivalent.

• ***int***: An integer (a finite subset of the integers that is machine dependent). The full form, integer, is equivalent.

• ***smallint***: A small integer (a machine-dependent subset of the integer type).

• ***numeric*** (p, d): A fixed-point number with user-specified precision. The number consists of p digits (plus a sign), and d of the p digits are to the right of the decimal point. Thus, numeric (3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

• ***real, double precision***: Floating-point and double-precision floating-point numbers with machine-dependent precision.

• ***float***(n): A floating-point number, with precision of at least n digits. [5]

Each type may include a special value called the ***null*** value. A null value indicates an absent value that may exist but be unknown or that may not exist at all. In certain cases, we may wish to prohibit null values from being entered. [5]

Not null: The not null constraint on an attribute specifies that the null value is not allowed for that attribute; in other words, the constraint excludes the null value from the domain of that attribute. [5]

create table department

(dept name varchar (20),

building varchar (15),

budget numeric (12,2),

primary key (dept name));

create table course (course id varchar (7),

title varchar (50),

dept name varchar (20),

credits numeric (2,0),

primary key (course id),

foreign key (dept name) references department);

create table instructor (ID varchar (5),
name varchar (20) not null,
dept name varchar (20),
salary numeric (8,2),
primary key (ID),
foreign key (dept name) references department);

create table section (course id varchar (8),
sec id varchar (8),
semester varchar (6),
year numeric (4,0),
building varchar (15),
room number varchar (7),
time slot id varchar (4),
primary key (course id, sec id, semester, year),
foreign key (course id) references course);
create table teaches (ID varchar (5),
course id varchar (8),
sec id varchar (8),
semester varchar (6),
year numeric (4,0),
primary key (ID, course id, sec id, semester, year),
foreign key (course id, sec id, semester, year) references section,
foreign key (ID) references instructor);

A newly created relation is empty initially. We can use the **insert** command to load data into the relation. For example, if we wish to insert the fact that there is an instructor named Smith in the Biology department with instructor id 10211 and a salary of $66,000, we write:

insert into instructor values (10211, 'Smith', 'Biology', 66000);

We can use the **delete** command to delete tuples from a relation. The command

delete from student;

To remove a relation from an SQL database, we use the **drop** table command. The drop table command deletes all information about the dropped relation from the database. The command

drop table r;

is a more drastic action than

delete from r;

We use the **alter** table command to add attributes to an existing relation. All tuples in the relation are assigned null as the value for the new attribute. The form of the alter table command is

alter table r **add** A D;

where r is the name of an existing relation, A is the name of the attribute to be added, and D is the type of the added attribute. We can drop attributes from a relation by the command

alter table r **drop** A; [5]

5.3.4 Basic Structure of SQL Queries:

The basic structure of an SQL query consists of three clauses: **select**, **from**, and **where**. The query takes as its input the relations listed in the from clause, operates on them as specified in the where and select clauses, and then produces a relation as the result. We introduce the SQL syntax through examples, and describe the general structure of SQL queries later. [5]

"Find the names of all instructors."

select name **from** instructor;

"Find the department names and ID of all instructors,"

select ID, dept_name **from** instructor;

"Find the names of all instructors in the Computer Science department who have salary greater than $70,000."

select name **from** instructor

where dept name = 'Comp. Sci.' **and** salary > 70000;

Queries on Multiple Relations: So far our example queries were on a single relation. Queries often need to access information from multiple relations.

Looking at the schema of the relation instructor, we realize that we can get the department name from the attribute dept name, but the department building name is present in the attribute building of the relation department. To answer the query, each tuple in the instructor relation must be matched with the tuple in the department relation whose dept name value matches the dept name value of the instructor tuple.

select name, instructor.dept name, building

from instructor, department

where instructor.dept name= department.dept name;

As we have seen earlier, an SQL query can contain three types of clauses, the select clause, the from clause, and the where clause. The role of each clause is as follows:

- The **select** clause is used to list the attributes desired in the result of a query.
- The **from** clause is a list of the relations to be accessed in the evaluation of the query.
- The **where** clause is a predicate involving attributes of the relation in the from clause. [5]

The Join: To make the life of an SQL programmer easier for this common case, SQL supports an operation called the natural join, which we describe below. In fact SQL supports several other ways in which information from two or more relations can be joined together. [5]

"For all instructors in the university who have taught some course, find their names and the course ID of all courses they taught", which we wrote earlier as:

select name, course id

from instructor, teaches

where instructor.ID= teaches.ID;

This query can be written more concisely using the natural-join operation in SQL as:

select name, course id

from instructor **join** teaches;

String Operations: SQL specifies strings by enclosing them in single quotes.

- 'Intro%' matches any string beginning with "Intro".

• '%Comp%' matches any string containing "Comp" as a substring, for example, 'Intro. to Computer Science', and 'Computational Biology'.

• '_ _ _ ' matches any string of exactly three characters.

• '_ _ _%' matches any string of at least three characters

SQL expresses patterns by using the like comparison operator. Consider the query "Find the names of all departments whose building name includes the substring 'Watson'." This query can be written as:

select dept_name

from department

where building like '%Watson%';

Attribute Specification in Select Clause: The asterisk symbol " * " can be used in the select clause to denote "all attributes."

select * instructor

from instructor, teaches

where instructor.ID= teaches.ID;

Ordering the Display of Tuples: SQL offers the user some control over the order in which tuples in a relation are displayed. The order by clause causes the tuples in the result of a query to appear in sorted order.

To list in alphabetic order all instructors in the Physics department, we write:

select name

from instructor

where dept name = 'Physics'

order by name;

Where Clause Predicates: SQL includes a between comparison operator to simplify where clauses that specify that a value be less than or equal to some value and greater than or equal to some other value. If we wish to find the names of instructors with salary amounts between $90,000 and $100,000, we can use the between comparison to write:

select name

from instructor

where salary **between** 90000 **and** 100000;

Aggregate Functions: Aggregate functions are functions that take a collection (a set or multiset) of values as input and return a single value. SQL offers five built-in aggregate functions:

- Average: avg
- Minimum: min
- Maximum: max
- Total: sum
- Count: count

Consider the query "Find the average salary of instructors in the Computer Science department." We write this query as follows:

select avg (salary)

from instructor

where dept_name = 'Comp. Sci.'; [5]

5.4 MySQL:

I have decided that the practical part for this thesis be run against a MySQL database. I have found its server to be extremely simple to download and install. It is one of the most commonly used open source database server and its server is available for free.

MySQL was developed by a consulting firm in Sweden called TcX. They were in need of a database system that was extremely fast and flexible. Unfortunately (or fortunately, depending on your point of view), they could not find anything on the market that could do what they wanted. So, they created MySQL, which is loosely based on another database management system called mSQL. The product they created is fast, reliable, and extremely flexible. It is used in many places throughout the world. Universities, Internet service providers and nonprofit organizations are the main users of MySQL, mainly because of its price (it is mostly free). Lately, however, it has begun to permeate the business world as a reliable and fast database system. [8]

MySQL is often confused with SQL, the structured query language developed by IBM. It is not a form of this language but a database system that uses SQL to manipulate, create, and show data. MySQL is a program that manages databases, much like Microsoft's Excel manages spreadsheets. SQL is a programming language that is used by MySQL to accomplish tasks within a database, just as Excel uses VBA (Visual Basic for Applications) to handle tasks with spreadsheets and workbooks. [8]

6. Integrity data:

6.1 Integrity data:

Data integrity refers to the reliability and trustworthiness of data throughout its lifecycle. It can describe the state of your data—e.g., valid or invalid—or the process of ensuring and preserving the validity and accuracy of data. Error checking and validation, for example, are common methods for ensuring data integrity as part of a process.

Data integrity is not to be confused with data security. Data security refers to the protection of data, while data integrity refers to the accuracy and consistency of data.

Data security focuses on how to minimize the risk of leaking intellectual property, business documents, healthcare data, emails, trade secrets, and more. Some data security tactics include permissions management, data classification, identity and access management, threat detection, and security analytics.

Maintaining or keeping data consistent throughout its lifecycle is a matter of protecting it (security) so that it's reliable. And data that's reliable is simply able to meet certain standards, with which compliance is necessary. [9]

Data is expected to be:

- Attributable – Data should clearly demonstrate who observed and recorded it, when it was observed and recorded, and who it is about.
- Legible – Data should be easy to understand, recorded permanently and original entries should be preserved.
- Contemporaneous – Data should be recorded as it was observed, and at the time it was executed.
- Original – Source data should be accessible and preserved in its original form.
- Accurate – Data should be free from errors and conform with the protocol. [10]

6.2 Types of data integrity:

6.2.1 Entity integrity:

Entity integrity ensures that each row of a table is uniquely identified, so that it can be retrieved separately if necessary. The concept of entity integrity is basic to database design and implementation. The primary key of a table is one or more columns that you use to uniquely

identify each row of the table. The value set of a primary key is unique; no two values may be the same.

The primary key enforces entity integrity. The rules of entity integrity state that no primary key can be null and that no change can render the primary key null. These rules guarantee that every row of a table is accessible, whether you're retrieving data or modifying it. You can retrieve each row separately by specifying the value of a primary key.

Null isn't a space (for character and date data types) or a zero value (for number data types). Null is a condition that represents at least one of three states of the data value: not applicable, applicable but not available, or applicability unknown. [11]

6.2.2 Referential Integrity:

Referential integrity is the logical dependency of a foreign key on a primary key. The integrity of a row that contains a foreign key depends on the integrity of the row that it references—the row that contains the matching primary key.

In other words, any foreign key field must agree with the primary key that is referenced by the foreign key. Thus, any primary key field changes must be applied to all foreign keys, or not at all. The same restriction also applies to foreign keys in that any updates (but not necessarily deletions) must be propagated to the primary parent key. From that, table relationships always be consistent. [12]

6.2.3 Domain integrity:

Domain integrity ensures that all the data items in a column fall within a defined set of valid values. Each column in a table has a defined set of values, such as the set of all numbers for Pubs..zip (five-digit), the set of all character strings for Pubs..au_lname, and the set of all dates for Sales..ord_date. When you limit the value assigned to an instance of that column (an attribute), you are enforcing domain integrity. Domain integrity enforcement can be as simple as choosing the correct data type and length for a column. In the Pubs database's titles table, the column PubDate has a datetime data type and is not nullable. The creator of the titles table chose a non-null condition and a datetime data type for this column. This choice ensures that a value is always present for the PubDate column, and that the value is a valid date, such as 11/22/99. In this way, the creator put in place some controls to maintain domain integrity. [11]

7. Practical:

7.1 Description of the project:

As part of a demonstration of the use of integrity constraints, I analyze student's dormitories of a university, called Dormitory Management System. This project is linked to a website of dormitory management of a university. The website is managed and updated by administrator who will manage all the activities of the dormitories include information and activities of employees, students, and dormitories. We have two types of actors: students who are the users and Administrator which is authority of the Hall. After creating an account by fulfilling all the information that is needed, users can book a room, do the payment, upload personal details and check the booking with all information about time and payment.

The project is a relational database that provides internal records of dormitories, rooms, employees, booking information, personal data of students, and payments of the rooms. All the internal records will be realized through integrity constraints, resulting in data consistency and accuracy. The administrator will easy to check, control and manage all the data of dormitories.

7.2 Implementation Tools:

The implementation Tools that I used for creating the project for this thesis is MySQL Community Server 8.0.15. MySQL Community Server is a freely downloadable version of the world's most popular open source database. It was installed and worked on Window 10 operating system.

7.3 Entity relationship diagram:

7.3.1 Entity relationship diagram explain:

Peter Chen developed ERDs in 1976. Since then Charles Bachman and James Martin have added some slight refinements to the basic ERD principles.

Entity relationship diagram (ERD) shows the relationships of entities in a database. It defines the entities, their attributes, illustrates the logical structure of databases and sketches out the design of a database by expressing the links between them.

An ERD contains different symbols and connectors that visualize two important information: The major entities within the system scope, and the inter-relationships among these entities. And that's why it's called "Entity" "Relationship" diagram (ERD)!

Entities in ERD is business objects such as people, room, tangible and intangible business objects and ERD helps us to easier and faster visualize how these entities relate to each other with in a system. It is needed for designing relational database

7.3.2 Constructing ERD:

Here are steps we need to do for constructing ERD:

- Identify the entities**.**
- Identify relationships of these entities**.**
- Describe the relationship.
- Add attribute.
- Complete the diagram.

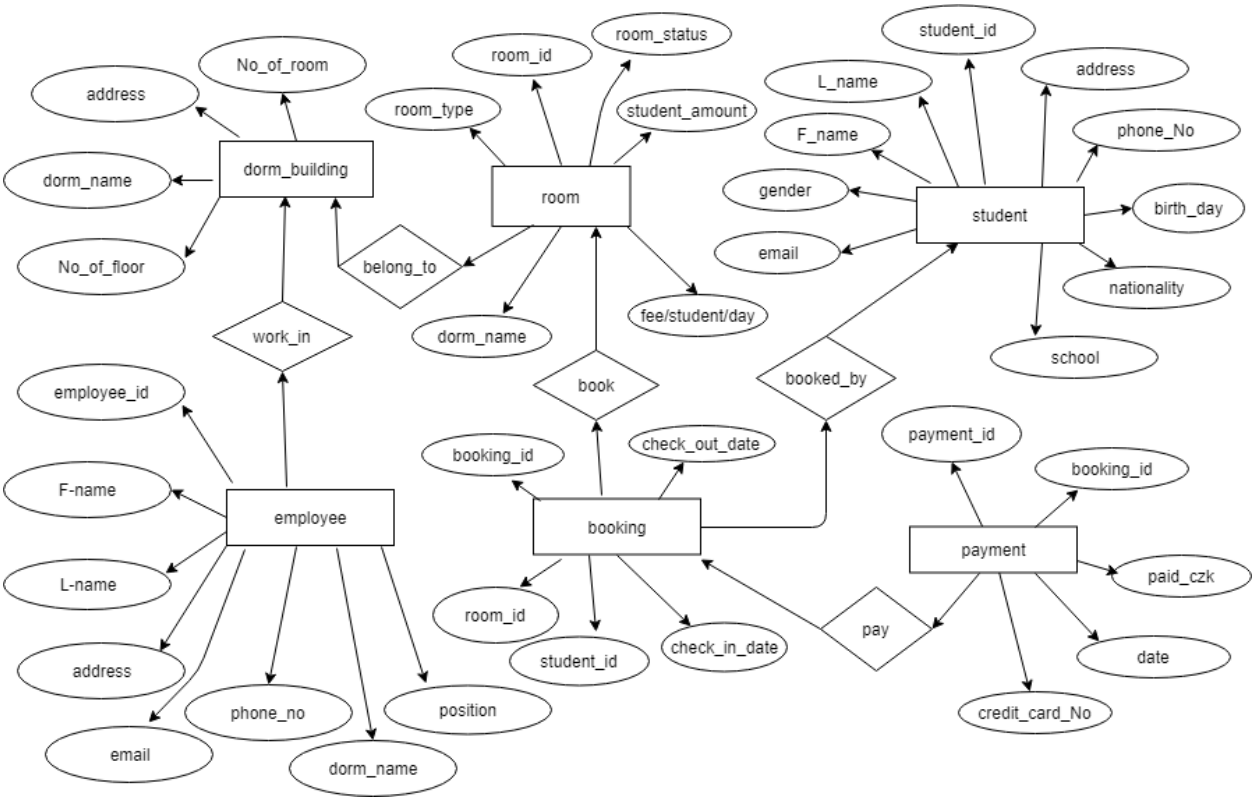


Figure 1: ERD of dormitory management system

7.3.3 Enhanced ERD:

Enhanced ERD is diagram where relationships of objects are defined clearly. The diagram was created in MySQL Workbench. We have 3 types of relationship:

- One-to-one relationship: one set of filed in a database table is associated with one and only one set of filed in another table.

- One-to-many relationship: one set of filed in database table can be associated with one or more sets of filed in another table.

- Many-to-many relationship: multiple set of filed in a database table are associated with multiple set of filed in another table.

To have the relationship between 2 entities, we need to link foreign key of an entity to primary key of other entity. In this project, I have 1 type of relationship: one-to-many:

- Each dormitory building has many rooms and many employees. Each room belong to one dormitory building and each employee is working in one dorm_building
- Each room can be booked many times depend on its status in the current time and each booking can book one room.
- Each student can make many bookings in deference time after previous booking has been finished.
- Each booking for a room can be payed in many times by students and each transaction (payment) is done for one booking.

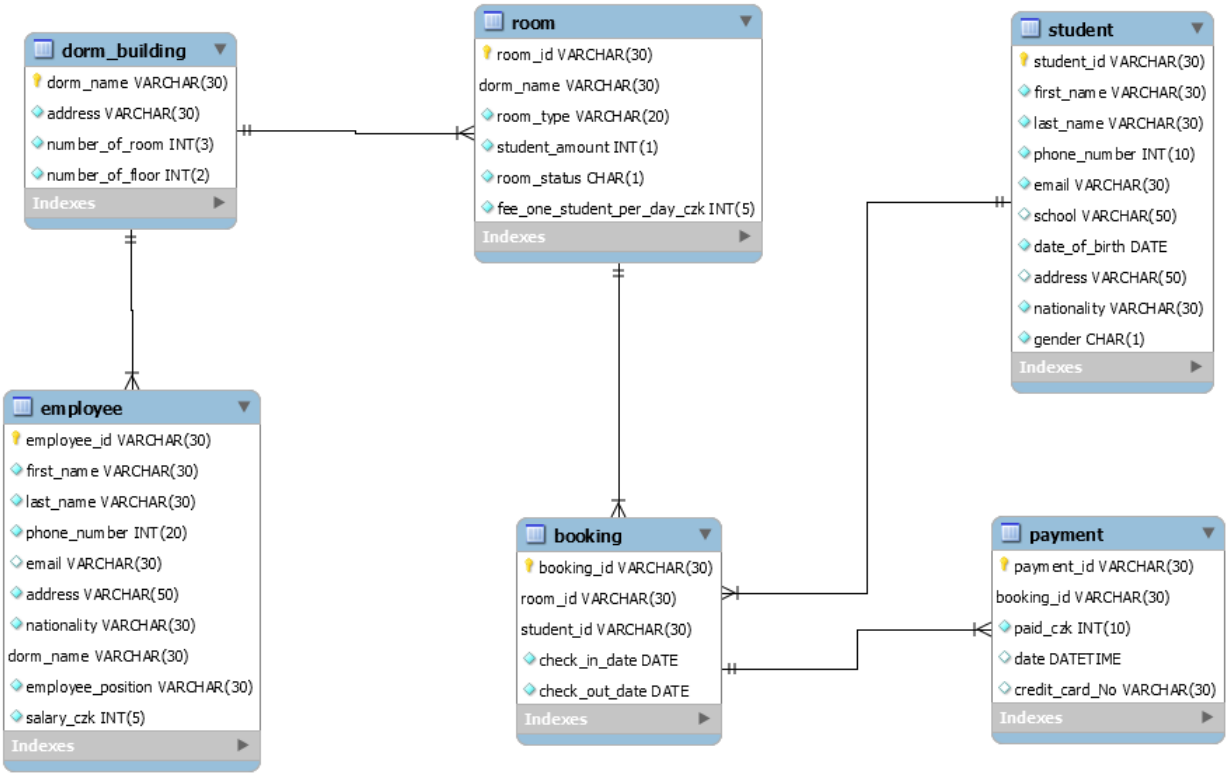


Figure 2: EERD of dormitory management system

7.4 Object oriented analysis:

In this part, every object will be found out and discussed. All the attributes and methods of objects will also be described, type of attributes and values of data.

Dorm_building object:

dorm_buiding is all basic information about dormitory buildings of the university.

Attribute	Description	Data Type
dorm_name	Name of the dormitory building	Varchar (30)
address	Address of the dormitory building	Varchar (30)
room_amount	Number of rooms of the dormitory building	Integer (3)
floor_amount	Number of floors of the dormitory building	Integer (2)

Table 6: dorm_building object

Employee object:

Employee records contain basic information about employees, which the dormitory uses for internal purposes.

Attribute	Description	Data type
employee_id	Employee's ID	Varchar (30)
first_name	Employee's first name	Varchar (30)
last_name	Employee's last name	Varchar (30)
phone_number	Employee's phone number	Integer (20)
email	Employee's email	Varchar (30)
address	Employee's address	Varchar (50)
nationality	Employee's nationality	Varchar (20)
dorm_name	Dormitory building's name where employee's working in	Varchar (30)
employee_position	Employee's position	Varchar (30)
salary_czk	Employee's salary/month in crown	Integer (5)

Table 7: employee object

Room object:

Room object is basic information of rooms in dormitories, from that student can choose type of the room which they want to live in and administrator can manage room for student.

Attribute	Description	Data type
room_id	Room's ID	Varchar (30)
dorm_name	Dormitory building's name that the room belong to	Varchar (30)
room_type	Type of the room (1 bed, 2 beds, 3 beds)	Varchar (20)
student_amount	Number of students is living in the room.	Integer (1)
room_status	Room's status (full? Y/N)	Char (1)
fee/student/day_czk	Fee/day for 1 student in the room	Integer (5)

Table 8: room object

Student object:

student object contains personal information of student, room, check-in, check-out date and paymentID. Students can be students from other university.

Attribute	Description	Data type
student_id	Identity given by university authority	Varchar (30)
first_name	Student's first name	Varchar (30)
last_name	Student's last name	Varchar (30)
phone_number	Student's contact number	Integer (20)
email	Student's email address	Varchar (30)
school	Student's school	Varchar (50)
date_of_birth	Student's birthday	Date
address	Student's address	Varchar (50)
nationality	Student's nationality	Varchar (30)
gender	Student's gender (M/F)	Char (1)

Table 9: student object

Booking object:

Booking object contains information of check-in and check-out date of students.

Attribute	Description	Data type
booking_id	Student's booking-ID	Varchar (30)
room_id	ID of the room that student have booked	Varchar (30)
student_id	ID of student have booking	Varchar (30)
check_in_date	Check-in time of student	Date
check_out_date	Check-out time of student	Date

Table 10: booking object

Payment object:

Payment object contains information about payment of students

Attribute	Description	Data type
payment_id	Transaction ID	Varchar (30)
booking_id	Booking-ID of the payment	Varchar (30)
paid_czk	Amount of money in each transaction	Integer (10)
date	Date of payment	Datetime
credit_card_No	Credit card number	Varchar (30)

Table 11: payment object

7.5 Create script:

All scripts were typed manually without the use of code generators.

Table dorm_building:

CREATE TABLE dorm_building (

dorm_name **varchar**(30) **primary key**,

address **varchar**(30) **not null**,

number_of_room **int**(3) **not null**,

number_of_floor **int**(2) **not null**);

The primary key of this table is dormitory building's name, which is unique for each building and it's identified by administrator.

Table employee:

CREATE TABLE employee (

employee_id **varchar**(30) **primariy key**,

first_name **varchar**(30) **not null**,

last_name **varchar**(30) **not null**,

phone_number **int**(20) **unique not null**,

email **varchar**(30**) unique**,

address **varchar**(50) **not null**,

nationality **varchar**(30) **not null**,

dorm_name **varchar**(30) **not null**,

employee_position **varchar**(30) **not null**,

salary_czk **int**(5) **not null**),

CONSTRAINT work_in **foreign key** (dorm_name) **references** dorm_building(dorm_name));

CONSTRAINT check_phone_number_employee **check**(phone_number **like**

'[+][1−9][2−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9] '

OR phone_number **like**

'[+][1−9][0−9][2−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9] '

OR phone_number **like**

‘[+][1−9][0−9][0−9][2−9][0−9][0−9][0−9][0−9][0−9][0−9] [0−9][0−9] ‘

OR phone_number **like**

‘[+][1−9][0−9][0−9][0−9][2−9][0−9][0−9][0−9][0−9] [0−9][0−9][0−9][0−9] ‘)

);

The primary key of this table is employee_id, which is unique for each employee and it’s identified by administrator.

Phone numbers are processed with regular expressions to specify international prefixes and subsequent numbers. Phone number of device data, constituting a restriction on UNIQUE keywords.

Table room:

Table room is basic information of rooms in dormitories, include id, type, status, fee per day in crown and dormitory building which the room is belong to.

CREATE TABLE room(

room_id **varchar**(30) **primary key**,

dorm_name **varchar**(30) **not null**,

room_type **varchar**(20) **not null**,

student_amount **int**(1) **not null**,

room_status **char**(1) **not null**,

fee/student/day_czk **int**(5) **not null**

CONSTRAINT belong_to **foreign key** (dorm_name) **references** dorm_building(dorm_name)

);

Table student:

CREATE TABLE student (

student_id **varchar**(30) **primary key**,

first_name **varchar**(30) **not null**,

last_name **varchar**(30) **not null**,

phone_number **int**(10) **unique not null**,

email **varchar**(30) **unique not null**,

school **varchar**(50),

date_of_birth **date not null**,

address **varchar**(50),

nationality **varchar**(30) **not null**,

gender **char**(1) **not null**),

CONSTRAINT check(gender in('M', 'F')),

CONSTRAINT check_date_of_birth **CHECK**(date_of_birth < getdate())

CONSTRAINT check_phone_number_student **CHECK**(phone_number **like** ‘[+][1−9][2−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9] ‘

OR phone_number **like**

‘[+][1−9][0−9][2−9][0−9][0−9][0−9][0−9][0−9][0−9][0−9] ‘

OR phone_number **like**

‘[+][1−9][0−9][0−9][2−9][0−9][0−9][0−9][0−9][0−9] [0−9][0−9] ‘

OR phone_number **like**

‘[+][1−9][0−9][0−9][0−9][2−9][0−9][0−9][0−9][0−9] [0−9][0−9][0−9][0−9] ‘)

);

The date_of_birth and gender is defined by domain integrity using constraint check to ensure that the birthday of student is ealer than the current date and gender can be defined by only 2 values, F or M (male or female).

Table booking:

CREATE TABLE booking (

booking_id **varchar**(30) **primary key**,

room_id **varchar**(30) **not null**,

student_id **varchar**(30) **not null**,

check_in_date **date not null**,

check_out_date **date not null**,

CONSTRAINT check_date_booking **CHECK** (check_in_date < check_out_date),

CONSTRAINT check_check_in_date **CHECK** (check_in_date > getdate())

CONSTRAINT book **foreign key** (room_id) **references** room(room_id)

);

The constraint check is using again to ensure that check_out_date is later than check_in_date and the check_in_date is later than the current date.

Reference integrity is supported here by defining a unique combination of number of booking and student identifiers, so no student can book twice on the time overlaps. Additional restriction requires the implementation of an additional SQL function that returns the number of violations of the specified rule. The return value of 0 corresponds to the non-violation of the rule and allows the insertion of a new record when using this function in the integrity constraint. Otherwise, insertion is disabled with an integrity violation message.

CREATE FUNCTION checkStudentBookRoom()

RETURNS int

BEGIN

RETURN(SELECT COUNT(*) FROM

(SELECT * FROM booking

JOIN student

ON booking.student_id = student.student_id)

AS a

JOIN (SELECT * FROM booking2 JOIN student

ON booking.student_id = student.student_id)

AS b

ON a.booking_id <> b.booking_id

AND a.check_in_date < b.check_out_date

AND a.check_out_date > b.check_in_date

AND a.student_id = b.student_id)

END

DELIMITER ;

ALTER TABLE student **ADD CONSTRAINT** check_student_book_room
CHECK(dbo.checkStudentBookRoom() = 0);

Table payment:

CREATE TABLE payment (

payment_id **varchar**(30) **primary key**,

booking_id **varchar**(30) **not null**,

paid_czk **int**(10) **not null**,

date datetime **DEFAULT** current_timestamp **on update** current_timestamp,

credit_card_No **varchar**(30),

CONSTRAINT pay_for **foreign key**(booking_id) **references** booking(booking_id));

The default constraint in the date of payment will provide the current date when the transaction happens so the data will be accurate and not missing.

7.6 Summary:

The database is a set of complex issues that are part of its design and operation. The outlined solution is a demonstration of a possible solution to the issue of integrity constraints that are supported by SQL functions. The work demonstrates the particular use of SQL and the application of the integrity constraints to the first table, using a combination of keyframes and other general functions, and combining various limitations. Generally, entity and reference integrity are used, along with an example of possible further narrowing of the allowed input data.

Domain integrity in this solution demonstrates the use of SQL functions and the generally valid principle of the impossibility of using subqueries for the CHECK clause.

8. Conclusion:

The whole thesis analyzes the issue of data integrity in relational database data. It delimitates both theoretically and practically the use of these constraints to the most practical and practical use of solutions to problems. The result of the work was the introduction of integrity constraints in operation and the acquisition of consistent and accurate data to simplify the work with them and to help realize the exact relationships between them and their context, which I demonstrated at the enterprise with typical database characteristics.

The applied methodology of the given bachelor thesis was based on the study and analysis of available information sources and existing solutions in the given area. The most important

were the methods and techniques of relational and database technology in the connection with the problematic of the integrity. The conclusions of this bachelor thesis are formulated based on the theoretical knowledge and the results obtained.

The benefit of this solution is the immediate application of the constraint already in the creation of the database or in the creation of the entire solution. A quality database system has stability and is easier to maintain, and any application running over them can report from such data.

The application of integrity constraints eases the work of an add-on application that can devote itself to more important application-related activities. Correct and valid data do not just work for programmers, especially for users who can further use the output data and benefit from the correctness. The relevant corrective data for the correct system operation and the essentials of not having the correct and unnecessary handling of the data are worthless because they are worthless. Enforcing these rules requires complex thinking, SQL knowledge, databases, and programming. However, it brings its results and should be an integral part of the database creation. In addition, today's widespread use of the Internet, various portal sites and applications, as standard mobile data, is the foundation of success and is also a source of information-gathering. Correctly formatted, non-accurate, large-scale, easier data traceability and clarity.

Finally, data integrity is a necessity, but it requires high quality requirements. Therefore, accountability, conscientiousness and diligence must be approached.

9. References

[1] T. E. o. E. Britannica, "Database," 26 September 2018. [Online].

[2] Ramez Elmasri, "Fundamentals of Database systems," United States of America, Addison-Wesley, 2011, p. 1201.

[3] R. Millman, "What is a relational database," 20 Aug 2018. [Online].

[4] A. Beaulieu, "Learning SQL," United state of America, O'Reilly Media, 2009, p. 319.

[5] A. Silberschatz, Database System Concepts, New York: McGraw-Hill, 2012, p. 1376.

[6] "Description of the database normanization basics," 10 05 2017. [Online].

[7] A. Hughes, "Query," 11 05 2018. [Online].

[8] M. Maslakowski, "Sam's Teach Yourself MySQL in 21 Days," United States of America, Michael Stephens, 2001, p. 532.

[9] Lan, "What is data integrity," 28 05 2016. [Online].

[10] C. Ng, "Data-integrity," 25 05 2018. [Online].

[11] M. A. Poolet, "ITProToday," [Online]. [Accessed 31 october 1999].

[12] "Referential Integrity," Techopedia. [Online].

[13] D. Finestone, "12 ways to reduce data integrity," [Online].

[14] S. b. cooper, "why are entity integrity referential integrity important in a database," [Online].