Logo LUT University

Description automatically generated

**CT60A7650 Database Systems Management**

Project – Spring 2022

Lappeenranta–Lahti University of Technology LUT

Master’s degree in Computational Engineering

April 2022

Student number: 0000275466

Examiner(s): Jiri Musto, D.Sc. (Tech)

Harish Ramesh (Teacher Assistant)

ABSTRACT

Lappeenranta–Lahti University of Technology LUT

Your school: LUT School of Engineering Science

Your degree programme: Computational Engineering

0000275466

**Project \_Spring 2022\_0000275466**

Master’s report

2022

24 pages, 3 figures, 0 tables and 10 appendices

Examiner(s): Jiri Musto, D.Sc. (Tech)

Harish Ramesh (Teacher Assistant)

Keywords: Relational model, Trigger, Partition, Integrity rules, Backup method, Recovery method

SYMBOLS AND ABBREVIATIONS

**Terms**

**Abbreviations**

**Table of contents**

Symbols and abbreviations

[1. Database schema based on the ER model 6](#_Toc101735252)

[2. Integrity rules 6](#_Toc101735253)

[3. Partitioning 6](#_Toc101735254)

[3.1 Partition\_1 6](#_Toc101735255)

[3.2 Partition\_2 7](#_Toc101735256)

[4. Access rights for users 7](#_Toc101735257)

[5. Management of values 7](#_Toc101735258)

[5.1 Define default values 7](#_Toc101735259)

[5.2 Define check constraints 7](#_Toc101735260)

[5.3 Define how to manage NULL values 8](#_Toc101735261)

[6. Triggers and a trigger graph 8](#_Toc101735262)

[6.1 Trigger 1 8](#_Toc101735263)

[6.2 Trigger 2 9](#_Toc101735264)

[6.3 Trigger 3 9](#_Toc101735265)

[7. Security issues and measures 10](#_Toc101735266)

[7.1 SQL Injection 10](#_Toc101735267)

[7.2 Postgres Role Privileges 10](#_Toc101735268)

[7.3 Access 11](#_Toc101735269)

[8. Backup and recovery database 12](#_Toc101735270)

[8.1 Back up method and schedule 12](#_Toc101735271)

[8.2 Three realistic disaster and one extremely unlikely disaster 12](#_Toc101735272)

[8.2.1 Disaster 1. Network disaster 12](#_Toc101735273)

[8.2.2 Disaster 2. Hardware die 13](#_Toc101735274)

[8.2.3 Disaster 3. Computer crash 13](#_Toc101735275)

[8.2.4 Extremely unlikely disaster 13](#_Toc101735276)

[8.3 The recovery method from disasters 13](#_Toc101735277)

[9. Discussion 14](#_Toc101735278)

[Appendix 1. Relational model 15](#_Toc101735279)

[Appendix 2. Partition\_1 16](#_Toc101735280)

[Appendix 3. Partition\_2 17](#_Toc101735281)

[Appendix 4. Access rights for Project1\_users 18](#_Toc101735282)

[Appendix 5. Default Value 19](#_Toc101735283)

[Appendix 6. Check constraints 20](#_Toc101735284)

[Appendix 7. The query code for checking NULL values for the primary key and foreign key. 22](#_Toc101735285)

[Appendix 8. Trigger\_1 23](#_Toc101735286)

[Appendix 9. Trigger\_2 24](#_Toc101735287)

[Appendix 10. Trigger\_3 25](#_Toc101735288)

**Figures**

[Figure 1. Trigger1\_graph 8](#_Toc101735289)

[Figure 2. Trigger2\_graph 9](#_Toc101735290)

[Figure 3. Trigger3\_graph 9](#_Toc101735291)

**Tables**

**No table of figures entries was found.**

1. Database schema based on the ER model

The relational model was created based on the given ER model and the company's organizational structure. Secondly, there will be a total of 10 entities instead of 7 entities due to the essential requirement for linking relations formed based on each many-to-many relationship. The whole process of transforming ER model to a relational model abided strictly by 4 different steps learnt from the previous course.

* Transforming entities
* Transforming attributes
* Transforming relationships
* Transforming abstractions

(The ER model can be found in Appendix 1).

1. Integrity rules

The main reason for setting null for the “on delete” operation for foreign key LID on Customer table is that I want to keep all data relating to customers no matter location was removed so that we are able to analysing customer data or we can still monitor customer relationship in case we move offices to somewhere else. For other table, I always choose CASCADE for “on update” operation so that the database is consistent. For the “on update” operation, I always choose Cascade because I want all associated values in the child table would be also updated when applying operations on parent tables.

1. Partitioning
   1. Partition\_1

The Project table was partitioned into “ranges” defined by the Deadline column, specifically each range will last for 1 quarter which is 3 months. The main reason behind this is to help the user monitor the situation every quarter so that the company can meet the KPI as well as fulfil promises to customers. Moreover, I also implemented sub-partitioning to identify which projects will start and end within only 3 months. (The partition\_1 query can be found in Appendix 2).

* 1. Partition\_2

Due to the fact that the company has various departments in three different offices within one country, therefore, I want to categorize the department table based on each” LID”. (The partition\_2 query can be found in Appendix 3).

1. Access rights for users

From the beginning, an extra link was created from Project to User Group, which is one to one relationship because it would be more convenient to track which user group belong to which project and easier for the admin to grant access rights for users. In this case, I assumed that we already knew the user group for the project, and I just simply created a role for that group only and give them all privileges. The critical reason why I did not choose the “Superuser” parameter is that it is dangerous and should be used only when really needed. A superuser can override all access restrictions within the database, therefore I prefer myself as the only superuser. (The query can be found in Appendix 4).

1. Management of values
   1. Define default values

As the company is located in one country only, therefore, the default value of the Country column was set as ‘Finland’ so that if the company has any plans in extending the operation such as opening one more location. (More information can be found in Appendix 5)

* 1. Define check constraints

In this case, I created check constraints for Department\_name, that only accepts the correct name for each department. The same idea can also be applied to Role\_name or User group name. (Check constraints can be found in Appendix 6)

* 1. Define how to manage NULL values

There are many ways to avoid getting null values such as during creating a table, choosing not null for the primary key and foreign key will be a good option. However, in reality, we will never get all data we want. Therefore, we can use 5 different ways below during creating a query.

* IS NULL and IS NOT NULL operators
* ISNULL() Function
* COALESCE() Function
* Case Expression
* NULLIF() Functio

(The query code of checking NULL values for the primary key and the foreign key can be found in Appendix 7)

1. Triggers and a trigger graph
   1. Trigger 1

The idea behind the trigger 1 is that when user wants to put employee into table emp\_usergroup and before user does that, trigger 1 will help him check whether employee belongs to project or not. If employee belongs to project, then it should be no problem, otherwise, there will be report informing ‘employee not in project’.

A picture containing diagram

Description automatically generated

Figure 1. Trigger1\_graph

* 1. Trigger 2

The idea behind the trigger 2 is that after uses inputs new value into Project table, the new user group data for that project will be created right away and the name of user group will be created based on new project’s name. Of course, the project ID in the user group table will be same as project ID in the project table due to one-to-many relation.

Diagram

Description automatically generated

Figure 2. Trigger2\_graph

* 1. Trigger 3

The idea behind the trigger 3 is that after when the company decides to extend one more location, the department table will automatically have 5 new departments, which are HR, Software, Data, ICT and Customer Support , having same location id as new location id in the location table.

Diagram

Description automatically generated

Figure 3. Trigger3\_graph

1. Security issues and measures
   1. SQL Injection

SQL Injection is always a risk, and it may be even more so in the web application space. It is potentially dangerous to any SQL database, including PostgreSQL.

Unfortunately, there are several types of SQL injections, and they all have the same goal: inserting offensive SQL into queries for execution in the database, which was possibly not intended or designed by the developer.

Here are notable suggestions to reduce the risk of SQL injection:

Primary Defenses:

* Use of Prepared Statements (with Parameterized Queries)
* Use of Stored Procedures
* White List Input Validation
* Escaping All User Supplied Input

Additional Defenses:

* Enforcing Least Privilege
* Performing White List Input Validation as a Secondary Defense
  1. Postgres Role Privileges

It is unquestionably applicable to working in the PostgreSQL environment. Neglect, misunderstanding, or a lack of diligence provide just as many opportunities for attacks and unauthorized use as those launched on purpose. There are privileges that always needs revaluation or reassessment from time to time.

* SUPERUSER
* CREATROLE
* CREATEDB
* GRANT

This mash-up of benefits is definitely worth a look. A DBA would be better served with privileges like SUPERUSER or CREATEROLE in their hands rather than analysts or developers. When it comes to CREATEDB, does the role really need it? What about GRANT privilege? That attribute may be abused in the wrong hands. Carefully consider all the options before allowing these privileges to be applied to roles.

* 1. Access

Having physical access to a server can make it extremely difficult to prevent someone from accessing the data. However, there are a number of measures that can be taken, both physical and technological.

First and foremost, physical access should be limited as much as possible, by assuring that there is located in a secure facility. Depending on if this is a privately owned server room, security measures can be taken to ensure only authorized personnel can enter. Additionally, CCTV surveillance systems can be employed. In the case that a co-location facility is used, ensure that the chosen provider has a strictly enforced security policy appropriately designed to prevent unauthorized access, and in facilities that allow users to enter, that locking racks and cages are available to keep other customers away from your hardware.

**Connecting**

Connecting to a Postgres server can either be done via a Unix domain socket or with a TCP/IP socket

**Unix Domain Socket**

For Unix-like platforms, the Postgres database is connected by means of Unix Domain Sockets (UDS). At this time, they will not be implemented on Windows, but they will be in Postgres v13 and later.

**TCP/IP Socket**

When accessing your Postgres server using pgAdmin from a remote system, as is often the case with applications with multiple tiers or services, you must use a TCP/IP network socket.

**Firewall**

Firewall is an effective tool to prevent access to network ports from unauthorized sources. Many also offer logging facilities which can be used as part of a broader initiative to proactively detect intrusion attempts to help mitigate them before they are actually successful.

1. Backup and recovery database
   1. Back up method and schedule

Weekly full backups can be performed automatically. Daily differential backups are possible. The backup can be scheduled to run at night, during non-working hours of all locations. With the high concurrency, having backup during working hours is not optimal since it would prolong the waiting time for operations. All locations are incrementally backed up every midnight (if different time zones, more consideration should be given). Currently, the database contains only basic information, making it reasonable to have sufficient availability for administration. Subsequently, in arrange to speed up the method of backing up and blending information during the time that there's no activity, at this arrange, a cold reinforcement is more advantageous. Furthermore, exchange log reinforcements can be set to happen each 5 to 10 minutes, so that in the event that there's a calamity happening between two incremental reinforcements, we don't lose everything of the working day before the night comes.

* 1. Three realistic disaster and one extremely unlikely disaster

1. Disaster 1: Network disaster
2. Disaster 2: Hardware die
3. Disaster 3: Computer crash
   1. The recovery method from disasters

* Automated Backup

Reinforcements generally give high confirmation and shields against information misfortune. Reinforcement boosts your RPO as it helps to limit information misfortune when everything goes south. Things you need to consider and get ready for your computerized reinforcement covers your reinforcement apparatus/equipment, reinforcement information overt repetitiveness, security, execution, speed, and information stockpiling. It's additionally good that you could need to detach your reinforcement from your nearby organization or store it in a distant area. An elective methodology is to draw in with outsider suppliers. For example, putting away your reinforcement in the cloud can be a choice, and their office is profoundly complex and fulfils your necessities.

* Highly Available

An exceptionally accessible PostgreSQL bunch in a Hybrid Cloud generally guarantees that your data set correspondence guarantees uptime. The ideal instance of high accessibility relies upon the estimation of your accessibility. For this situation, a typical arrangement for a PostgreSQL conveyed in a half and half cloud can be either your information base facilitated in a public cloud can be your auxiliary bunch going about as your information recuperation group in the event that the essential bunch comes up short or experiences an organization calamity and can take a lot of personal time. In some arrangement, it is conceivable that the optional bunch lying in the public cloud probably won't be by and large as refined as the essential, suppose this is your on-prem or private cloud.

* Cluster control

ClusterControl likewise has a Backup and Restore include that permits clients to deal with your reinforcement, make a reinforcement, plan a reinforcement, and re-establish a reinforcement. Dealing with your reinforcement is extremely clear and making or booking a reinforcement is basic yet likewise offers progressed choices. It likewise offers cloud reinforcement choices that permit you to have reinforcement information overt repetitiveness, fortifying your Disaster Recovery choices.

ClusterControl permits the association or organizations to deal with the data set with adaptability and lessen the arrangement's general intricacy. ClusterControl offers failover, robotized reinforcement, gives a profoundly accessible arrangement, load adjusting, and upholds a conveyed climate sending, making it simpler to add hubs either in a public cloud or secretly or on-prem.

1. Discussion

For this assignment, I want to try everything by using only PgAdmin4 and I realized some pros and cons during my work. For instance, I try to generate ERD by Pgadmin4, but it seems that there are only 2 options for connections which are one-to-many relationships and many-to-many relationships. As a result, it’s hard for me to express what I wanted in designing the relational model.Ddraw.io is still the best app for designing ER model from my point of view and I will continue using it for this assignment.

Appendix 1. Relational model

Graphical user interface

Description automatically generated

Appendix 2. Partition\_1

Text

Description automatically generated

Appendix 3. Partition\_2

Text

Description automatically generated with medium confidence

Appendix 4. Access rights for Project1\_users

Graphical user interface, text, application

Description automatically generated

Appendix 5. Default Value

Table

Description automatically generated

Appendix 6. Check constraints

Graphical user interface, text

Description automatically generated

Appendix 7. The query code for checking NULL values for the primary key and foreign key.

Text

Description automatically generated

Appendix 8. Trigger\_1

Graphical user interface, text, application, email

Description automatically generated

Appendix 9. Trigger\_2

Graphical user interface, text, application

Description automatically generated

Appendix 10. Trigger\_3

Graphical user interface, text, application

Description automatically generated