VIETNAM NATIONAL UNIVERSITY, HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

Database System Lab (CO2014)

Retail Management System

ASSIGNMENT 2 REPORT

Lecturer: Assoc. Prof. Trần Minh Quang

Class : CC01

Students: Lê Đức Cầm - 1952588
Nguyễn Tiến Dương - 1952639
Phạm Mạnh Dũng - 1952633

HO CHI MINH CITY, OCTOBER 2021

Contents

1 Normal Form in DBMS

1.1 Discuss about NF

⋆ What is Normalization ?

Normalization is the process of **minimizing redundancy** from a relation or set of relations. Redundancy in relation may cause insertion, deletion and updation anomalies. So, it helps to minimize the redundancy in relations.

⋆ What is Normal Form ?

Certain rules in database management system design have been developed to better organize tables and minimize anomalies. The stage at which a table is organized is known as its normal form (or a stage of normalization). Normal forms are used to eliminate or reduce redundancy in database tables.

⋆ How can we determine which normal form we have already satisfy ?

Using Keys and Functional Dependencies of a relation certify with certain rules to figure out which normal form we have reached.

⋆ How does Normalization help DB designers determine the best relation schema ?

· It's a formal framework for designer to analyze relation schemas based on their keys and on the functional dependencies among their attributes.
· It consist of a series of normal form tests that can be carried out on individual relation schemas so that the relational database can be normalized to any desired degree.

⋆ What will be changed if we use Normalization ?

The data will be structured in such a way that attributes are grouped with the Primary Key. Attributes that do not directly depend on Primary Key may be extracted to form a new relation.

⋆ What is the purpose of using Normalization ?

Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties.

⋆ Do we need to apply NF strictly in every circumstance ?

Actually, We don't need to normalize to the highest possible normal form (BCNF ,4F, 5F), in an usual situation, achieve the third normal form is good enough.

1.2 Solutions for improving NF

Apply sequentially below normal form from low to high level will basically ensures a well designed database.

⋆ First Normal Form (1NF) and dependency problems.

· A table is said to be in 1NF if the following rules hold:
1. Columns must have single values.
2. Columns must have unique names.
3. Values of a given attribute must be of the same data type.
4. No two records (or rows) can be identical.
⟶ Disallows composite attributes, multivalued attributes, and nested relations

· For example, the figure below show that it is not satisfy the 1NF due to the violation of first and third rule.

ClientID	ClientName	ClientNo (multivalued attribute)
015	John	256-2568, 524-4589
016	Princess	487-5485,451-copy
027	Tom	458-4587 (wrong data type)
028	Claire	478-2689, 265-1486
029	Robert	485-5584, 254-visa

Hình 1: Not satisfy 1NF

· Solution: Create new relation (good option) or Split into new tuples, each of them contain a part of multivalued or composite attribute

⋆ **Second Normal Form (2NF) - solves partial dependency.**

· A table is said to be in 2NF if the following rules satisfied:
1. That relation must be in first normal (1NF) form.
2. All attributes must be fully functionally dependent on the primary key.
⟶ 2NF solves partial dependency problem in 1NF.

· Consider the following figure, to understand about partial dependency. As you see, non-prime attribute COURSE-FEE is dependent on COURSE-NO but COURSE-NO is subset of candidate key {STUD-NO, COURSE-NO} so partial dependency occurs.

STUD_NO	COURSE_NO	COURSE_FEE
1	C1	1000
2	C2	1500
1	C4	2000
4	C3	1000
4	C1	1000
2	C5	2000

Hình 2: Not satisfy 2NF

· Solution: Split into smaller tables.

Table 1

STUD_NO	COURSE_NO
1	C1
2	C2
1	C4
4	C3
4	C1

Hình 3: Satisfy 2NF: table 1

Table 2

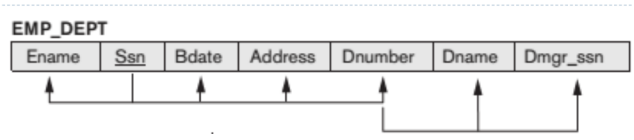
COURSE_NO	COURSE_FEE
C1	1000
C2	1500
C3	1000
C4	2000
C5	2000

Hình 4: Satisfy 2NF: table 2

⋆ **Third Normal Form (3NF) - solves indirect dependency.**

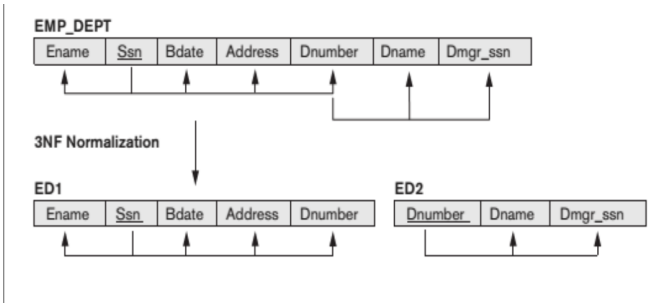
· A table is said to be in 3NF if the bellow rules satisfied:
1. That relation must be in second normal (2NF) form.
2. There is no transitive dependency for non-prime attributes.
⟶ 3NF solves indirect (transitive) dependencies problem in 1NF and 2NF.

· Look at the following figure, we can easily see that SSN -> Dnumber but Dnumber -> Dname and Dnumber -> Dmgr-ssn. So Dname and Dmgr-ssn is transitively dependent on SSN so it violate the rule of 3NF.



Hình 5: Not satisfy 3NF

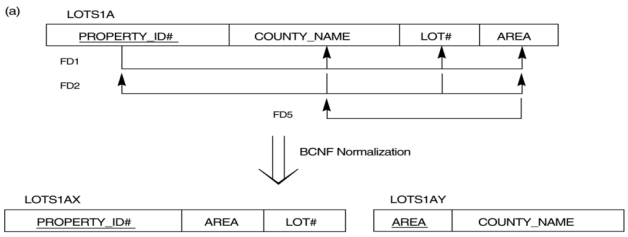
· Solution: identify all transitive dependencies and each transitive dependency will form a new relation, with non-prime attributes participating in the transitive dependency and the attribute which determines others as the attributes for the new relation . And for our example there will be new relation contain Dnumber and its fully functional dependencies



Hình 6: Satisfy 3NF

⋆ Boyce-Codd Normal Form (BCNF) - well-normalized relations

· A table is said to be in BCNF if the following rules satisfied:
1. That relation must be in third normal (3NF) form.
2. if whenever an FD X -> A holds in R, then X is a superkey of R
⟶ In real world database systems it's generally not required to go beyond BCNF.
· Below figure will help you understand better about BCNF



Hình 7: BCNF

⋆ Fourth Normal Form (4NF)

· A table is said to be in 4NF if the following rules satisfied:
1. It should be in the Boyce-Codd Normal Form.
2. The relation should not have any Multi-valued Dependency.

· Below figure will help you an insight about 4NF. First we have the following table

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
2	C#	Cricket
2	Php	Hockey

Hình 8: Not Satisfy 4NF

· If we want to add some more data like below figure, it will be a lots duplication

s_id	course	hobby
1	Science	Cricket
1	Maths	Hockey
1	Science	Hockey
1	Maths	Cricket

Hình 9: Not Satisfy 4NF

· Solution: Decompose into sub table.

s_id	course
1	Science
1	Maths
2	C#
2	Php

Hình 10: Satisfy 4NF: table 1

s_id	hobby
1	Cricket
1	Hockey
2	Cricket
2	Hockey

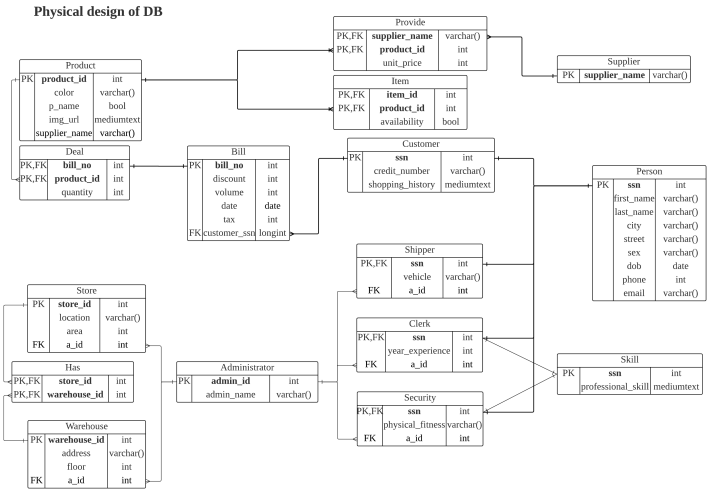
Hình 11: Satisfy 4NF: table 2

⋆ Fifth Normal Form (5NF).

· A table is said to be in 5NF if the following rules satisfied:
1. It should be in the Fourth Normal Form.
2. The relation should not contains any join dependency and joining should be lossless.

1.3 Practical apply to our implemented database

⋆ Step 0: Let see review how our physical DB design looks like.



Hình 12: Physical design of Retail Management System DB

⋆ Step 1: Check if our database satisfy 1NF

As we have already known, using properly mapping technique will always ensure that our DB is in 1NF due to no multivalued nor composite attribute because we've divided them into another relation like 'Skill' table.

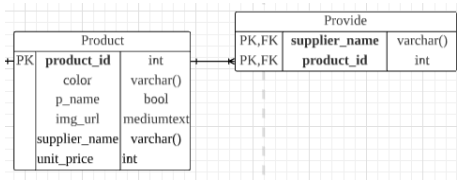
⟶ Now we are in 1NF

⋆ Step 2: Check if our database satisfy 2NF

Since we have already been in 1NF, moreover most of our relations have PK comprises a single attribute so those tables are said to be in 2NF, except the 'Has', 'Deal', 'Provide', 'Item' tables, however only 'Provide' relation have partial dependency problem as a product will have same unit-price despite the supplier change maybe in real world there will be a lot of duplication of unit-price when we have many of suppliers for one product.

Solution: Split into 2 tables like bellow figure. We already have a 'Product' table so it's better to put unit price in that table

⟶ Now we are in 2NF



Hình 13: 2 new tables solving partial dependency

⋆ **Step 3: Check if our database satisfy 3NF**

Currently, we are in 2NF and luckily we have no transitive dependency in our design because we had discussed to eliminate the 'country' attribute in 'Person' table before conducting our conceptual design, so the transitive dependency in 'Person' relation (ssn -> city -> country) will not occur

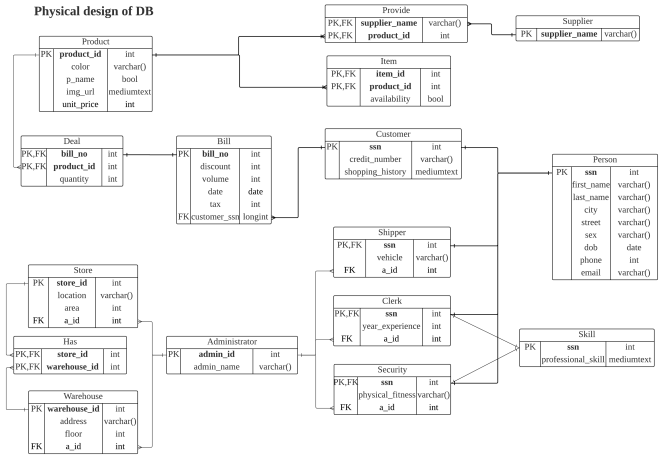
⟶ Now we are in 3NF

⋆ **Step 4: Check if our database satisfy BCNF**

Because in our design, every determinant is a candidate key and we have reached 3NF.

⟶ Now we can confirm we are in BCNF

⋆ Although there are some more normal form but in real world database systems it's generally not required to go beyond BCNF.So we think our current design is acceptable. And you can see our final design bellow

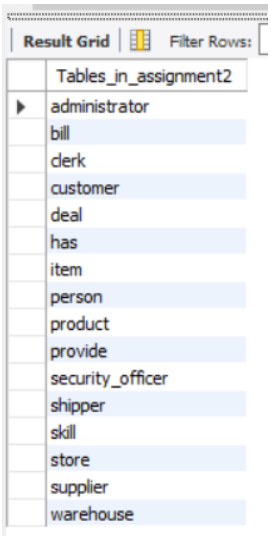


Hình 14: Final design of our DB system

2 Complete implementation of DB into DBMS after normalization

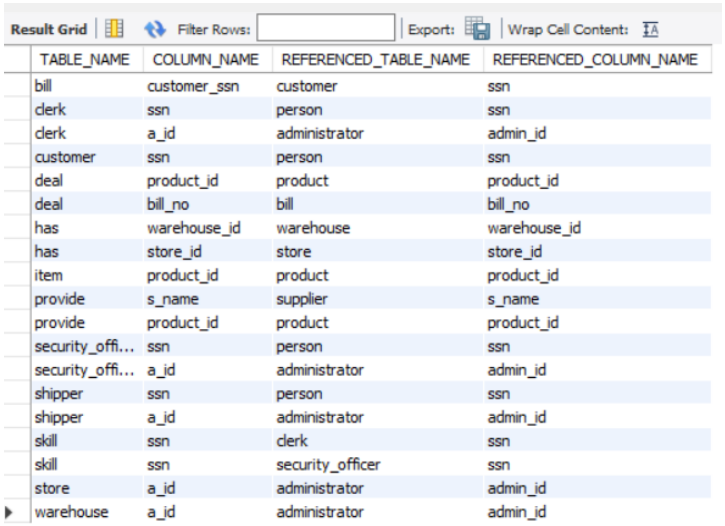
⋆ According to the Assignment 1 report, you can check that we have already completely implemented everything into MySQL DBMS and write some basic command to query, add, delete and update data from the database. So let see a again all relations in our DB.

· Below is all of the tables we have.



Hình 15: All tables in our DB

· And this is all of the relationships between tables in our DB.



TABLE_NAME	COLUMN_NAME	REFERENCED_TABLE_NAME	REFERENCED_COLUMN_NAME
bill	customer_ssn	customer	ssn
clerk	ssn	person	ssn
clerk	a_id	administrator	admin_id
customer	ssn	person	ssn
deal	product_id	product	product_id
deal	bill_no	bill	bill_no
has	warehouse_id	warehouse	warehouse_id
has	store_id	store	store_id
item	product_id	product	product_id
provide	s_name	supplier	s_name
provide	product_id	product	product_id
security_offi...	ssn	person	ssn
security_offi...	a_id	administrator	admin_id
shipper	ssn	person	ssn
shipper	a_id	administrator	admin_id
skill	ssn	clerk	ssn
skill	ssn	security_officer	ssn
store	a_id	administrator	admin_id
warehouse	a_id	administrator	admin_id

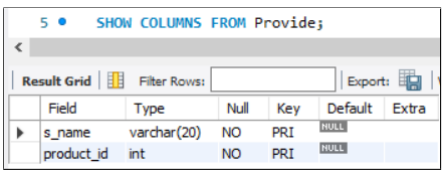
Hình 16: All foreign key constrain in our DB

⋆ As we have change a little bit in our design after normalization however it does not has much impact on our DB because all of the PK and FK constrain stay the same. So we just need to move the 'unit-price' column from 'Provide' table to 'Product' table.

·Now let run some command to make sure our DB is in Boyce-Codd Normal Form.

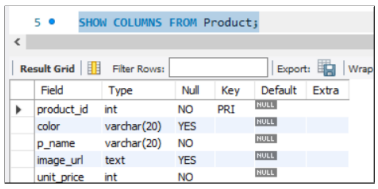
```
ALTER TABLE Provide
    DROP COLUMN unit_price;

ALTER TABLE Product
    ADD unit_price INT;
```

· And check if it is successfully updated or not



Field	Type	Null	Key	Default	Extra
s_name	varchar(20)	NO	PRI	NULL	
product_id	int	NO	PRI	NULL	

Hình 17: New 'Provide' table



Field	Type	Null	Key	Default	Extra
product_id	int	NO	PRI	NULL	
color	varchar(20)	YES		NULL	
p_name	varchar(20)	NO		NULL	
image_url	text	YES		NULL	
unit_price	int	NO		NULL	

Hình 18: New 'Product' table

3 Database security

3.1 Discuss about database security

⋆ What is Database Security ?

Database Security includes a variety of measures used to secure database management systems from malicious cyber-attacks and illegitimate use. Database security programs are designed to protect not only the data within the database, but also the data management system itself, and every application that accesses it, from misuse, damage, and intrusion. Database security refers to the range of tools, controls, and measures designed to establish and preserve database confidentiality, integrity, and availability. Database security must address and protect the following:

- The data in the database
- The database management system (DBMS)
- Any associated applications
- The physical database server and/or the virtual database server and the underlying hardware
- The computing and/or network infrastructure used to access the database

Database security is a complex and challenging endeavor that involves all aspects of information security technologies and practices. It's also naturally at odds with database usability. The more accessible and usable the database, the more vulnerable it is to security threats; the more invulnerable the database is to threats, the more difficult it is to access and use.

⋆ Database Security Threats

Insider Threats An insider threat is a security risk from one of the following three sources, each of which has privileged means of entry to the database: A malicious insider with ill-intent; A negligent person within the organization who exposes the database to attack through careless actions; An outsider who obtains credentials through social engineering or other methods, or gains access to the database's credentials.

Human Error Weak passwords, password sharing, accidental erasure or corruption of data, and other undesirable user behaviors are still the cause of almost half of data breaches reported.

Exploitation of Database Software Vulnerabilities Attackers constantly attempt to isolate and target vulnerabilities in software, and database management software is a highly valuable target. New vulnerabilities are discovered daily, and all open source database management platforms and commercial database software vendors issue security patches regularly.

SQL/NoSQL Injection Attacks A database-specific threat involves the use of arbitrary non-SQL and SQL attack strings into database queries. Typically, these are queries created as an extension of web application forms, or received via HTTP requests.

Buffer Overflow Attacks Buffer overflow takes place when a process tries to write a large amount of data to a fixed-length block of memory, more than it is permitted to hold. Attackers might use the excess data, kept in adjacent memory addresses, as the starting point from which to launch attacks.

Denial of Service (DoS/DDoS) Attacks In a denial of service (DoS) attack, the cybercriminal overwhelms the target service—in this instance the database server—using a large amount of fake requests. The result is that the server cannot carry out genuine requests from actual users, and often crashes or becomes unstable.

Malware Malware is software written to take advantage of vulnerabilities or to cause harm to a database. Malware could arrive through any endpoint device connected to the database's network.

Loss of integrity Database integrity refers to the requirement that information be protected from improper modification. Modification of data includes creating, inserting, and updating data; changing the status of data; and deleting data. Integrity is lost if unauthorized changes are made to the data by either intentional or accidental acts. If the loss of system or data integrity is not corrected, continued use of the contaminated system or corrupted data could result in inaccuracy, fraud, or erroneous decisions.

Loss of availability Database availability refers to making objects available to a human user or a program who/which has a legitimate right to those data objects. Loss of availability occurs when the user or program cannot access these objects.

Loss of confidentiality Database confidentiality refers to the protection of data from unauthorized disclosure. The impact of unauthorized disclosure of confidential information can range from violation of the Data Privacy Act to the jeopardization of national security. Unauthorized, unanticipated, or unintentional disclosure could result in loss of public confidence, embarrassment, or legal action against the organization.

3.2 Solutions and security mechanism

When considering the threats facing databases, it is important to remember that the database management system alone cannot be responsible for maintaining the confidentiality, integrity, and availability of the data. To protect databases against the threats discussed above, it is common to implement four kinds of control measures: access control, inference control, flow control, encryption and some other solutions:

⋆ **Actively Manage Passwords and User Access**

Automating access management via password management or access management software. This will provide permitted users with a short-term password with the rights they need every time they need to gain access to a database.

⋆ **Test Our Database Security**

Once we have put in place our database security infrastructure, we must test it against a real threat. Auditing or performing penetration tests against our own database will help us get into the mindset of a cybercriminal and isolate any vulnerabilities we may have overlooked.

To make sure the test is comprehensive, involve ethical hackers or recognized penetration testing services in the security testing. Penetration testers provide extensive reports listing database vulnerabilities, and it is important to quickly investigate and remediate these vulnerabilities. Run a penetration test on a critical database system at least once per year.

⋆ **Use Real-Time Database Monitoring**

Continually scanning your database for breach attempts increases our security and lets us rapidly react to possible attacks.

⋆ **Use Web Application and Database Firewalls**

We should use a firewall to protect our database server from database security threats. By default, a firewall does not permit access to traffic. It needs to also stop the database from starting outbound connections unless there is a particular reason for doing so.

As well as safeguarding the database with a firewall, we must deploy a web application firewall (WAF). This is because attacks aimed at web applications, including SQL injection, can be used to gain illicit access to the databases.

⋆ **Access control**

A security problem common to computer systems is that of preventing unauthorized persons from accessing the system itself, either to obtain information or to make malicious changes in a portion of the database. The security mechanism of a DBMS must include provisions for restricting access to the database system as a whole. This function, called **access control**, is handled by creating user accounts and passwords to control the login process by the DBMS.

⋆ **Inference control**

Statistical databases are used to provide statistical information or summaries of values based on various criteria. Security for statistical databases must ensure that information about individuals cannot be accessed. It is sometimes possible to deduce or infer certain facts concerning individuals from queries that involve only summary statistics on groups; consequently, this must not be permitted either. This problem, called statistical database security, the corresponding control measures are called inference control measures.

⋆ **Flow control**

Preventing information from flowing in such a way that it reaches unauthorized users. Covert channels are pathways on which information flows implicitly in ways that violate the security policy of an organization.

⋆ **Data encryption**

Protecting sensitive data (such as credit card numbers) that is transmitted via some type of communications network. Encryption can be used to provide additional protection for sensitive portions of a database as well. The data is encoded using some coding algorithm. An unauthorized user who accesses encoded data will have difficulty deciphering it, but authorized users are given decoding or decrypting algorithms (or keys) to decipher the data.

A DBMS typically includes a database security and authorization subsystem that is responsible for ensuring the security of portions of a database against unauthorized access. It is now customary to refer to two types of database security mechanisms:

- Discretionary security mechanisms. These are used to grant privileges to users, including the capability to access specific data files, records, or fields in a specified mode (such as read, insert, delete, or update).
- Mandatory security mechanisms. These are used to enforce multilevel security by classifying the data and users into various security classes (or levels) and then implementing the appropriate security policy of the organization. For example, a typical security policy is to permit users at a certain classification (or clearance) level to see only the data items classified at the user's own (or lower) classification level. An extension of this is role-based security, which enforces policies and privileges based on the concept of organizational roles.

4 Using tool for tuning and manipulating the DB

4.1 Simple to complex query

One of the main feature of DB is retrieving data from database, and with the help of SELECT statement,we can easily get the data that we want.

⋆ This the general syntax of SELECT statement.

```
SELECT column1, column2, ...
    FROM table_name;
    -- or --
SELECT * FROM table_name;
```

⋆ Result of simple SELECT statement.

```
SELECT *
    FROM Administrator;

SELECT admin_id, admin_name
    FROM Administrator;
```

Hình 19: SELECT statement

admin_id	admin_name
1	A
2	B
3	NULL
4	Cam
5	Dung
6	Duong

Hình 20: Result of simple query

⋆ We have 2 more tables below which were added some data, can be use through all examples.

warehouse_id	address	floor	a_id
1	Street1	5	1
2	Street2	1	1
3	Street1	1	2
4	Street2	NULL	4
5	Street4	3	3
6	Street5	2	5
7	Street7	2	6
8	Street2	2	6
9	Street2	1	1
10	Street10	1	1

Hình 21: 'Warehouse' relation

store_id	location	area	a_id
1	Street1	45	1
2	Street2	45	2
3	Street2	47	3
4	Street4	47	4
5	Street10	50	4
6	Street11	52	6
7	Street5	52	4
8	Street1	52	4
9	Street9	55	3
10	Street3	55	2
11	Street6	NULL	2

Hình 22: 'Store' relation

⋆ Let retrieve information from 'Warehouse' and 'Administrator' tables satisfied some condition: id of admin < id of warehouse .

```
SELECT  a_id, admin_name, warehouse_id
FROM  Warehouse
INNER JOIN  Administrator
ON  Warehouse.a_id = Administrator.admin_id
WHERE  admin_id < warehouse_id
    AND admin_name IS NOT NULL ;
```

Hình 23: Query

a_id	admin_name	warehouse_id
1	A	2
1	A	9
1	A	10
2	B	3
5	Dung	6
6	Duong	7
6	Duong	8

Hình 24: Result of query

⋆ In case we want to find the name of admin of the store located in Street1 and their store both use the same warehouse in the same street, we can use this.

```
SELECT location,warehouse_id, store_id, admin_name
FROM Administrator AS A
JOIN (SELECT location, warehouse_id, store_id, a_id
      FROM Store as S
      JOIN (SELECT warehouse_id,address FROM Warehouse
            WHERE Warehouse.address='Street1' AND Warehouse.floor<=3)  AS W
      ON S.Location = W.address) AS SW
      -- select all pair of warehouse with <=3 floor and store in the street1
ON A.admin_id = SW.a_id; -- find name of admin
```

Hình 25: Query

location	warehouse_id	store_id	admin_name
Street1	3	1	A
Street1	3	8	Cam

Hình 26: Result of query

⋆ What if we want to retrieve 3 largest and 3 smallest store(by area) in our DB. We can do something like the following figure

```
SELECT DISTINCT  area
FROM Store s1
WHERE 3 >= (SELECT count(DISTINCT area) FROM Store s2 WHERE s1.area <= s2.area)
    OR 3 >= (SELECT count(DISTINCT area) FROM Store s3 WHERE s1.area >= s3.area)
ORDER BY s1.area DESC;
```

Hình 27: Query

area
55
52
50
47
45
NULL

Hình 28: Result of query

⋆ In the situation we need to count number of warehouse in some particular street but they are not located on the same street with Stores in 1st and 2nd street

```
SELECT count(warehouse_id), address, a_id
FROM warehouse
WHERE (address) NOT IN
    (SELECT location
     FROM Store
     Where location='Street1' OR location='Street2')
GROUP BY address;
```

Hình 29: Query

count(warehouse_id)	address	a_id
1	Street4	3
1	Street5	5
1	Street7	6
1	Street10	1

Hình 30: Result of query

⋆ Consider the case, we want to have a table with all of the address of our store and warehouse but just need to retrieve top 2 of them

```
(SELECT address FROM warehouse)
UNION (SELECT location FROM store)
ORDER BY address DESC LIMIT 2;
```

Hình 31: Query

address
Street9
Street7

Hình 32: Result of query

4.2 Stored Procedures and Trigger

⋆ Stored Procedures

· General syntax of stored procedure:

```
DELIMITER //
CREATE PROCEDURE p0()
BEGIN
  SELECT  * FROM administrator;
END //
DELIMITER ;
CALL p0();
DROP PROCEDURE p0;
```

Hình 33: Query

admin_id	admin_name
1	A
2	B
3	NULL
4	Cam
5	Dung
6	Duong

Hình 34: Result of procedure

· Example of procedure to find which stores are being manage by a particular admin (input is admin-name)

```
DELIMITER //
CREATE PROCEDURE p1
(IN a_name CHAR(20))
BEGIN
  SELECT  admin_name,store_id
  FROM (SELECT admin_name,store_id
        FROM Administrator AS A
        JOIN (SELECT a_id,store_id from store ) as S
        ON A.admin_id = S.a_id)  as T
  WHERE a_name = T.admin_name;
END //
DELIMITER ;
CALL p1('cam');
DROP PROCEDURE p1;
```

Hình 35: Procedure Statement

admin_name	store_id
Cam	4
Cam	5
Cam	7
Cam	8

Hình 36: Result of procedure

· Example of procedure to count number of warehouses on 2 streets (input is 2 street name)

```
DELIMITER //
CREATE PROCEDURE p2(IN street1 varchar(20),street2 varchar(20))
BEGIN
    SELECT count(warehouse_id), address
    FROM warehouse
    WHERE (address) IN
        (SELECT location FROM Store
        Where location=street1 OR location=street2)
    GROUP BY address;
END //
DELIMITER ;
CALL p2('street1','street2');
DROP PROCEDURE p2;
```

Hình 37: Procedure Statement

count(warehouse_id)	address
2	Street1
4	Street2

Hình 38: Result of procedure

⋆ Trigger

· General syntax and a specific output of Trigger:

```
DELIMITER //
CREATE TRIGGER t0 BEFORE UPDATE
ON administrator
FOR EACH ROW
IF NEW.admin_name IS NULL THEN
SIGNAL SQLSTATE '50001' SET MESSAGE_TEXT = 'Admin name should not be null';
END IF; //
DELIMITER ;
```

Hình 39: Trigger Statement

Hình 40: Error when add null value

· We can use trigger to audit lots of actions on an object in our DB:

```
CREATE TRIGGER before_admin_update
BEFORE UPDATE ON Administrator
FOR EACH ROW
INSERT INTO admin_audit
 SET action = 'update',
     old_admin_name = OLD.admin_name,
     new_admin_name = NEW.admin_name,
     changedate = NOW();
drop trigger  before_admin_update;
```

Hình 41: Trigger Statement

```
82 • select * from admin_audit;
```

Result Grid | Filter Rows: | Edit: | Expor

id	old_admin_name	new_admin_name	changedate	action
1	A	1sr change	2021-10-29 11:08:50	update
2	1sr change	2nd change	2021-10-29 11:09:01	update
3	2nd change	3nd change	2021-10-29 11:09:08	update
4	3nd change	A	2021-10-29 11:09:17	update

Hình 42: Audit table

· We can also use trigger to modify invalid input from user before insert something to DB:

```
DELIMITER //
CREATE TRIGGER t2
BEFORE INSERT ON Administrator
FOR EACH ROW
BEGIN
  IF NEW.admin_id <= (select max(admin_id) from administrator)
    THEN SET NEW.admin_id = (select max(admin_id)+1 from administrator);
  END IF;
  IF NEW.admin_name = 'Not_allow_this name'
     OR NEW.admin_name IS NULL THEN
        SET NEW.admin_name ='default name';
   END IF;
END;//
INSERT INTO Administrator Values(1,NULL);
INSERT INTO Administrator Values(1,"Not_allow_this name");
```

Hình 43: Trigger Statement

admin_id	admin_name
1	A
2	B
3	NULL
4	Cam
5	Dung
6	Duong
7	default name
8	default name

Hình 44: Result after add 2 invalid tuples

4.3 Indexing

⋆ Listing/Showing Indexes

Tables can have multiple indexes. Managing indexes will inevitably require being able to list the existing indexes on a table. The syntax and result of viewing an index is like figure below

```
93  SHOW INDEX FROM Store;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type	Comment	Index_comment	Visible	Expression
store	0	PRIMARY	1	store_id	A	11	NULL	NULL		BTREE			YES	NULL
store	1	a_id	1	a_id	A	5	NULL	NULL	YES	BTREE			YES	NULL

Hình 45: Show indexes

⋆ Creating Indexes

Index creation has a simple syntax. The difficulty is in determining what columns need indexing and whether enforcing uniqueness is necessary. Tables can have multiple indexes and it is useful for creating indexes attuned to the queries required by your application or website

```
CREATE INDEX idx ON Store (location);
CREATE INDEX first_idx ON Store (a_id,store_id);
CREATE UNIQUE INDEX second_idx ON Store (store_id,a_id,area);
ALTER TABLE Store ADD FULLTEXT third_idx(location);
```

Hình 46: Sample create index syntax

Table	Non_unique	Key_name	Seq_in_index	Column_name	Collation	Cardinality	Sub_part	Packed	Null	Index_type
store	0	PRIMARY	1	store_id	A	11	NULL	NULL		BTREE
store	0	second_idx	1	store_id	A	11	NULL	NULL		BTREE
store	0	second_idx	2	a_id	A	11	NULL	NULL	YES	BTREE
store	0	second_idx	3	area	A	11	NULL	NULL	YES	BTREE
store	1	first_idx	1	a_id	A	5	NULL	NULL	YES	BTREE
store	1	first_idx	2	store_id	A	11	NULL	NULL		BTREE
store	1	idx	1	location	A	9	NULL	NULL		BTREE
store	1	third_idx	1	location	NULL	11	NULL	NULL		FULLTEXT

Hình 47: Show indexes

⋆ Testing effectiveness of using Indexes

We have create a simple index named 'idx' above, so let test a little bit to see the difference of using indexes by figures below:

```
EXPLAIN SELECT * FROM Store IGNORE INDEX (idx)
    WHERE location= "Street1" AND a_id != 1;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Store	NULL	ALL	first_idx,third_idx	NULL	NULL	NULL	11	11.11	Using where

Hình 48: Query not use index

```
EXPLAIN SELECT * FROM Store USE INDEX (idx)
    WHERE location= "Street1" AND a_id != 1;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	Store	NULL	ref	idx	idx	82	const	2	50.00	Using where

Hình 49: Query using index

⟶ Obvious that using index need less effort to query data, especially in large database.

⋆ Deleting Indexes

While managing indexes, you may find it necessary to remove some. Deleting indexes is also a very simple process, see the example below:

```
DROP INDEX second_idx ON Store;
ALTER TABLE Store
    DROP INDEX third_idx;
```

Hình 50: Sample delete index syntax

Table	Non_unique	Key_name	Seq_in_index	Column_name
store	0	PRIMARY	1	store_id
store	1	first_idx	1	a_id
store	1	first_idx	2	store_id
store	1	idx	1	location

Hình 51: Result after drop 2 indexes

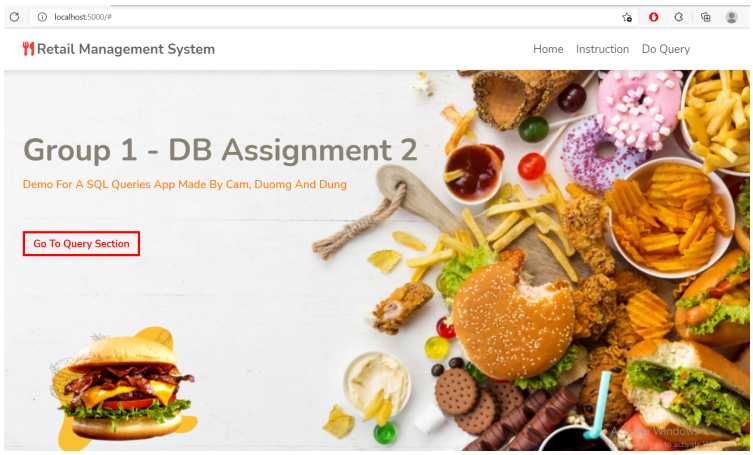
⋆ Conclusion: Despite have some drawback like consuming more memory, slow down the speed of writing queries, such as INSERT, UPDATE and DELETE, Indexing is still a strong and effective technique if you want to retrieve data from a large database. So based on a specific situation, we will choose the best one that suitable for our DB and demand

5 Developing simple app on top of the designed DB system

5.1 Client Side

⋆ Home section

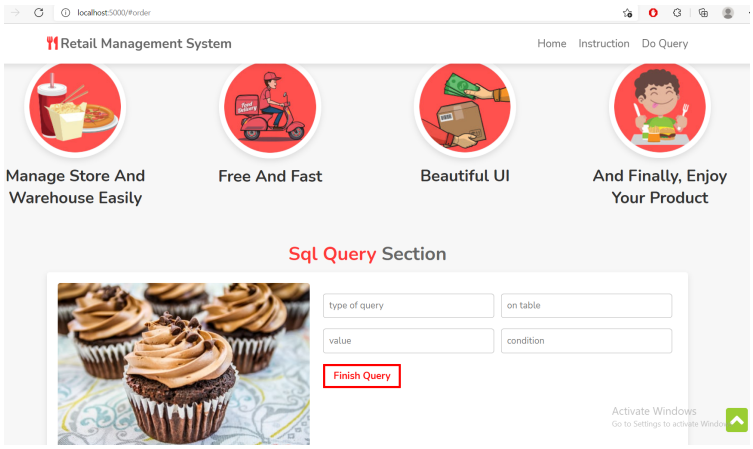
App contain some basic features to help user to navigate to the section they need.



Hình 52: Home section

⋆ Doing query section

This section will help user to manipulate and do some queries on the data in our DB via a provided form.



Hình 53: Query section

5.2 Server Side

⋆ Set up connection

Prepare some packets needed for our app, and connect to MySQL vs Client-side

Hình 54: Set up

```
var con = mysql.createConnection({
        host: "localhost",
        port: "3306",
        user: "root",
        password: "Camcam270301",
        database: "assignment2"
});
con.connect(function(err) {
        if (err) throw err;
        console.log("DB Connected!");
});
var query,table,value,condition;
var sql = "";
```

Hình 55: Connect to MySQL

```
function doOperation() {
        if (query=="INSERT"){ ···
        }
        else if(query=="SELECT"){ ···
        }
        else if(query=="DELETE"){ ···
        }
        else if(query=="UPDATE"){ ···
        };
};
io.on('connection',(socket) =>{
        console.log('user connected');
        socket.on('query',data =>{
            query = data.message;
            console.log(query);
        });
        socket.on('table',data =>{
                table = data.message;
                console.log(table);
        });
        socket.on('value',data =>{
                value = data.message;
                console.log(value);
        });
        socket.on('condition',data =>{
                condition = data.message;
                console.log(condition);
                doOperation();
        });
})
```

Hình 56: Function manage input and queries

5.3 Test and check our database

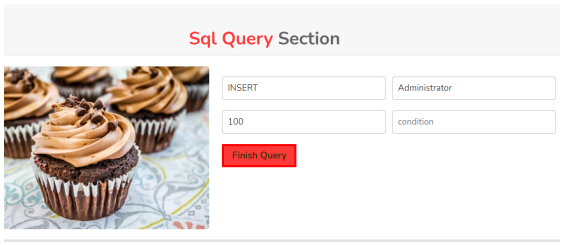
⋆ Run server and see whether it work well or not

```
PS C:\Users\HP\Desktop\dbapp> yarn start
yarn run v1.22.11
$ nodemon index.js
[nodemon] 2.0.14
[nodemon] to restart at any time, enter `rs`
[nodemon] watching path(s): *.*
[nodemon] watching extensions: js,mjs,json
[nodemon] starting `node index.js`
listening on port 5000
DB Connected!
user connected
```

Hình 57: Run server and check status

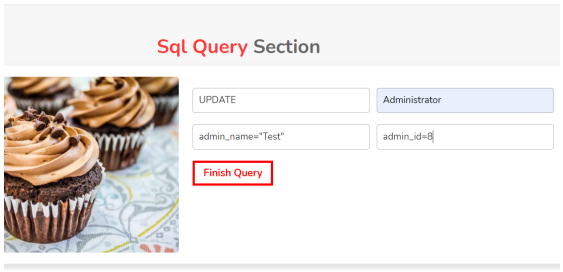
⋆ Do some simple queries

· INSERT



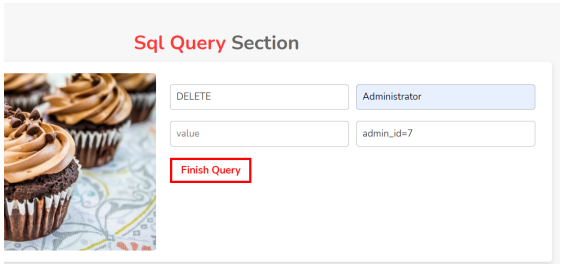
Hình 58: INSERT Operation

· UPDATE



Hình 59: UPDATE Operation

· DELETE



Hình 60: DELETE Operation

· Print out some information about query in server and check if our DB and app works well or not

```
listening on port 5000
DB Connected!
user connected
INSERT
Administrator
100

1 rows inserted on DB
UPDATE
Administrator
admin_name="test"
admin_id=8
1 New update on Administrator
DELETE
Administrator

admin_id=7
```

Hình 61: Server side

admin_id	admin_name
1	A
2	B
3	NULL
4	Cam
5	Dung
6	Duong
7	default name
8	default name
NULL	NULL

Hình 62: Before doing operation

admin_id	admin_name
1	A
2	B
3	NULL
4	Cam
5	Dung
6	Duong
8	test
100	default name

Hình 63: After doing operation

6 Conclusion

For this assignment, we have examined the Retail Management System and tried to build a database for it using MySQL. Regarding the requirements, the biggest difficulty we faced was the simplicity of the retail management system we have constructed, because actually we have no experience in terms of retailing or management in real life. Moreover, the main purpose of a retail management system is to record as much in formation from customers, employees, product, suppliers ... but those things seem to be very abstract for us- who are CS students. We also have gone through database designing, implementing and doing queries steps, from conceptual design, ERD mapping ,logical design, physical design to implementation and doing operations on implemented-database, that's why we have learnt a lots. Although, there are many things more to be learnt but via this assignment, we have already had a big picture of how things work and we confidently that we can do better in case of constructing a new database for other systems in practice later.

7 Reference

1. https://www.imperva.com/learn/data-security/database-security/
2. https://dev.mysql.com/doc/
3. https://www.w3schools.com/sql/
4. https://www.sqlshack.com/
5. https://stackoverflow.com/
6. https://www.geeksforgeeks.org/
7. https://www.tutorialspoint.com/
8. https://www.guru99.com/
9. https://database.guide/