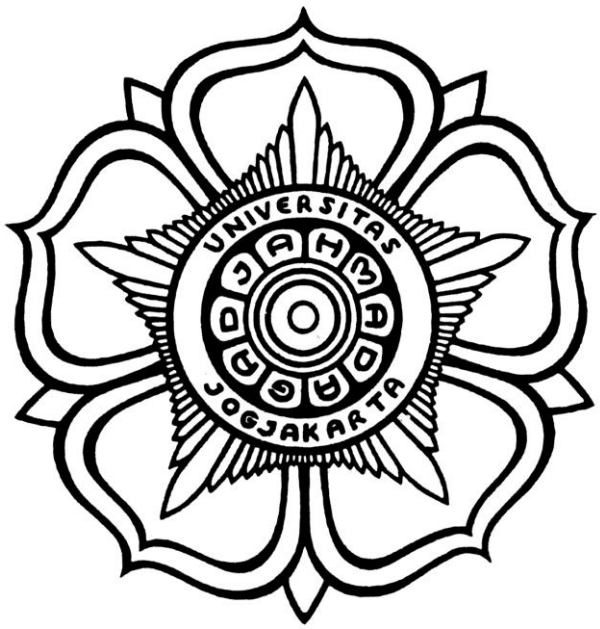
PRACTICAL HANDBOOK

DATABASE

(MII2502)



BASIC LABORATORY OF COMPUTER

Department of Computer Science and Electronics
Faculty of Mathematics and Natural Sciences
Universitas Gadjah Mada
Yogyakarta
2017

OFFICIAL RULES

1. The students should enter the class according to schedule agreement, if more than 15 minutes late, there is no permission to enter the class.
2. The students whohas been morethan3times absence-without-noticeare notpermitted to participate infinal examination.
3. There is nosubstitute formidterm or final examination, exceptif there is an agreement from lectures.
4. The students must wear appropriate clothes:
   - **Boys :**
     - Long or shortsleeveshirt
     - Trousers
     - Must wear shoes
     - Collarless T-shirt or sleeveless shirt not allowed
   - **Girls :**
     - Long or shortsleeveshirt (not tight and transparent)
     - Skirts OR trousers (not tight and transparent)
     - Must wear shoes
     - Collarless T-shirt or sleeveless shirt not allowed
5. No food, no drink, or use of cigarette in any form is allowed in the class.
6. Bags andotherequipmentshould be placed in a provided place (the students only allowed to carry valuable belongings, e.g.: mobile phone, wallet, laptop, andtools related to activities).
7. The students are responsible for their own belongings(the *laboratory* will *not* be *responsible* for *loss*of data, damage, *loss* of personal items).
8. **Turn off cell phones.** If you need to use it, please take it to the hallway.
9. The students must keep the room clean and also conducive learningconditions.
10. The students should maintain professional and courteous communication. Electronic devices should be used on a professional level. No obnoxious or belligerent behaviour will be tolerated.
11. Software may be installed by Computer Labs staff only. Do not install any software on your own. Files not put on by Computer Labs staff will be routinely removed.
12. Please operate the equipment with respect and care.
13. The students should fill the attendance list form and write seats/computers number used during the class.
14. For any hardware, software, problems, please contact an aide.
15. The lab aides are here to help when they can and to maintain the labs' operation. However, the lab aides are not here to do your work for you. The lab aides will refer students to their instructors at the lab aides discretion.
16. The students should turn off the computers and place chairs, tables, and other equipment into a good form after finishing class.
17. Any failure to follow these lab rules may result in the loss of your lab privileges.

BASIC LABORATORY OF COMPUTER
DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
UNIVERISTAS GADJAH MADA

PREFACE

Pratical activities in the laboratory is an instructional activities to help students understand the theory courses are guided by an instructor and supervised by a lecturer. This practicum will focus it has standard guidelines, which are able to handle students and instructors to manage class laboratory.

This practical guide book is a revision of a previous book, with additional material, exercise, and assignments are adjusted.

Hopefully, this book useful for learning the progress of computer science and improving the quality of learning.

.

BASIC LABORATORY OF COMPUTER
DEPARTMENT OF COMPUTER SCIENCE AND ELECTRONICS
FACULTY OF MATHEMATICS AND NATURAL SCIENCES
UNIVERISTAS GADJAH MADA

TABLE OF CONTENTS

CHAPTER I
ENTITY RELATIONSHIP

1.1 LEARNING OUTCOMES

1. Students understand ERD symbols.
2. Students able to draw ERD based on given cases.

1.2 THEORY

In ERD, there are three important things, those are: entitiy set, atribute and relationship.

1.2.1 Entity Set

Entity set is an object that exists in the real world that is different from other objects, have the attributes and as the builder of the system. For example, a man who works in a company is an entity. Entity has an attribute-value (values), for example: number 000-11-3452 is an id number of a worker. In addition, a worker also has the date of birth. Here, the birth date is an attribute of a worker that's parallel to the id number of workers.

Entity has couple of attribute types, those are:

1. Simple attribute: attribute with a simple value, cannot be decomposed into more specific attribute.

2. Composite attribute: entity whose attributes can be divided into smaller attribute. For example: attribute name can be divided into first name, middle name, and the last name.

3. Single-valued attribute: entity whose attributes can contain only one value. Eg. Worker's id number.

4. Multi-valued attribute: entity whose attributes can contain zero, one or more than one value. For example: telephone number, it could be a worker has one or more telephone number.

5. Derivative Attributes: entity whose attributes can be derived from other attributes. For example: age, its value can be derived from date of birth and current date.

1.2.2. Attribute

Attributes are descriptions that describe an entity, such as NIM, name, faculty, department for student entity.

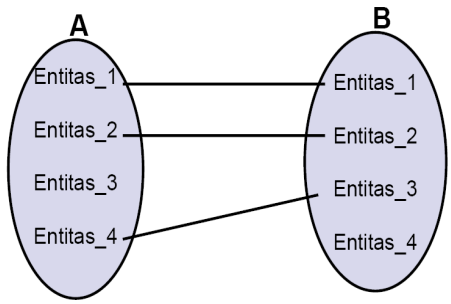
1.2.3 Relationship

Relationship explains relationship between the entities. For example in the above example the human entity has a relationship with the entity address is "lived in". In designing a database, an entity should have a relationship with another entity, at least with one entity. If there's an entity in the database that does not have a relationship with any other entity, then there will be errors in the design. Usually unrelated entity will be eliminated.

There are four kinds of relationship among entities:

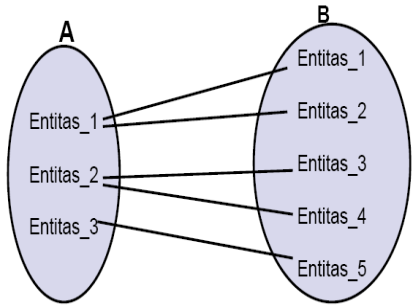
1. *One to one*

   Each entity in the entity set A is related to the most with a single entity in the entity set B, and vice versa every entity in the entity set B is related most one entity in the entity set A.



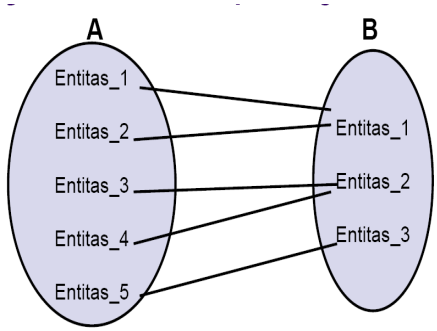
2. *One to many*

   Each entity in the entity set A relate with many entities in entity set B, but not vice versa, where each entity in the entity set B relate most with a single entity in the entity set A.



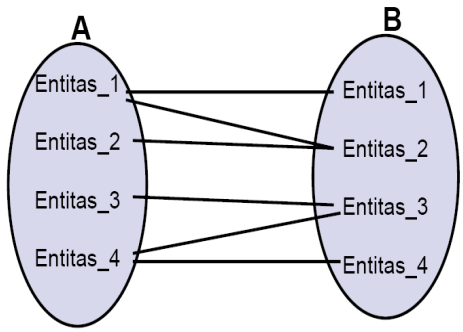
3. *Many to one*

   Each entity in the entity set A related at most with one entity in the entity set B, but not vice versa, where each entity in the entity set B relate to a number of entities in the entity set A.



4. *Many to many*

   Each entity in the entity set A can relate with many entities in entity set B, and vice versa, where each entity in the entity and related himpunen many entities in the entity set A.



1.2.4. *Key*

1.2.4.1 Primary Key

As mentioned above, that entity is a different object with another object. Then each entity must have attributes that can distinguish between objects with each other. For example human entity, the entity doesn't have an attribute to distinguish between objects manusia1 with manusia2 object. However, if the human being is an employee then the object manusia1 and manusia2 as a member of the employee entity will have a distinguishing attribute, namely NIP (workers ID number). That attribute is called the key attribute. Manusia1 and manusia2 must not have the same NIP.

Key may consist of several attributes, commonly called a candidate key. However, in practice it's almost an obligatory that attribute key consists of only one attribute. This is to facilitate the processing of data. To get the key from the candidate key (a combination of several attributes), select the most unique attributes of the entity (the same value in the attribute does not exist) and can be distinguished between an entity member to another entity member. If you've got it, the key is what is called a primary key.

Suppose there is a candidate key form of the name, NIM, and date of birth. It needs to be sorted, which is the most unique attribute of these three attributes. Attribute name is not unique; because there might be two entities worker have identical names. Similarly, the attribute date of birth, there could be two entities worker have the same birth date. Only NIP attributes are certainly different from each entity worker.

1.2.4.2 *Foreign Key*

We have already seen that the entity in the database must have a relationship with other entities. Therefore, an entity that has a relationship with another entity should load the primary key of the entity that there is a relationship with him. For example, an Employee entity has NIP as its primary key. This entity has a relationship with the entity with primary key part of the work, the working section ID. To be able to connect the two entities, the entity need not enter the working part of the primary key of the entity worker into one of its attributes. But it was the workers who enter the entity's primary key of the entity part of the work (employment section ID).The process holds the primary key of another entity to the attributes of the entity is also called the foreign key.

There is some basis for the making the primary key of an entity to another entity related. This premise is wearing characteristic relationships used.

1. *One to one*: A related entity to entity B is one to one, then the primary key of the entity A is loaded into the entity B or vice versa.
2. *Many to one*: A related entity to entity B is many to one, and then the primary key of the entity B is loaded into the entity A.
3. *Many to many*: A related entity to entity B is many to many, the loading of the primary key of each entity will involve a new entity.

1.2.5. Entity-Relationship Diagram

E-R diagram is used to create a database model. Then they invented the system of the model database. As for the various components in the E-R diagram is:

1. The rectangles: represent entity
2. Ellipses: represent attributes
3. Rhombus: represent relationships between entities
4. Line: represents the link between attributes
5. Double Ellipse: represent attributes that have a lot of value
6. Ellipses dashed: represent attributes derivatives
7. Double Lines: represents the total participation of an entity in the relationship
8. Double rectangles: weak entities

There are two types of entities; the first is a strong entity is an entity that has a primary key. The second is a weak entity is an entity that does not have a primary key. Strong and weak entity will be discussed more in the matter of normalization.

1.2.5.1 E-R Diagram Example

At every airport already have schedules for all airlines flying both domestic and international flights. Where the airport had to reschedule to the aircraft and flight service both departures and arrivals. Officer to manage the air traffic is commonly called air traffic controllers (ATC) has an important role especially on the aircraft to prevent an aircraft is not too close to each other, to prevent collisions between aircraft and aircraft with obstacles in the vicinity during operation , Additionally ATC has a role in regulating the smooth flow of traffic, helping the pilot in control of a state of emergency, giving the information needed pilots (such as weather information, flight navigation information, and air traffic information). ATC is the closest associates during a pilot in the air, ATC huge role in the achievement of the flight. In flight service system has several services offered via the web including the following:

1. The airport can see the current weather situation, good or bad to do perbangan, flight status (landing, takeoff, delay), arrivals, flight and flight destination.
2. These can see the task schedule for flights from and to where, who pilots, flight attendants and crew-crew.
3. Agent can sell, book, view prices of planes and flight schedule.
4. Customer can see flight schedule, flight route, see ticket prices, and book tickets.

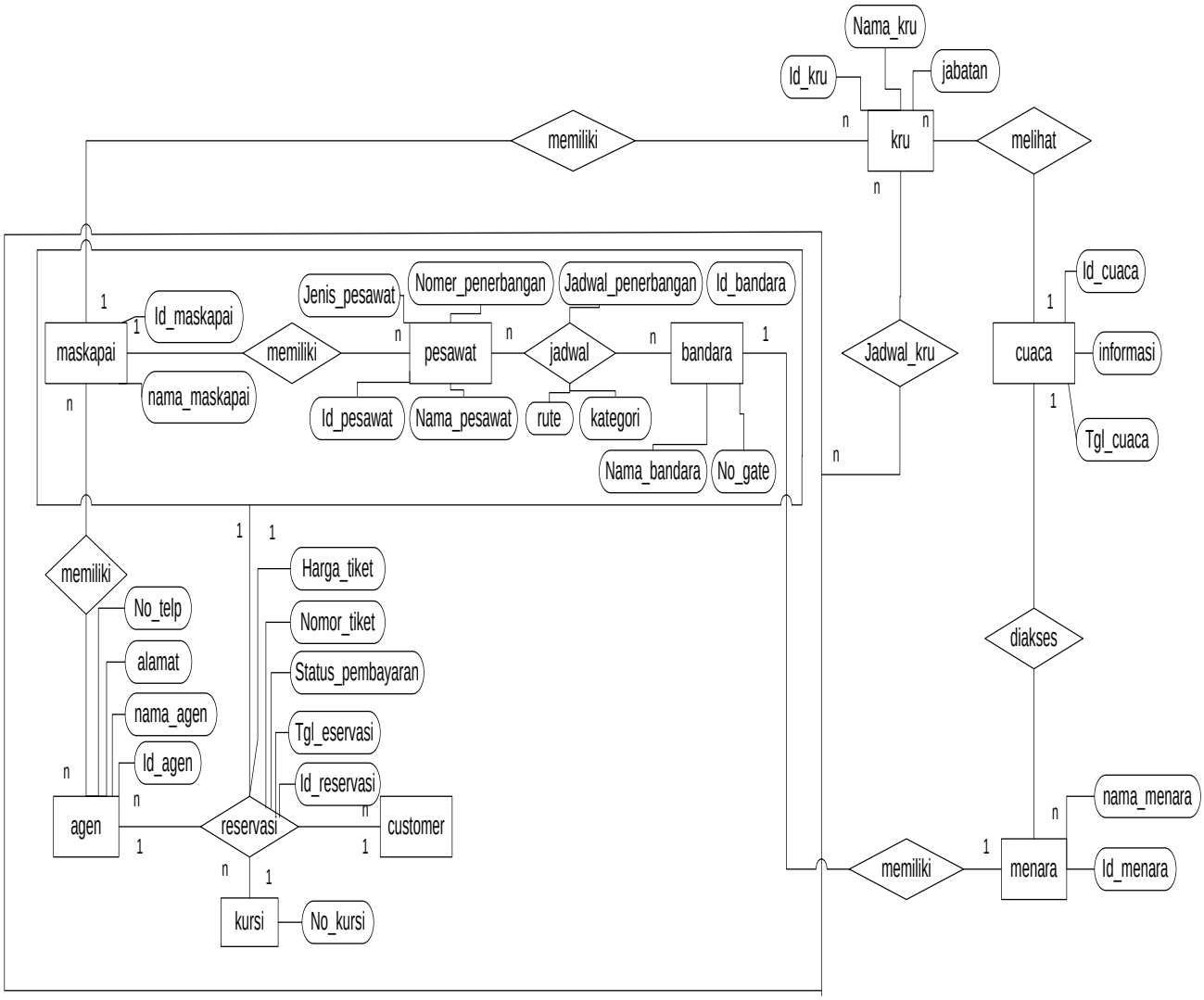


Fig. 1.1 Flight Information System ERD

1.3 STUDENTS ACTIVITY

1. Students learn and try to make ER Diagram on Fig. 1.1.
2. Students do exercise 1.4.

1.4 EXERCISE

Make a database model for library system. Specify all the entities involved and its attributes. Specify which entity is the *primary key*, strong or weak entity, and relationships of the entities. After that, draw the model in ER diagram!

CHAPTER II
NORMALIZATION

2.1 LEARNING OUTCOMES

1. Students are able to understand the urgency of normalization in database tables.
2. Students understand unnormalized form.
3. Students understand 1NF, 2NF and 3NF.

2.2 THEORY

2.1.1 Normalization

The process of normalization is a technique in the logical design of a database that classifies attribute of a relation so as to form a good relationship structure (without redundancy), so that most of the ambiguity can be eliminated. The purpose of normalization is to eliminate the data redundancy (redundancy), reduce complexity and weeks to facilitate modification of data.

Additionally normalization is needed to avoid anomalous insertion, deletion anomalies, and update anomalies. In a database that is already normal, insertion anomaly can be avoided because filling the same data on multiple tables just do enough to fill in one table and the rest will follow, as well as update and delete commands. But the tables were not normal, charging the same data at multiple tables is done one by one. This allows for input by human error (human error) so that some records which should have the same data, will have different data.

Forms of normalization there are several kinds, ranging from the normalization 1, normalization 2, and so on. Some of these cases reach the stage of normalization to more than 5. But there is also the case that only reached the stage of normalization 2 only. Normalization each case is different. It is tailored to the needs. If the database normalization is sufficient to two terms above, it is not necessary next stage of normalization.

2.1.2 Non-Normalization

There are some things to consider before starting the normalization process is carried out, namely:

1. Every label has to have a *primary key.*
2. A label cannot contain redundant data.

Figure 2.1 shows an example of a table **Orders** in the form of a non-normalized table, but has been qualified to do the normalization process.

Orders

OrderId	CustomerId	OrderDate	Items
1	4	5/1/2001	5 32-hammer, 3 2-screwdriver, 6 76-monkey wrench
2	23	5/9/2001	1 32-hammer
3	15	7/4/2001	2 113-deluxe garden hose, 2 121-economy nozzle
4	2	8/1/2001	15 1024-hack saw
5	23	8/2/2001	1 2-screwdriver
6	2	8/2/2001	5 52-key

Fig. 2.1 Orders table - non normalized

From Figure 2.1 it can be seen that the orders table does not have a data set that is repetitive, but has a recurring set of values, namely on the field items. On the field there are 2 items problems, that this column is a multi-part field, the field can be broken down into a unique piece, then multi-value, meaning that 1 OrderId can be associated with one or more field items. Therefore, the next process is to transform into first normal form.

2.1.3 Normalisation I

First normal form has these following rules:

- There's no multi-valued attribute, composite attribute, or its combination.
- Key attributes are defined.
- Every attribute in table has an *atomic* value.

From Figure 2.1 for normalization in the Orders table is done by eliminating multi value of field items.

Orders

OrderId	OrderDate	CustomerId	Item1	Qty1	Price1	Total1	Item2	Qty2	Price2	Total2	Item3	Qty3	Price3	Total3
1	5/1/2001	4	32-hammer	5	$12.99	$64.95	2-screwdriver	8	$7.99	$63.92	76-monkey wrench	6	$8.99	$53.94
2	5/9/2001	23	32-hammer	1	$12.99	$12.99								
3	7/4/2001	15	113-deluxe garden hose	2	$4.50	$9.00	121-economy nozzle	2	$3.85	$7.70				
4	8/1/2001	2	1024-hack saw	15	$17.00	$255.00								
5	8/2/2001	23	2-screwdriver	1	$7.99	$7.99								
6	8/2/2001	2	52-key	5	$1.75	$8.75								

Fig. 2.2 First step in transforming into first normal form

From Figure 2.2 it can be seen that we now have a set of repeating fields: Item1, Quant1, Price1, Item2, Quant2, Price2, and Item3, Quant3, Price3. Furthermore, to overcome performed by dividing into three different areas, being: Item, Quant and Price. In addition, we must remove the multipart characteristics of the item field by dividing it into two distinct areas: ProductID and Product. Figure 2.3 shows the result of the modification.

Orders

OrderId	OrderDate	CustomerId	ProductId	Product	Quantity	Price	Total
1	5/1/2001	4	32	hammer	5	$12.99	$64.95
1	5/1/2001	4	2	screwdriver	8	$7.99	$63.92
1	5/1/2001	4	76	monkey wrench	6	$8.99	$53.94
2	5/9/2001	23	32	hammer	1	$12.99	$12.99
3	7/4/2001	15	113	deluxe garden hose	2	$4.50	$9.00
3	7/4/2001	15	121	ecomomy nozzle	2	$3.85	$7.70
4	8/1/2001	2	1024	hack saw	15	$17.00	$255.00
5	8/2/2001	23	2	screwdriver	1	$7.99	$7.99
6	8/2/2001	2	52	key	5	$1.75	$8.75

Fig. 2.3. First normal form of Orders

Although the shape of the table orders is still not perfect, but the list is already at normal conditions can be tested in the first and second normal form.

2.1.4 Normalization II

First normal form has these following rules:

- Normal form 2NF met in a table if it complies with 1NF form, and all the attributes other than the primary key, as a whole has a Functional Dependency on primary key.
- A table does not meet the 2NF, if there are attributes that dependence (Functional Dependency) only partial (only depending on the part of the primary key).
- If there are attributes that have no dependency on the primary key, then the attributes must be moved or removed.

The table Orders in Figure 2.3 does not pass the required form of second normalization, because there are three different issues, namely:

1. Tabel Orders memiliki 2 subjek, yaitu Orders dan OrderDetails.
2. Tabel Orders mengandung *field* dengan proses perhitungan (Total), dan
3. Tabel Orders memiliki ketergantunagan parsial antara OrderID dan Product.

The first step is to divide the tables into two smaller tables, namely Tables Orders and OrderDetails. At this stage mamstikan that the Orders table only explain about one and only one subject.

Orders

OrderId	CustomerId	OrderDate
1	1	5/1/2001
2	3	5/9/2001
3	1	7/4/2001
4	2	8/1/2001
5	1	8/2/2001
6	2	8/2/2001

OrderDetails

OrderId	ProductId	Product	Quantity	Price	Total
1	32	hammer	5	$12.99	$64.95
1	2	screwdriver	8	$7.99	$63.92
1	76	monkey wrench	6	$8.99	$53.94
2	32	hammer	1	$12.99	$12.99
3	113	deluxe garden hose	2	$4.50	$9.00
3	121	ecomomy nozzle	2	$3.85	$7.70
4	1024	hack saw	15	$17.00	$255.00
5	2	screwdriver	1	$7.99	$7.99
6	52	key	5	$1.75	$8.75

Fig 2.4 Applying second normal form to table Orders

Now the Orders table has been in second normal form, which need to be considered now is OrderDetails table, because at this table are calculated fields and partial dependence. For a field that contains the calculation process can be resolved by deleting it from the table, whereas to overcome partial dependence that can be done is to divide it into two subjects, namely OrderDetails and Product.

To overcome partial dependence done by removing the ProductID field of OrderDetails, and create a new table is a table with ProductID Product and Product is the content of field. ProductID is very important in the Product table, because ProductId will relate the Product table with OrderDetails. Once this is done, table OrderDetails already are on the second normal form.

OrderDetails

OrderId	ProductId	Quantity	Price
1	32	5	$12.99
1	2	8	$7.99
1	76	6	$8.99
2	32	1	$12.99
3	113	2	$4.50
3	121	2	$3.85
4	1024	15	$17.00
5	2	1	$7.99
6	52	5	$1.75

Products

ProductId	Product	Price
2	screwdriver	$7.99
32	hammer	$12.99
52	key	$1.75
113	deluxe garden hose	$4.50
121	ecomomy nozzle	$3.85
1024	hack saw	$17.00
76	monkey wrench	$8.99

Fig. 2.5 Second normal form of OrderDetails and Product

2.1.5 Normalization III

The third normal form has these following characteristics:

- Each value in the field will be updated independently. If you change the value of a field from the data given, then the change is not giving adverse effect on the value of another field.
- Each field identifies certain characteristics of the subject table.
- Each non-key fields in the table are functionally dependent (functional dependency) on the primary key.
- The table only describes one and only one subject.

Orders table and OrderDetail have had second normal form, then the normal processes applied to third in the table. In the Orders table, the requirements of the third normal form has been fulfilled, the Orders table has been in third normal form.

Orders

OrderId	CustomerId	OrderDate
1	1	5/1/2001
2	3	5/9/2001
3	1	7/4/2001
4	2	8/1/2001
5	1	8/2/2001
6	2	8/2/2001

Fig. 2.6 Third normal form of table Orders

If we look at the OrderDetails table in Figure 2.5, can be brought on the table there is a field that is not clear on the subject of the table. This field is a field Price. Price does not represent the specific characteristics of the table OrderDetails, so the value of the price is determined solely by ProductID. In the table OrderDetails have a combined primary key consisting of OrderID and ProductID, but the field Price is not relying solely on the entire primary key, then the Price field can be removed from the table OrderDetails.

OrderDetails

OrderId	ProductId	Quantity
1	32	5
1	2	8
1	76	6
2	32	1
3	113	2
3	121	2
4	1024	15
5	2	1
6	52	5

Fig. 2.7 Third normal form of table OrderDetails

2.3 STUDENTS ACTIVITY

1. Students pay attention to the instructor explanation regarding normalization.
2. Students try and learn the form of non-normalized tables in Figure 2.1.
3. Students try to normalize the table into its first form shown in Figure 2.2 and Figure 2.3.
4. Students try to normalize the table into its second form shown in Figure 2.4 and Figure 2.5.
5. Students try to normalize the table into its second form shown in Figure 2.6 and Figure 2.7.
6. Students practice / assignment to train the understanding of normalization

2.4 EXERCISE

HEALTH HISTORY REPORT

PET ID	PET NAME	PET TYPE	PET AGE	OWNER	VISIT DATE	PROCEDURE
246	ROVER	DOG	12	SAM COOK	JAN 13/2002	01 - RABIES VACCINATION
					MAR 27/2002	10 - EXAMINE and TREAT WOUND
					APR 02/2002	05 - HEART WORM TEST
298	SPOT	DOG	2	TERRY KIM	JAN 21/2002	08 - TETANUS VACCINATION
					MAR 10/2002	05 - HEART WORM TEST
341	MORRIS	CAT	4	SAM COOK	JAN 23/2001	01 - RABIES VACCINATION
					JAN 13/2002	01 - RABIES VACCINATION
519	TWEEDY	BIRD	2	TERRY KIM	APR 30/2002	20 - ANNUAL CHECK UP
					APR 30/2002	12 - EYE WASH

From these cases, do the normalization by explaining each step until the third normal form. Give detailed reasons, such as what attribute being removed for its redundancy, etc.

CHAPTER III
DATA DEFINITION LANGUAGE (DDL)

3.1 LEARNING OUTCOMES

1. Students can explain the concept of Data Definition Language.
2. Students can use the commands command - basic command to define an object from the database.

3.2 THEORY

Data Definition Language (DDL) is a group of commands used to perform database and table definition. Basic DDL commands are:

1. CREATE : this command is used to create databases, tables, view and index.
2. ALTER : this command is used to alter table structure.
3. DROP : this command is used to delete databases, tables, view and index from the database.

3.2.1 Database

The command to create a database that is:

```
CREATE DATABASE databasename;
```

This command is used first before creating a table, view, or other components in the system database. The database name can consist of a combination of letters and characters but should not contain spaces and punctuation to facilitate future database settings. Example:

```
CREATE DATABASE car_service;
```

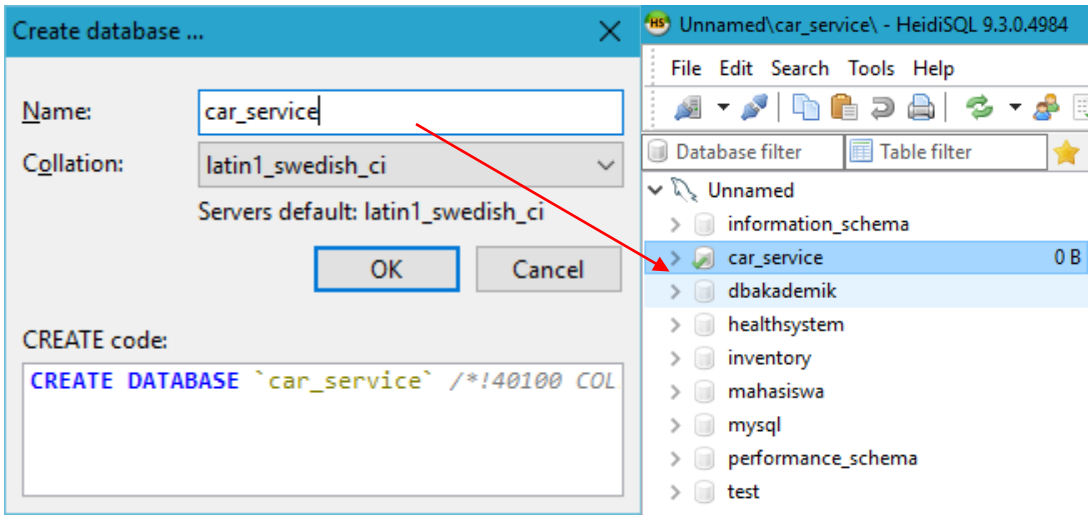


Fig. 4.1 Create database car_service

While the order to eliminate the database, namely:

```
DROP DATABASE databasename;
```

Sample query:

```
DROP DATABASE car_service;
```

To be able to manage a database, a user must first be entered into a database environment that will be processed. To be able to process a database syntax used is

```
USE nama_database;
```

3.2.2 Table

General command to create a table in a database is:

```
CREATE TABLE namatabel (
Field1 TipeData1(jumlahdigit)[keterangan primary key
atau not null jika diperlukan],
Field2 TipeData2 (jumlahdigit)[keterangan primary
key atau not null jika diperlukan],
Field3....);
```

Or in more detail:

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS]tbl_name
(create_definition,...)
[table_option ...]
[partition options]
```

Field1 is an attribute that became the name of the first column, Field2 is the name of the second column, and so on a number of columns to be defined on a table. Tipedata Tipedata1 is used to define the type of data used in the column in question, while jumlahdigit is the number of characters allocated for these data types. However, there is also a declaration of data types that do not need to be declared jumlahdigit her because it is automatically defined by the database.

Here is an example of creating a table:

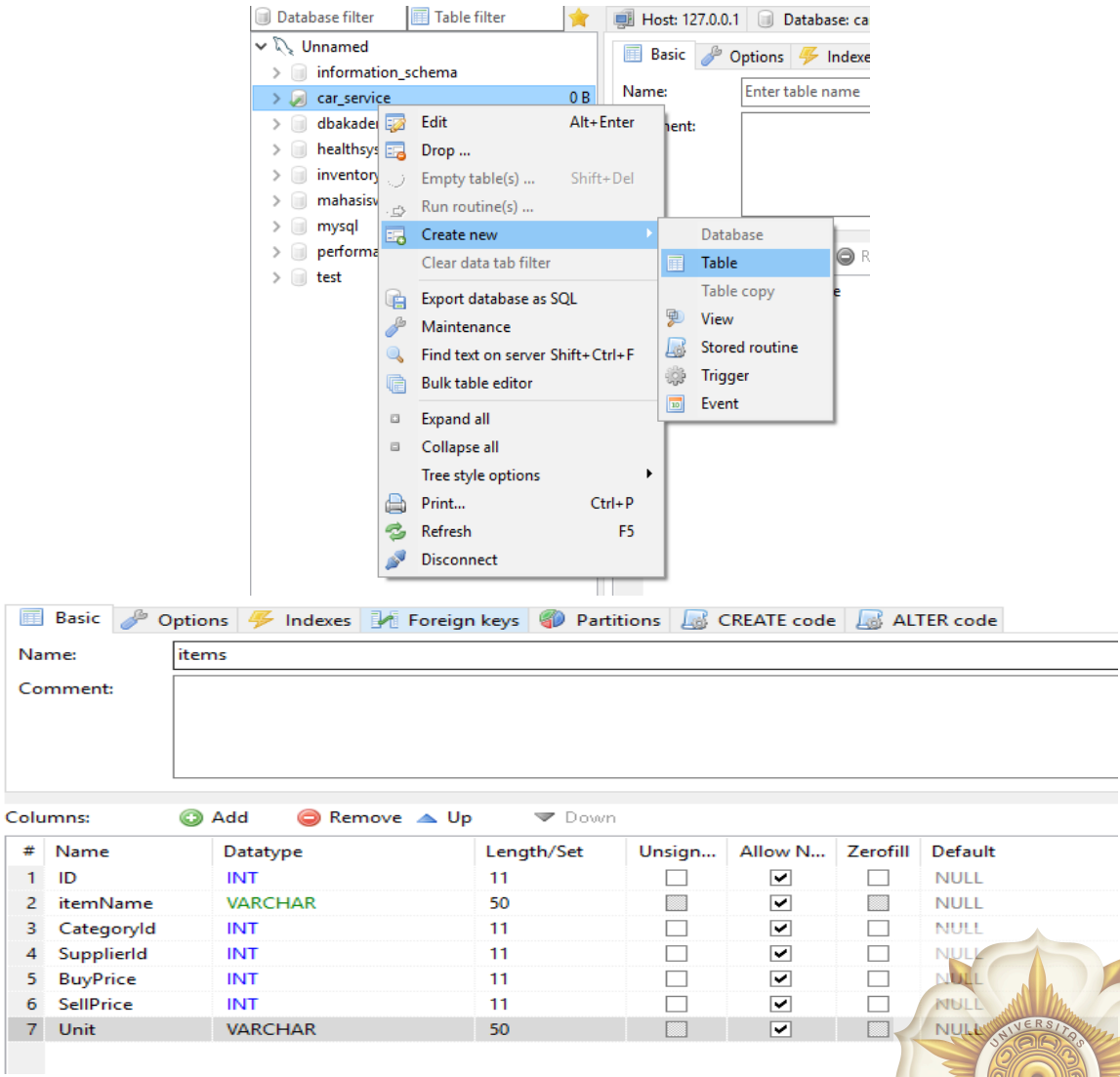


Fig. 3.2 Creating table: items

The following types of data - the data types supported by MySQL:

Table 3.1 Data types supported by MySQL

Data type	declaration	description
Char	char(long)	Column of type char can be filled with text data. The maximum length of only 255 characters.
Varchar	varchar(long)	this type contain text data with max length as declared
Tinytext	Tinytext	Just like varchar
Text	Text	Contains the text data with panjangkarakter wider than previous types.
Integer	Int(long) [unsigned]	Data is a number between 0-4294967295 if unsigned or -2147483648 - 2147483647 if signed. The maximum length is as declared.
Tinyint	Tinyint(long) [unsigned]	Data is a number between 0-255 if unsigned or -128 - 127 if signed. The maximum length of characters corresponding declared
Mediumint	mediumint(long) [unsigned]	Data is a number between 0-1677215 if unsigned or -8,388,608 - 8,388,607 if signed. The maximum length of this data type is as declared
Bigint	bigint(long) [unsigned]	Data is a number between $0 – 10^{20}$
Float	float(long)	Data in the form of real numbers with precision up to power 38. The maximum length is 53.
Double	double(long)	Data in the form of real numbers with precision up to the rank of 308.
Date	Date	Save the Date Date value in format (year - month - date)
Datetime	Datetime	Menyimpan nilai dalam format (tahun - bulan- tanggal – jam – menit - detik)
Timestamp	Timestamp (long)	Save the date and time based on the given length.
Time	Time	stores the data value in (hours minutes seconds) format

3.2.3 Foreign Key

How to define a foreign key is diverse depending on the case. Here are examples of defining foreign key. Here is an example of a query on how to define a foreign key:

```
create table matakuliah(kode varchar(10) primary key not
null, nama varchar(50), sks int, dosen_pengampu
varchar(20) references dosen(nip), hari vachar(10),jam
```

Usually the definition of foreign key is followed by actions that will be valid for the referenced table and the referencing. The action was supported by MySQL include:

1. On Delete Restrict: the corresponding column in the referenced table can not be deleted.
2. On Delete No Action: same as On Delete Restrict.
3. On Delete Cascade: if the referenced column in the table is deleted, the data in the column that references were also removed.
4. On Delete Set Null: If the referenced column in the table is deleted, the data in the column that references changed to null.
5. On Update Restrict: update on a column that is referenced a table not allowed
6. On Update Cascade: if the referenced column in the table is updated, the data table column that references were also updated.
7. On Update Set Null: if the referenced column in the table is updated, the data table column that references will be converted to null.

The command to rename a table in the database is:

```
RENAME TABLE nama_tabel_lama to nama_tabel_baru;
```

An example of SQL implementation is shown in Figure 3.3 to replace the item table into a table items.

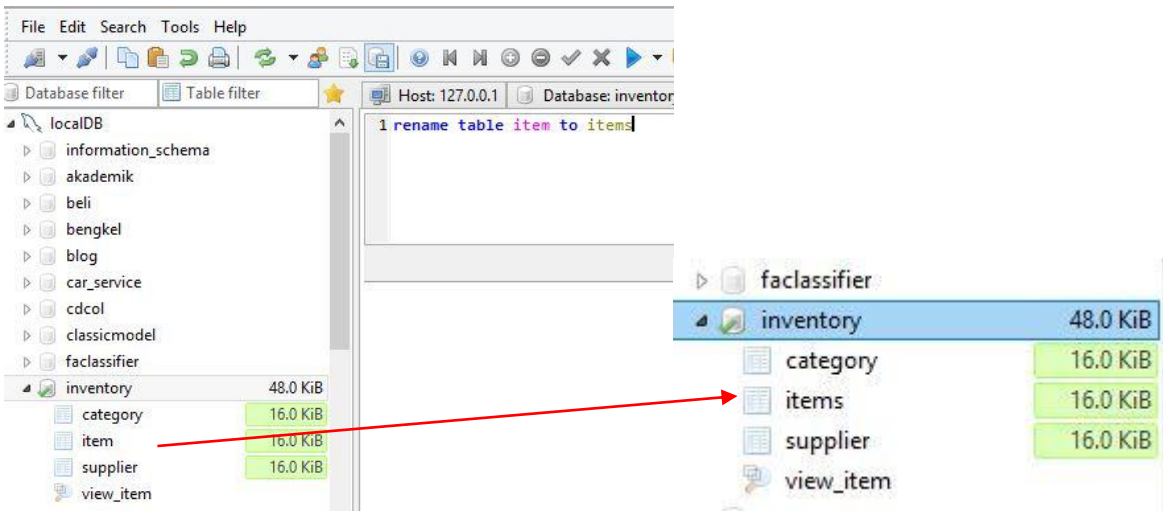


Fig 3.3 Query to change table name from item to items

The command to delete a table in the database is:

```
DROP TABLE tablename;
```

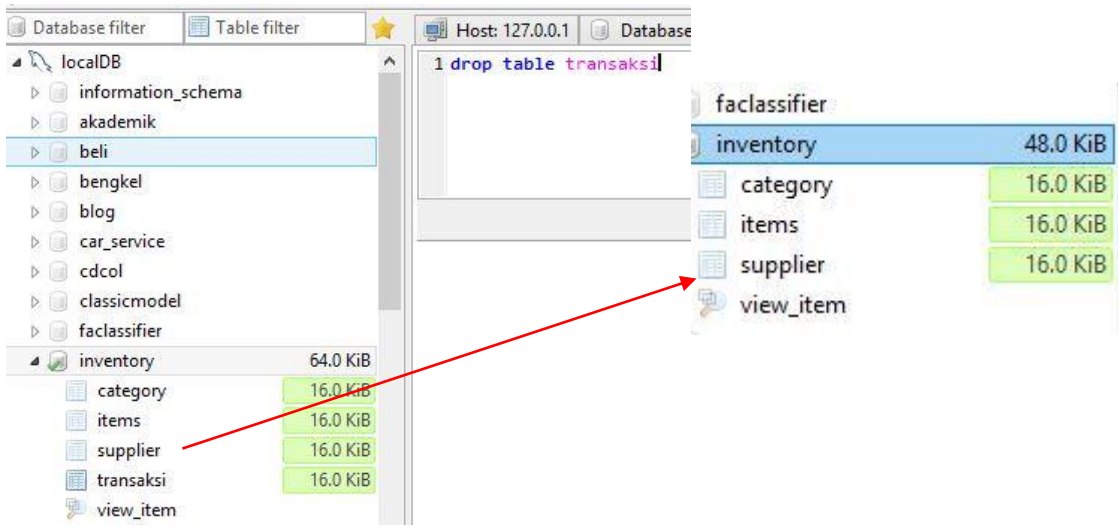


Fig 3.2 Query to delete table *transaksi* from the database

To display the existing table in a database syntax used is:

```
SHOW TABLES database_name;
```

Fig. 3.3 is an implementation from *syntax* show tables that's used to show tables on car_service *database*.

Fig 3.3 Query to show tables on car_service database

To show a table's structure uses this following syntax:

```
DESCRIBE table_name;
```

Figure 3.4 is a query to see the structure the table supplier.

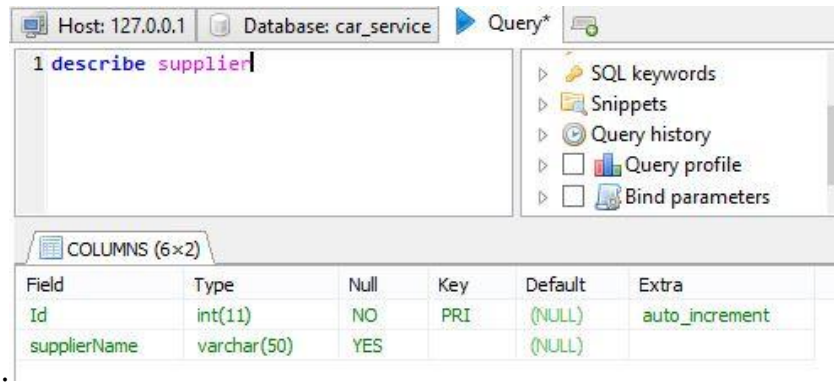


Fig 3.4 Query to show the structure of *supplier* table

3.2.4 ALTER TABLE

"Alter table" is used to change the table structure. Some of them are alter table add column, modify columns, rename columns, and so on. The general syntax for the alter table form:

```
ALTER TABLE table_name alter_type condition;
```

To add a column in a table, use the syntax:

```
ALTER TABLE namatabel add nama_kolom tipe
AFTER/BEFORE nama_kolom_eksis;
```

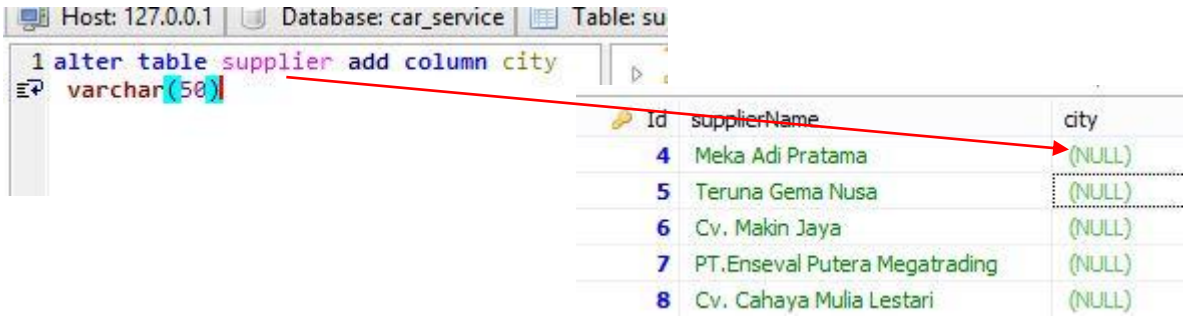


Fig 3.3 Query to add a column named *city* to a supplier table

The next query is a query to change column's name that's aleready on a table. Use this following syntax:

```
ALTER TABLE namatabel change nama_kolom_lama
nama_kolom_baru tipe_data_baru(panjang);
```

The usage of this syntax is shown in Figure 3.4 to rename the column name itemName to name.

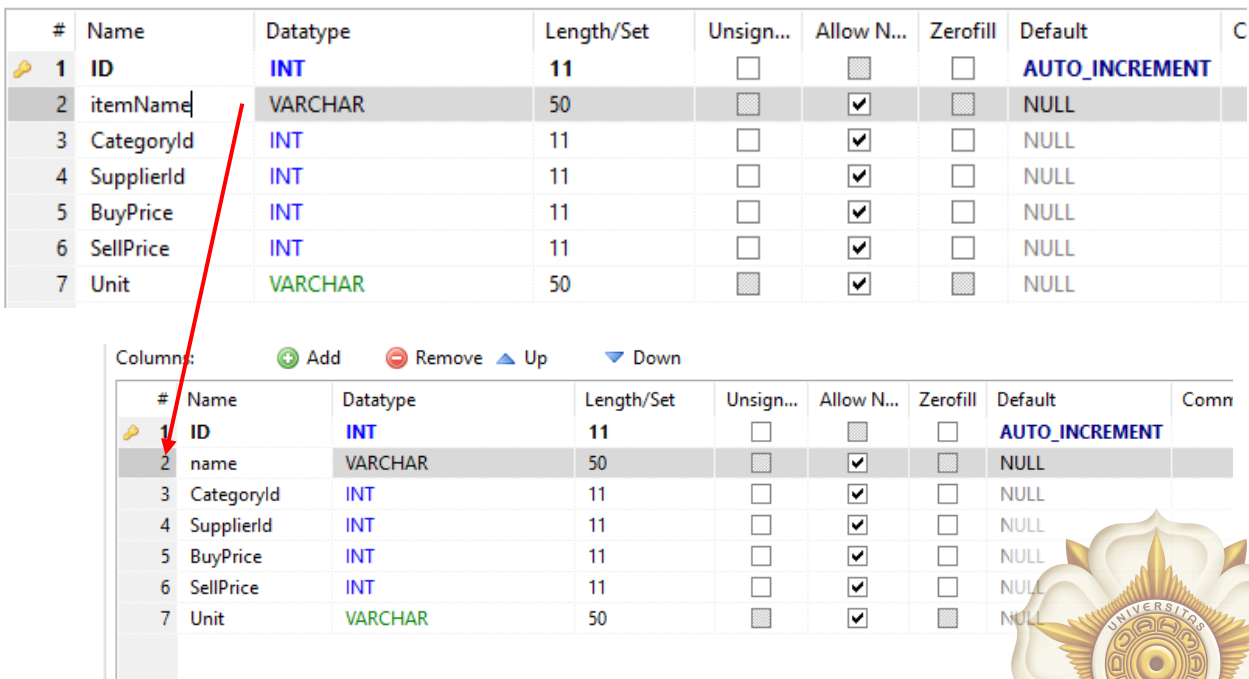


Fig 3.4 Query to change columns name from itemName to name

To delete a column of a table, use this following syntax:

```
ALTER TABLE namatabel DROP nama_kolom;
```

For example,

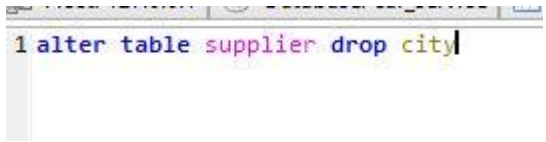


Fig 3.5 Query to delete column city from table *supplier*

To change / modify existing columns in a table, use the syntax:

```
ALTER TABLE namatabel MODIFY COLUMN nama_kolom
kolom_definiton;
```

To add/change a column of a table into its primary key use the following syntax:

```
ALTER TABLE namatabel ADD PRIMARY KEY(nama_kolom);
```

To delete a primary key on a table use this syntax:

```
ALTER TABLE namatabel DROP PRIMARY KEY;
```

3.2.5 Index

Index is the schema objects that are used to increase the speed in getting the desired data rows by using a pointer. The index is created on the basis of the column. Index can be created when a table is created or the existing table. To make the index when the table is created, use the command:

```
CREATE TABLE nama_tabel(kolom type definition,
kolom type definition,…,INDEX(column));
```

To create an index on a table that has been created previously, use the command:

```
CREATE INDEX nama_index ON nama_tabel(nama_kolom);
```

As for removing the index using the command:

```
DROP INDEX nama_index;
```

3.3 STUDENTS ACTIVITY

1. Students try DDL query example from Figure 3.1 to 3.5.
2. Students do exercise 3.4.

3.4 EXERCISE

1. Create an academic database, then create student table (attributes: NIM, name, gender, place of birth, date of birth, telephone and counselors), lecturer (NIP consists of attributes, name, gender, occupation, interests, and telephone), subjects (consisting of attributes: course code, name of courses, credits, lecturers, day, hour, and class room), KRS (consisting of the id attribute KRS, course code, NIM, year, semester, and value). Each field as primary key must be defined auto_increment and not null.
2. Show all existing tables in the database.
3. Show the structure of each table that has been created.
4. Write queries to do the following things:
   a) Delete (drop) primary key on “NIP”.
   b) Re-add primary key on “NIP”.
   c) See *dosen*’s table field.
   d) Change the table name "professor" to "lecturer"
   e) Change the name of the attribute "name" with "lec_name".
   f) Change the data type "jenis_kelamin" became enum { 'male', 'female'}.
   g) Change the data type "phone" into an int.

CHAPTER IV
DATA MANIPULATION LANGUAGE

4.1. LEARNING OUTCOMES

1. Students can write SELECT statements that access data to more than one table using the Join operator, and displays data that does not meet the conditions by using the Outer Join Operator.
2. Students are able to describe the type of problem that can be solved by a sub query, define sub query, understand the types of sub-query, sub query writing a single line and multiple line.
3. Students are able to use the aggregation function.

4.2. THEORY

4.2.1. Data Manipulation Language (DML)

Data manipJulation language (DML) is the SQL commands used to manipulate the data in the database. DML commands including the following:

1. Insert

   Insert is used to insert data into a table.

2. SELECT

   Select is used to select and display data.

3. Update

   Update is used to change the value of data in a table.

4. Delete

   Delete is used to delete data from a table.

4.2.2. Insert

There are common syntaxes used to insert data into the table, namely:

1. INSERT INTO nama_tabel VALUES (nilai1, nilai2, ...)

   This command is used to put the data in the table in sequence from the leftmost column to the very end. Values provided must be equal in number to the columns available in the table.

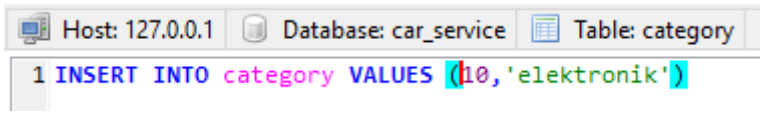


Fig. 4.1 Query MySQL to insert 'ekeltronik' to *category* table

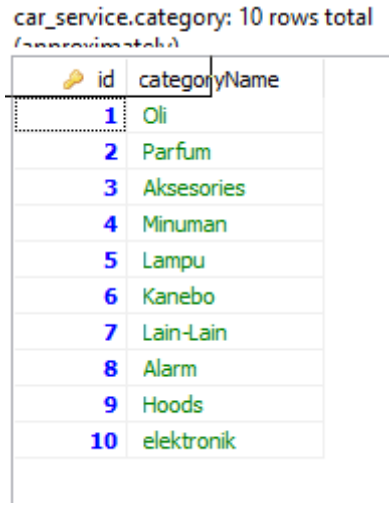


Fig 4.2 The result of insert in Fig. 4.1

2. INSERT INTO table_name (columns) VALUES (data_values)

   Inserting data like this regardless of whether the column should NOT NULL filled or not. It is used as needed, because it can lead to errors as the primary key column is not filled.

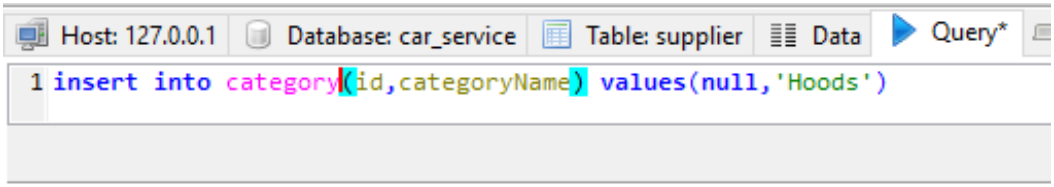


Fig. 4.3 Query MySQL to insert 'Hoods' to *category* table

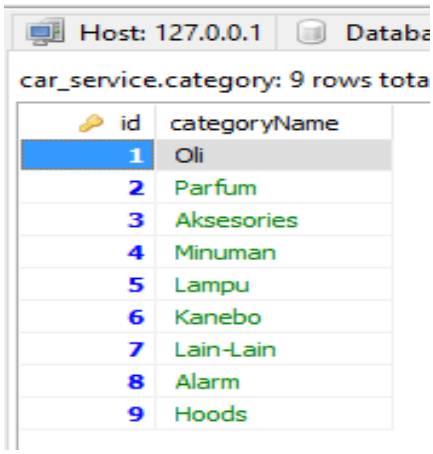


Fig 4.4. The result of inserting 'Hoods' to *category* table

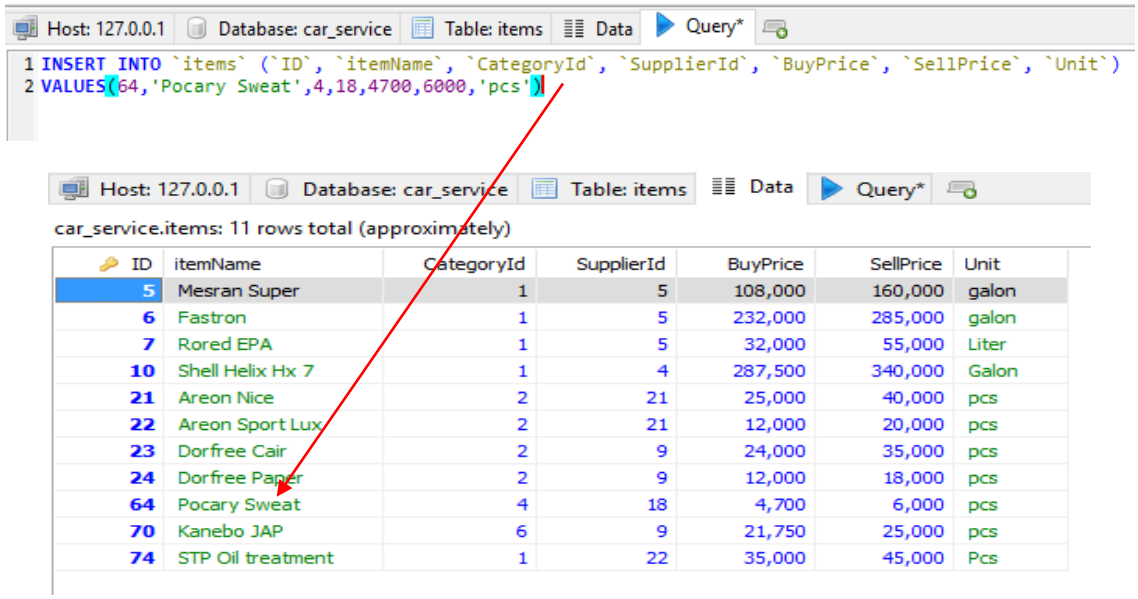


Fig 4.5 MySQL Query to insert 'Pocary Sweat' to *items* table

3. INSERT INTO nama_tabel SET (nama_kolom = nilai, ...)

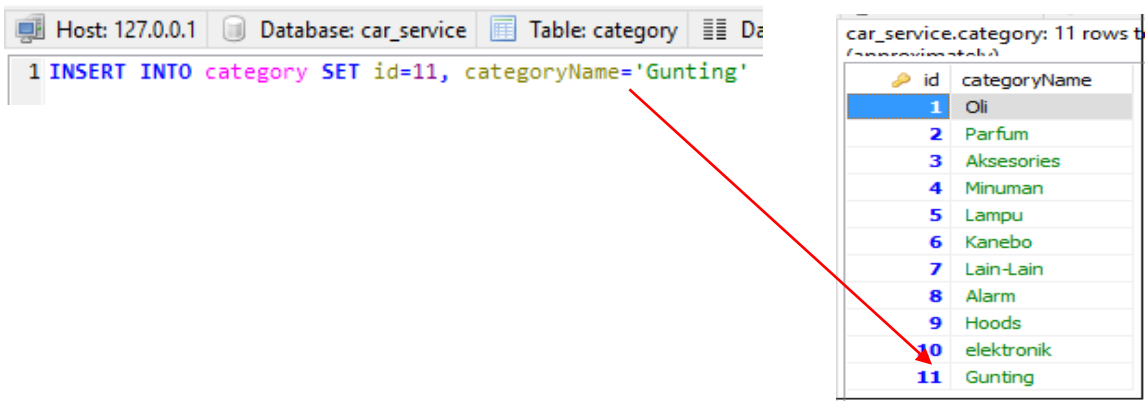


Fig 4.6. MySQL query to insert 'Gunting' to *items*

4.2.3. Delete

Delete is used for deleting records of a table. to delete data in a particular column using the update command. The syntax to delete data from a table in general form:

```
DELETE FROM namatabel

WHERE <kondisi>;
```

Here are some examples of deleting data from table items:

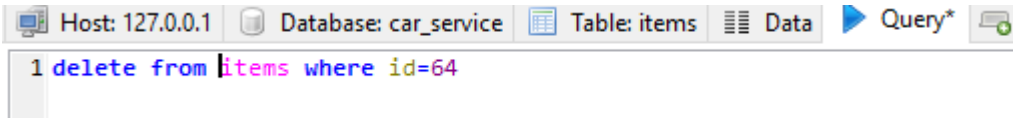
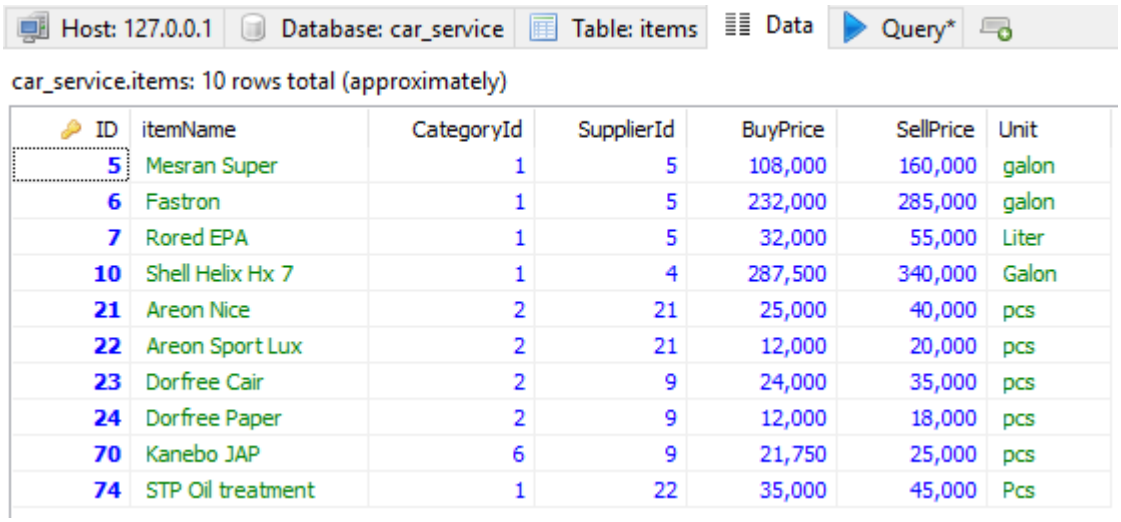


Fig. 4.5. MySQL Syntax to delete "pocary sweat"



ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit
5	Mesran Super	1	5	108,000	160,000	galon
6	Fastron	1	5	232,000	285,000	galon
7	Rored EPA	1	5	32,000	55,000	Liter
10	Shell Helix Hx 7	1	4	287,500	340,000	Galon
21	Areon Nice	2	21	25,000	40,000	pcs
22	Areon Sport Lux	2	21	12,000	20,000	pcs
23	Dorfree Cair	2	9	24,000	35,000	pcs
24	Dorfree Paper	2	9	12,000	18,000	pcs
70	Kanebo JAP	6	9	21,750	25,000	pcs
74	STP Oil treatment	1	22	35,000	45,000	Pcs

Fig. 4.6. The resul after "pocary sweat" has been deleted

Filtering delete uses a SELECT function:

```
DELETE FROM <namatabel> WHERE <namakolom> IN (SELECT <kolom>
FROM namatabel WHERE <kondisi>;
```

Select can only be performed on one column because value matching can only be done with one column.

Filtering using *aggregate* function:

```
DELETE FROM krs WHERE nilai < (SELECT AVG(nilai) FROM krs);
```

The thing that must be considered in delete is this function can only involve one table. FROM seen in the table after only contains a table. This method differs from the FROM SELECT function (discussed below), which could involve more than one table.

4.2.4. Update

Update is a command to change the existing data in the table. Update command has a general syntax as follows:

```
UPDATE namatabel

SET <nama kolom> = <value> WHERE <kondisi>;
```

Example of an update on the items table

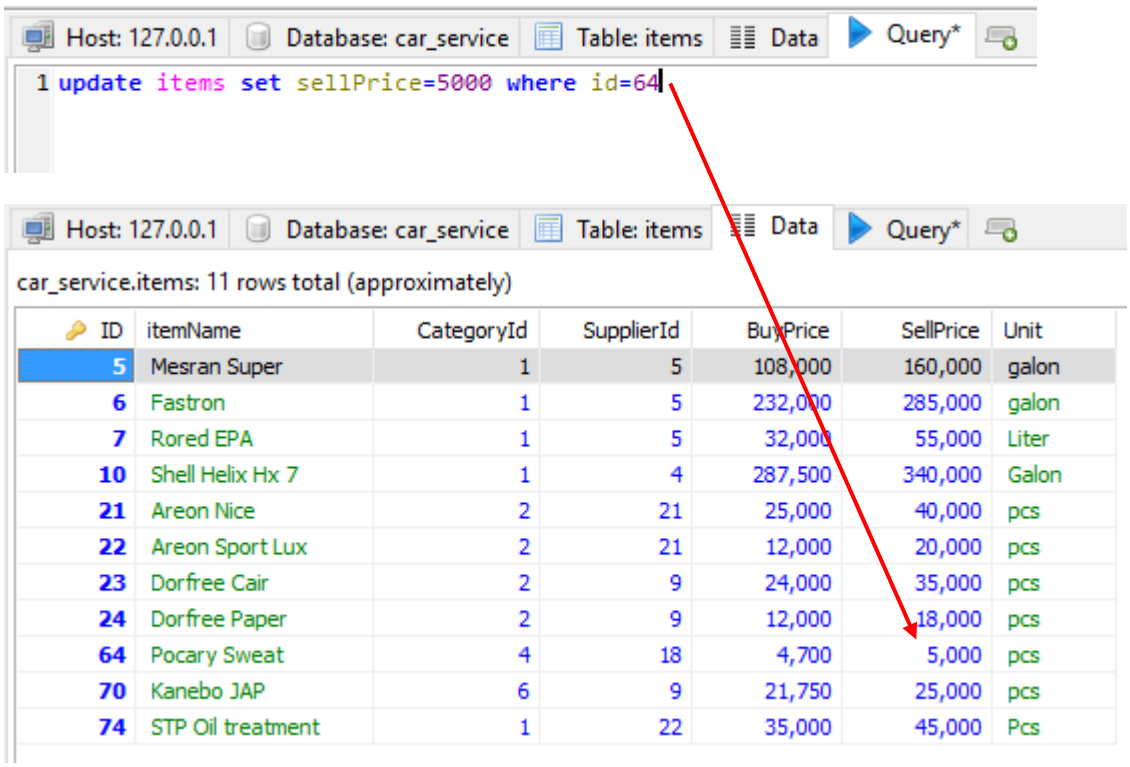


Fig. 4.7 Query to update data sellPrice Pocary Sweat to 5000

4.2.5. Select

Select command is used to select and display data from a database, either from one or more tables. SELECT function in a simple form only has a couple FROM said. Forms are generally as follows:

```
SELECT <nama_kolom>

FROM <nama_tabel>
```

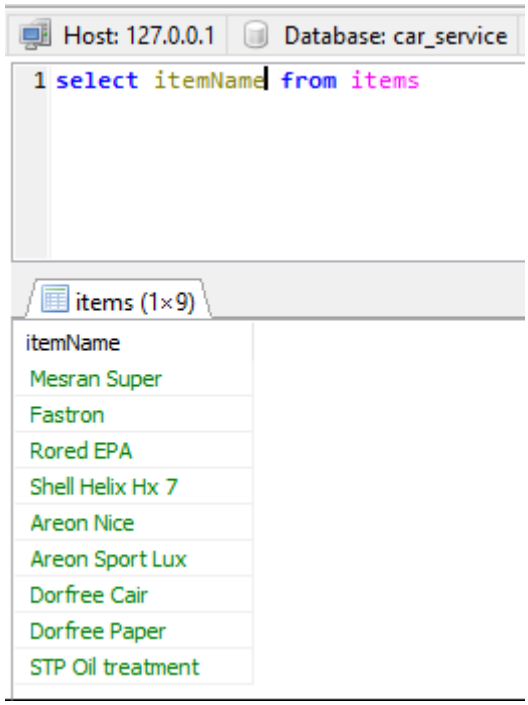
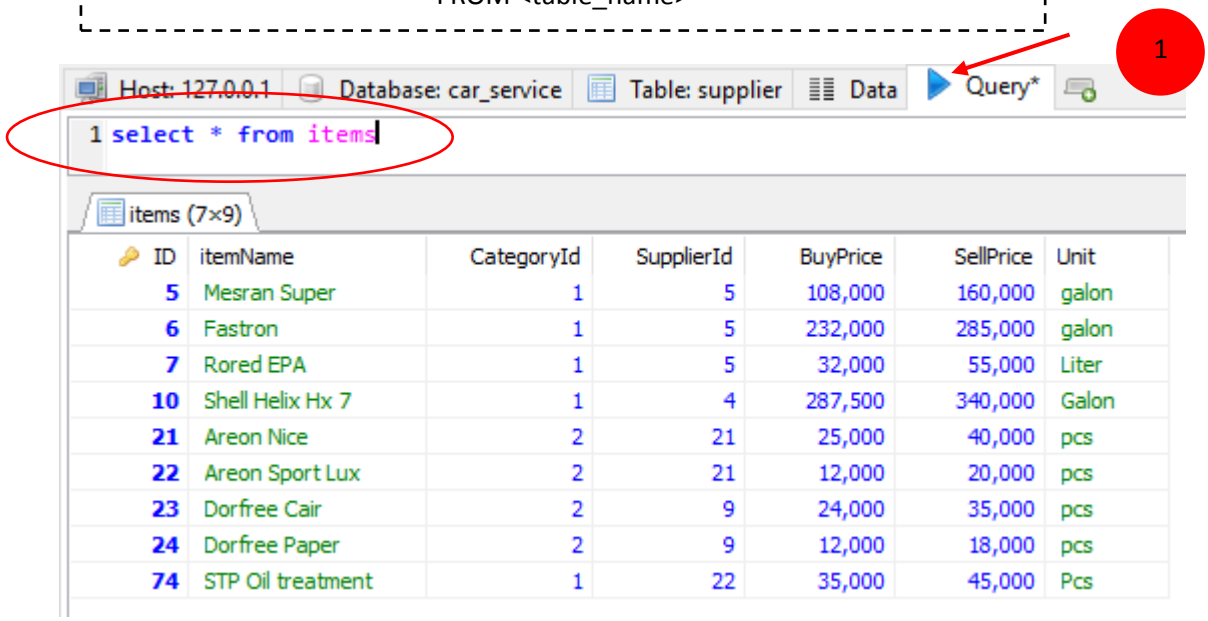


Fig 4.8 Query to show one data grom table *items*

MySQL syntax to show all data in a table is:

```
SELECT *
FROM <table_name>
```



ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit
5	Mesran Super	1	5	108,000	160,000	galon
6	Fastron	1	5	232,000	285,000	galon
7	Rored EPA	1	5	32,000	55,000	Liter
10	Shell Helix Hx 7	1	4	287,500	340,000	Galon
21	Areon Nice	2	21	25,000	40,000	pcs
22	Areon Sport Lux	2	21	12,000	20,000	pcs
23	Dorfree Cair	2	9	24,000	35,000	pcs
24	Dorfree Paper	2	9	12,000	18,000	pcs
74	STP Oil treatment	1	22	35,000	45,000	Pcs

Fig 4.9 Query to show all data from *items* table

The above command is used to select all columns and all records in the table *items*. This kind of selections allows data redundancy (multiple data).

MySQL syntax to select multiple columns on one of the tables is

```
SELECT <namakolom1>, <namakolom2>, …..
FROM <nama_tabel>
```

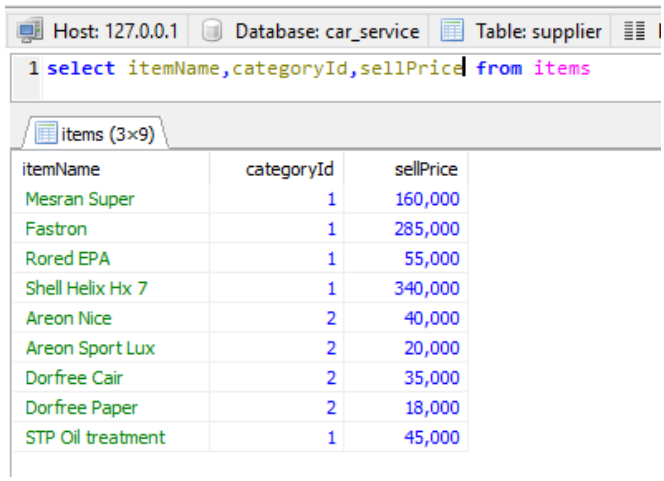


Fig 4.10 Query to show three attributs from *items*

4.2.5.1. Selecting Data Using Math Function

1. Counting the number of rows in a table

```
SELECT count(*) FROM nama_tabel;
```

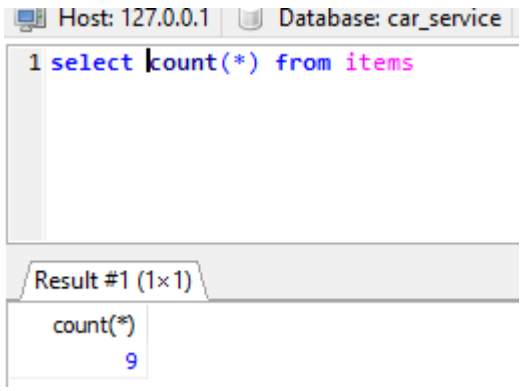


Fig 4.11 Query to count the number of rows

2. Add up the value of some records of a particular column

```
SELECT SUM(nama_kolom) FROM nama_tabel;
```

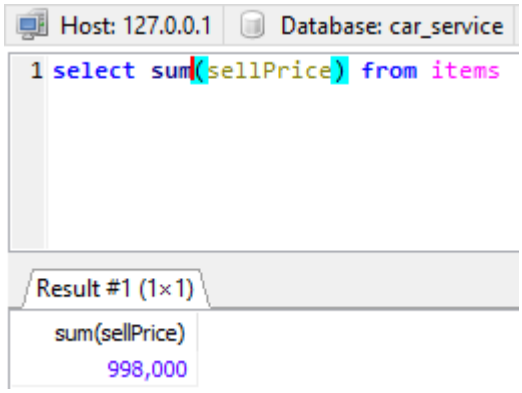


Fig. 4.14 MySQL query to add up data values of a columns

3. Calculate the average value of several records in a particular column

```
SELECT AVG(nama_kolom) FROM nama_tabel;
```

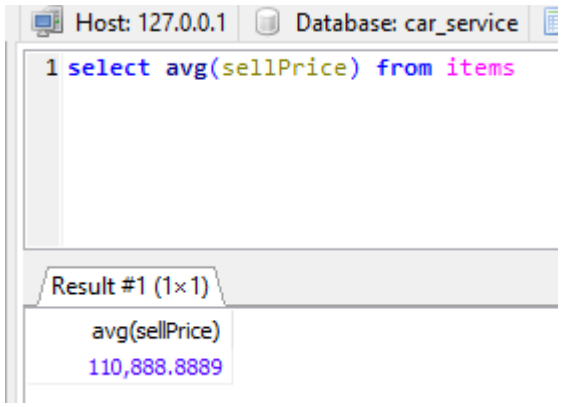


Fig 4.12 MySQL query to calculate the average value of a column

4. Finding the maximum value in a particular column in a table

```
SELECT MAX(nama_kolom) FROM nama_tabel;
```

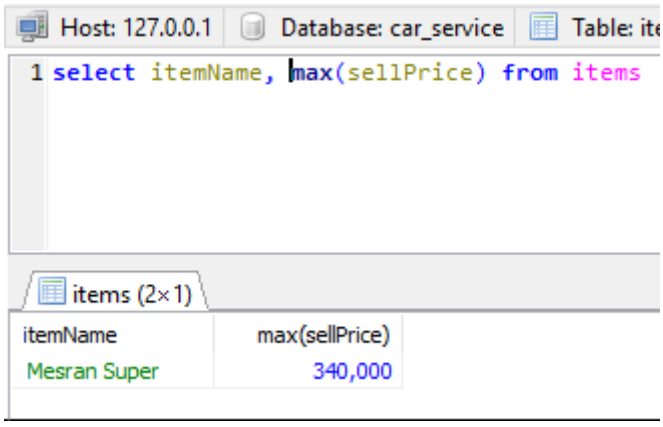


Fig 5.13 MySQL Query to get the maximum value of a column

5. Finding the minimum value in a particular column in a table

```
SELECT MIN(nama_kolom) FROM nama_tabel;
```

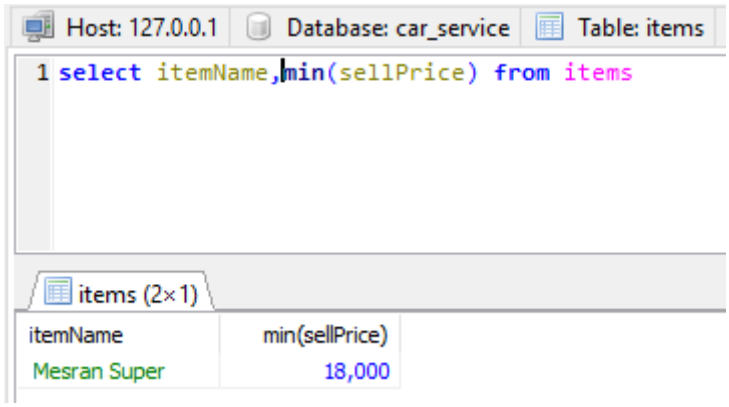


Fig 4.14 MySQL Query to get the minimum value of a column

4.2.5.2. Relational and Logic Operator

There are complements of the function SELECT…FROM that serves to provide the conditions so that the selection results of the SELECT command is as required. There is a relational operators with the sign =, <>, <,>, <=,> =, and also logical operations AND, OR, XOR that are used as a condition in where. The syntax is as follows:

```
SELECT <namakolom1, namakolom2, ….>

FROM <nama tabel>

WHERE <kondisi>
```

The word SELECT is followed by the name of the column to be selected. The word FROM followed by the name of the table where the origin of the selected column. The word followed by a restriction where the records are selected (condition). Examples of use where the SELECT, consider the following example:

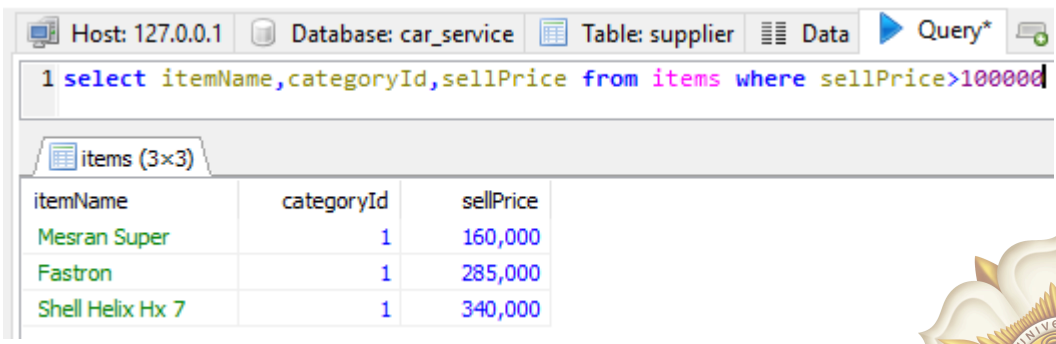


Fig 4.15 MySQL Query to show three attributes with sellPrice above 100000

4.2.5.3. Operator In and Not In

Other functions that follow where is in and not in. Functions “in” or “not in” are used for filtering the selected record. If the data in a column is in accordance with the list in or not in, the records that contain data will be displayed.

```
SELECT <nama kolom>

FROM <nama tabel>

WHERE <nama kolom> in (katakunci1,katakunci2,…)
```

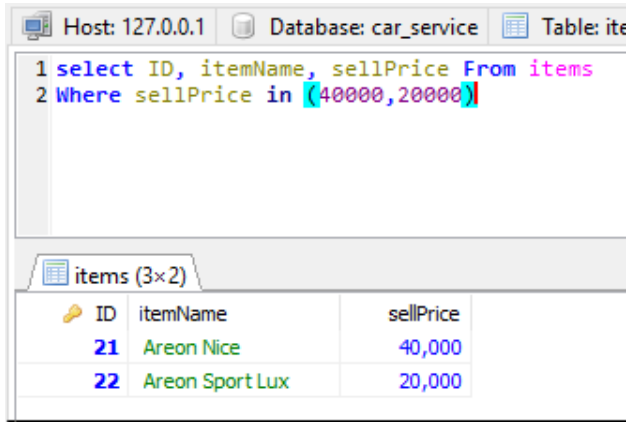


Fig 4.16 Query to show three attribute with sellPrice 40000 or 20000

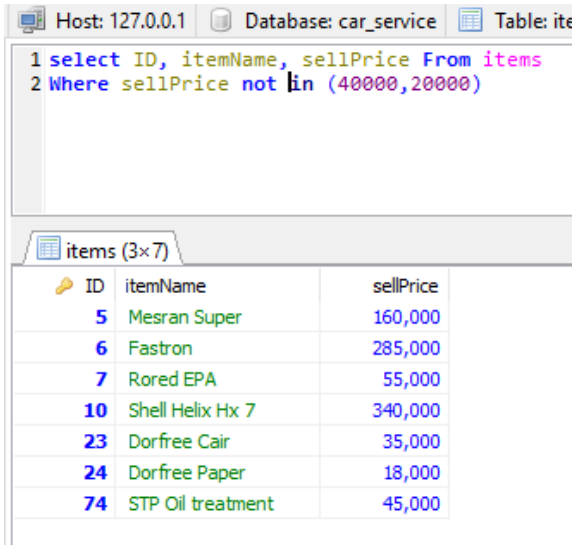


Fig 4.17 MySQL query to show three attributes with sellPrice not 40000 and 20000

4.2.5.4. Operator Like

"Like" can only be used if the data is a string or a character. As an example consider the following query:

```
SELECT <nama kolom>

FROM <nama tabel>

WHERE <nama kolom> LIKE (operator);
```

Karakter yang digunakan untuk fungsi like adalah persen (%) dan underscore(_).

1. % → any string
2. _ → any character

The following examples will further facilitate the understanding of the use of functions like.

1. "Rum%" →"Rumanystring" → any words start with Rum.
2. "%fi%" →"anystringfianystring" → any words containing "fi", in the middle, beginning, or the end of the word.
3. " _ _ " → word that contains exactly two characters.
4. " _ _ _ _%" → word of a minimum of 4 characters.

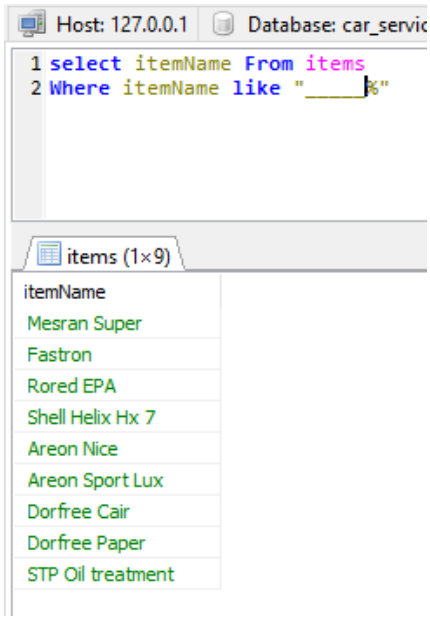


Fig 4.18 Query using LIKE to show data with minimum five characters

4.2.5.5. Null dan Not Null

Additional filtering can be done by using the keyword *is null* or *is not null*. Consider the following query:

```
SELECT <nama kolom>

FROM <nama tabel>

WHERE <nama kolom> is (not null/null);
```

4.2.5.6. Renaming

Renaming is to change the name of a table or give alias. Substitute Naaman is only valid in the query that was it, so that after the query completes, alias created will be lost. Renaming is generally used to abbreviate the name of the table.

```
SELECT <namakolom1, namakolom2,...>

AS <nama1, nama2,...>
```

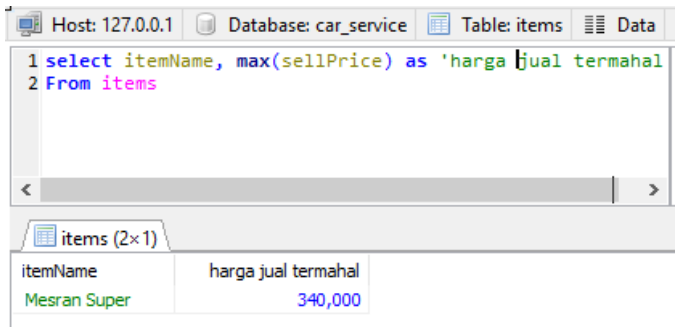


Fig 4.19 Query to show itemName with maximum sellPrice

4.2.5.7. Distinct

Distinct used to display data contained in a column without repetition. Consider the following example.

```
SELECT DISTINCT <nama kolom> FROM <nama tabel>
```

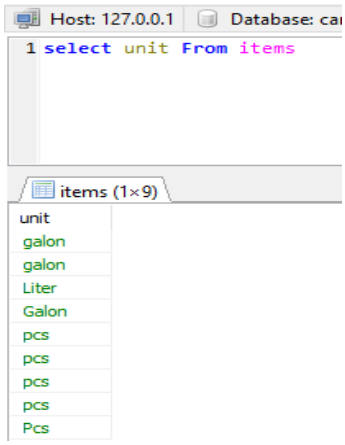


Fig 4.20 Query to show data on unit column

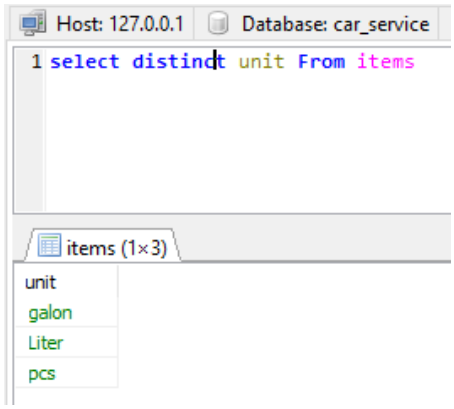


Fig 4.21 MySQL query to show data without repetition

Notice the query in Figure 5.20 shows that the column unit of the entire row in the table krs displayed. While in query Figure 5.21, only unique values are displayed. Or in other words, when using a distinct, repetitive data will be shown once.

4.2.6. Order By

ORDER BY keyword is used to sort the data. There are two types of sequencing data, which in ascending and descending order. Default ORDER BY function is to sort in ascending. If ORDER BY followed by DESC said the new data will be sorted in descending order.

```
SELECT <nama kolom> FROM <nama tabel>

ORDER BY <nama kolom> (DESC/ASC);
```

Here is an example that uses the MySQL query *order by…descending*.

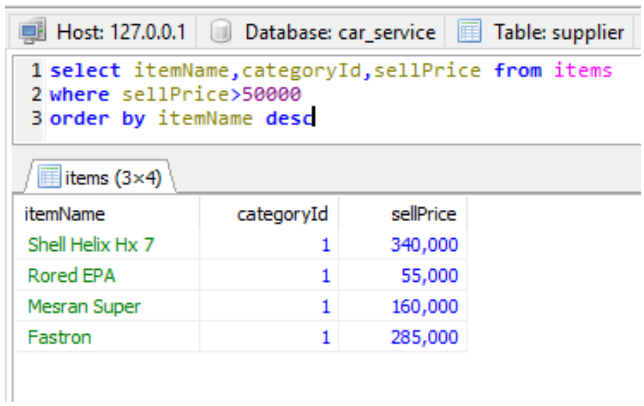


Fig 4.22 MySQL query: *order by*.

4.2.7. Limit

LIMIT function is used to restrict the display of records from a MySQL table we use LIMIT parameter in SELECT command. LIMIT This parameter will be useful when making pagination or settings page to display the contents of the table.

The usage of LIMIT by defining the number or record retrieved is:

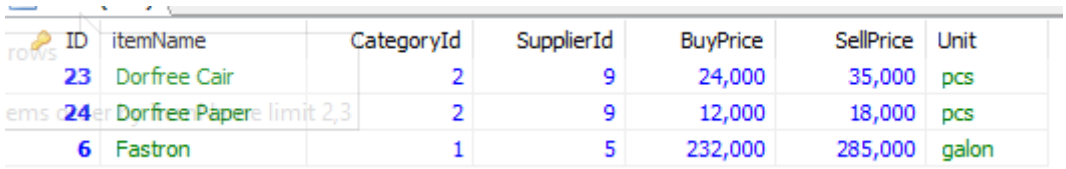
```
SELECT * FROM nama_tabel LIMIT jumlah_record;
```

The usage of LIMIT by writing the initial record and the number of records displayed is as follows:

```
SELECT * FROM nama_tabel LIMIT awal_record, jumlah_record;
```

Here is an example of a SELECT query with LIMIT function.

```
SELECT * FROM items order by itemName limit 2,3 ;
```



ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit
23	Dorfree Cair	2	9	24,000	35,000	pcs
24	Dorfree Paper	2	9	12,000	18,000	pcs
6	Fastron	1	5	232,000	285,000	galon

Fig 4.23 The result of a query with LIMIT

```
SELECT * FROM items order by itemName limit 2,3 ;
```

ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit
24	Dorfree Paper	2	9	12,000	18,000	pcs
6	Fastron	1	5	232,000	285,000	galon

Fig 4.24. The result of a query with LIMIT

There are differences in the results of running the query differences in both syntax. The explanation on the matter is:

1. Limit 2, 3: The query results will display data starting from the index line 2, as many as 3 record data.
2. Limit 3, 2: The query results will display data starting from the 3rd row index, as many as 2 record data.

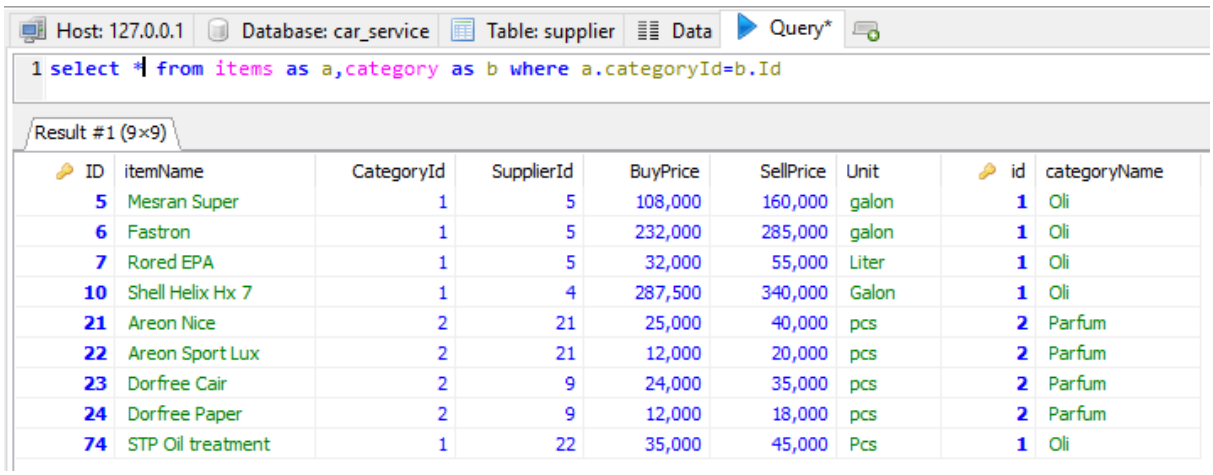
Catatan: Index rows in MySQL starts from 0 (zero).

4.2.8. Cross Join

Cross join is an amalgamation of several tables that will produce a Cartesian product. General of the CROSS JOIN syntax is:

```
SELECT nama_kolom FROM nama_tabel1, nama_tabel2, nama_tabel-n
```

Here's an example of *cross join* implementation that has been modified:



ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit	id	categoryName
5	Mesran Super	1	5	108,000	160,000	galon	1	Oli
6	Fastron	1	5	232,000	285,000	galon	1	Oli
7	Rored EPA	1	5	32,000	55,000	Liter	1	Oli
10	Shell Helix Hx 7	1	4	287,500	340,000	Galon	1	Oli
21	Areon Nice	2	21	25,000	40,000	pcs	2	Parfum
22	Areon Sport Lux	2	21	12,000	20,000	pcs	2	Parfum
23	Dorfree Cair	2	9	24,000	35,000	pcs	2	Parfum
24	Dorfree Paper	2	9	12,000	18,000	pcs	2	Parfum
74	STP Oil treatment	1	22	35,000	45,000	Pcs	1	Oli

Fig 4.25 The result of *cross join* query (categoryid on items = id on category)

4.2.9. Inner Join

INNER JOIN will retrieve all rows from the table origin and destination table with key value-related conditions only (if any). If no, then the row will not appear. If there are no associated key conditions between tables, then all rows from both tables combined. General of the INNER JOIN syntax is as follows:

```
SELECT nama_kolom1, nama_kolom2, nama_kolom-n FROM nama_tabel1 INNER JOIN
nama_tabel2 ON klausa
```

```
1 select * from items inner join category on categoryId=category.Id
```

Result #1 (9×9)

ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit	id	categoryName
5	Mesran Super	1	5	108,000	160,000	galon	1	Oli
6	Fastron	1	5	232,000	285,000	galon	1	Oli
7	Rored EPA	1	5	32,000	55,000	Liter	1	Oli
10	Shell Helix Hx 7	1	4	287,500	340,000	Galon	1	Oli
21	Areon Nice	2	21	25,000	40,000	pcs	2	Parfum
22	Areon Sport Lux	2	21	12,000	20,000	pcs	2	Parfum
23	Dorfree Cair	2	9	24,000	35,000	pcs	2	Parfum
24	Dorfree Paper	2	9	12,000	18,000	pcs	2	Parfum
74	STP Oil treatment	1	22	35,000	45,000	Pcs	1	Oli

Fig 4.26 The result of query with *inner join* (categoryId on *items* = Id on *category*)

4.2.9. Left (Outer) Join

Left outer join will display all rows from the left table, and will display the results in the right table NULL if there is no relationship with the left table.

Host: 127.0.0.1 | Database: car_service | Table: supplier | Data | Query*

```
1 select * from supplier left outer join items on supplierId=supplier.Id
```

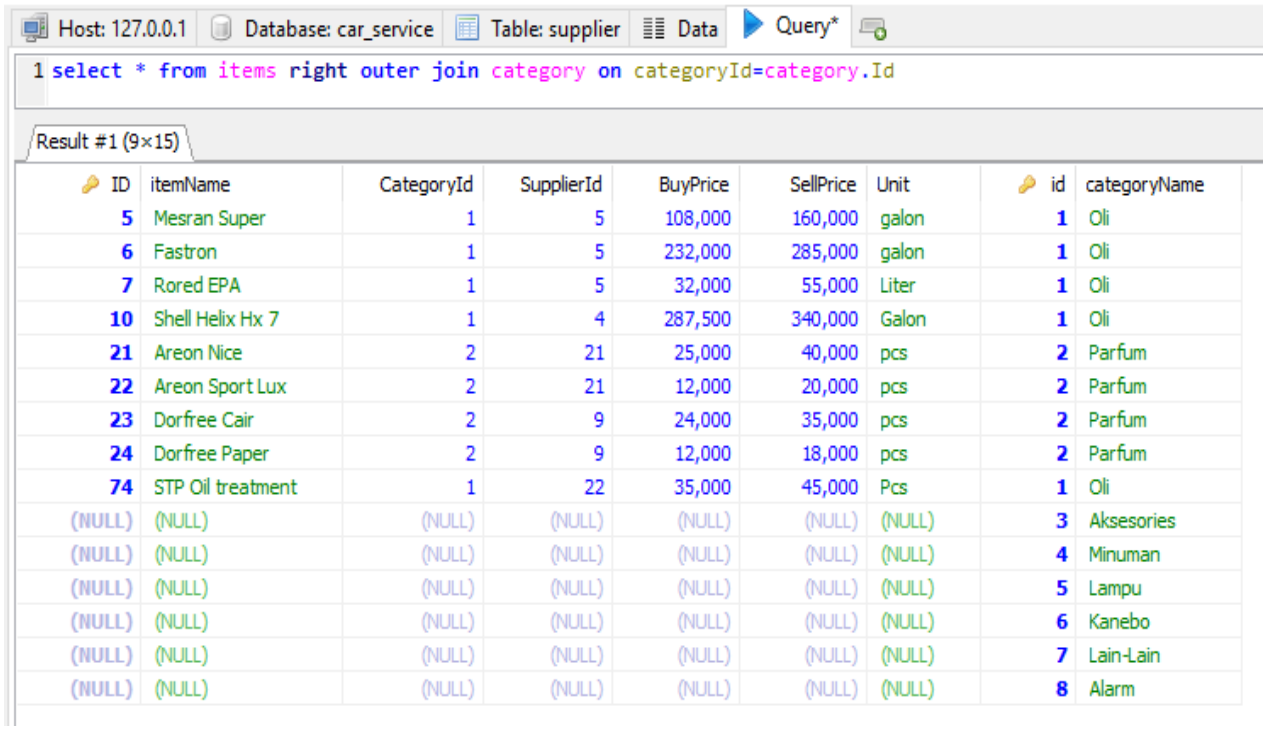
Result #1 (9×18)

Id	supplierName	ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit
5	Teruna Gema Nusa	5	Mesran Super	1	5	108,000	160,000	galon
5	Teruna Gema Nusa	6	Fastron	1	5	232,000	285,000	galon
5	Teruna Gema Nusa	7	Rored EPA	1	5	32,000	55,000	Liter
4	Meka Adi Pratama	10	Shell Helix Hx 7	1	4	287,500	340,000	Galon
9	JJ Cars Accessories	23	Dorfree Cair	2	9	24,000	35,000	pcs
9	JJ Cars Accessories	24	Dorfree Paper	2	9	12,000	18,000	pcs
6	Cv. Makin Jaya	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
7	PT.Enseval Putera Megatrading	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
8	Cv. Cahaya Mulia Lestari	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
10	RS Accessories	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
11	Padi Mili	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
12	Aneka Jaya	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
13	Sumber Rejeki	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
16	Trimo Motor	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
17	Wurth	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
18	PT. Dimas Surya Aditama	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
19	Toko Karunia	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)
20	Nabita Motor	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)

Fig 4.27 The example of a result of *left outer join* query

4.2.10. Right (Outer) Join

The opposite of a left outer join, right outer join will display all rows from the right table, and will display the results in tables left NULL if there is no relationship with the right table.



ID	itemName	CategoryId	SupplierId	BuyPrice	SellPrice	Unit	id	categoryName
5	Mesran Super	1	5	108,000	160,000	galon	1	Oli
6	Fastron	1	5	232,000	285,000	galon	1	Oli
7	Rored EPA	1	5	32,000	55,000	Liter	1	Oli
10	Shell Helix Hx 7	1	4	287,500	340,000	Galon	1	Oli
21	Areon Nice	2	21	25,000	40,000	pcs	2	Parfum
22	Areon Sport Lux	2	21	12,000	20,000	pcs	2	Parfum
23	Dorfree Cair	2	9	24,000	35,000	pcs	2	Parfum
24	Dorfree Paper	2	9	12,000	18,000	pcs	2	Parfum
74	STP Oil treatment	1	22	35,000	45,000	Pcs	1	Oli
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	3	Aksesories
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	4	Minuman
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	5	Lampu
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	6	Kanebo
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	7	Lain-Lain
(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	(NULL)	8	Alarm

Fig 4.28 The result of *right outer join* which supplierId in *items* table must have the same value as Id in supplier table

4.2.11. Select with many tables

SQL does not specify a limit to the number of tables that may be incorporated in the SELECT statement. The ground rule is to make the join remains the same. First, list all the tables and then define the relationship between each table. For example, as follows:

```
SELECT
      tabel1.column_name1,
      tabel1. column_name2,
      tabel2. column_name1,
FROM tabel1, tabel2  WHERE tabel1.PK=tabel2.FK
```

For example of select with many tables can be seen in Figure 4.28

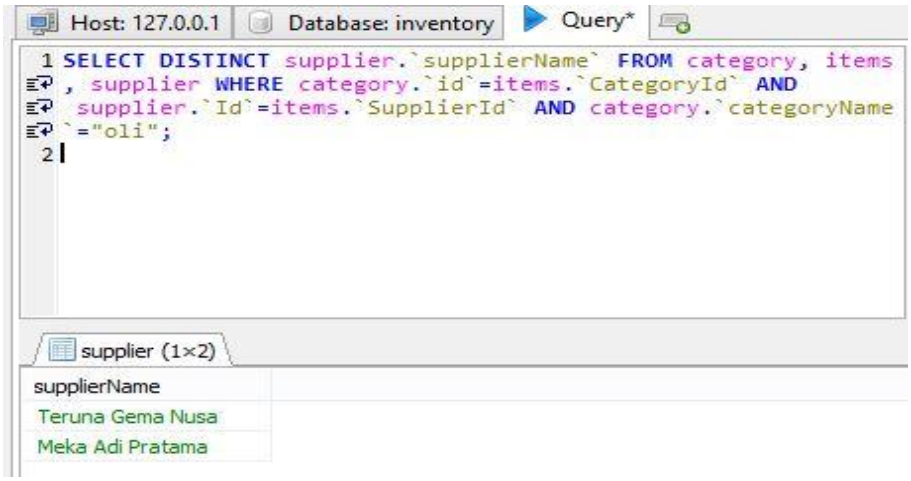


Fig 4.28 Example of query with many tables

The example above is a query to display the name of the supplier to supply goods to the oil category. On querying the merger of the three tables that category, items and supplier. The number of tables that can be combined in a query can be adjusted with the will of the practitioner.

4.2.12. Subquery

Nested query or subquery is also called a query that has at least two SELECT commands. The format of the subquery is as follows:

```
SELECT nama_kolom [, nama_kolom]
FROM   tabel1 [, tabel2 ]
WHERE column_name OPERATOR
      (SELECT nama_kolom [, nama_kolom]
      FROM tabel1 [, tabel2]
      [WHERE])
```

For the application of sub-query can be seen in the example shown in Figure 4.29

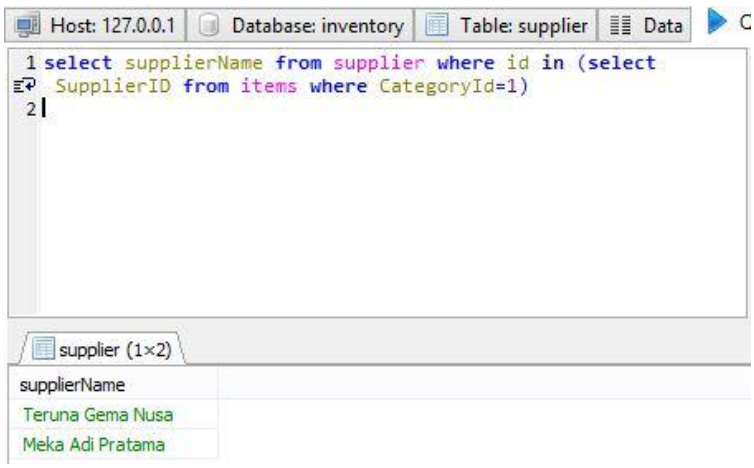


Fig 4.29 Shows supplier name that supplies items in oil category using subquery

The above picture shows the query to display supplier which supplies bbarang with oil category. This subquery will show the same results as a query using the many tables. A subquery used to restore data to be used in the main query as a condition to further givie restriction to the data to be retrieved.

4.2.13. Aggregate Function and GROUP BY

Aggregate functions are simple mathematical functions in SQL. Aggregate functions can only be used for a column of type numbers (eg, integer, shortint). Aggregate functions are:

SUM	□	the
AVG	□	the average value of a column
MAX	□	the maximum value of a cooumn
MIN	□	the minimum value of a column
COUNT	□	the number of available data in a column

GROUP BY function is used to classify data based on according to one or more columns. This function is used for facilitating aggregate function.

```
SELECT column1, column2
FROM table_name
WHERE [ conditions ]
GROUP BY column1, column2
ORDER BY column1, column2
```

The implementation example of this query is shown in Figure 4.30., displaying the supplier and the number of items that supplier is supplying.

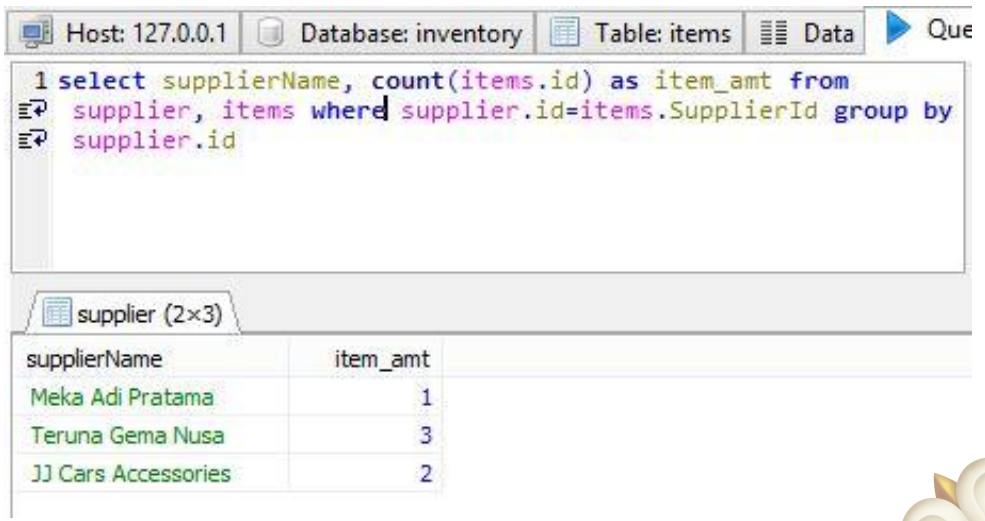


Fig 4.30 Shows supplier's name and the number of item the suppliers supply

4.2.14. HAVING

Having is a complementary function of aggregate and used with GROUP BY. Its function is to select the data displayed by certain aggregate conditions.

```
SELECT column1, column2
FROM table1, table2
WHERE [ conditions ]
GROUP BY column1, column2
HAVING [ conditions ]
ORDER BY column1, column2
```

Example of the usage of **having** in a query is shown in fig. 4.31.

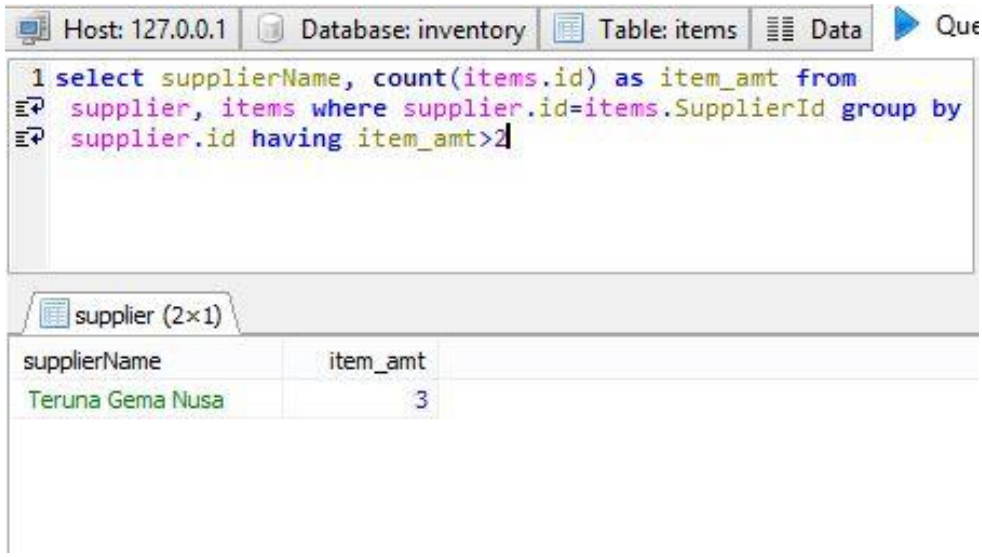


Fig 4.31 Showing supplier's name who supplies more tha two items

Figure 4:31 is an example of the use of having to display the name of the supplier with the number of items that are stocked more than 2. The result of this query is Teruna Gema Nusa by the number of items is equal to 3.

4.2.15. Exist

Exist meaning of it is there. This keywords used in the filtering exists after the word where, normally used on a nested query. Notice the common format in use exists.

```
SELECT column-names
 FROM table-name
 WHERE EXISTS
 (SELECT column-name FROM table-name WHERE condition)
```

Exists Syntax exists is used to determine keberasdaan every record in the subquery. Exists will return true if the value of subquery provide one or more data records. Exists syntax is also commonly used with correlated subquery. Implementation of use exists can be seen in the example query to display the name of the supplier with items that have a purchase price> 100000.

Fig 4.33 Show supplier's name having item's buy price above 100000

From the query results shows that a supplier who has a purchase price of items with price> 100000 is Meka Adi Pratama and Teruna Gama Nusa

4.2.16. VIEW

View started there in MySQL version 5.0. View used to simplify SQL queries and limit the fields that are needed when accessing the table. View as a table, but the data came from another table. View will be stored in the database matches the name that is created. View actually is the composition of the table in the form of a query that has been set.

Sintaks view adalah sebagai berikut:

```
CREATE VIEW view_name AS

SELECT column1, column2.....

FROM table_name

WHERE [condition];
```

Based on the view that existing syntax, it will be created a view with the name view_items whose content consists of Items_View, selling price and unit. Implementation syntax view can be seen in the example below:

Host: 127.0.0.1 | Database: inventory | Table: [Untitled] | Query*

```
1 CREATE VIEW Items_View AS
2 SELECT itemName, SellPrice, Unit
3 FROM  items
```

This query will produce a virtual table that is named *Items_View* containing data such as item name, selling price and unit, as shown below:

inventory.view_item

itemName	SellPrice	Unit
Mesran Super	160,000	galon
Fastron	285,000	galon
Rored EPA	55,000	Liter
Shell Helix Hx 7	340,000	Galon
Areon Nice	40,000	pcs
Areon Sport Lux	20,000	pcs
Dorfree Cair	35,000	pcs
Dorfree Paper	18,000	pcs
STP Oil treatment	45,000	Pcs

Fig 4.33 Creation view to show item's name, sell price, and its unit.

This view can be generated from one or many tables depending on the SQL query is written to make an appearance. This view allows the user to do the following:

- Restricting access to data so that users can only view or change only what is needed.
- Summarizing data from multiple tables that can be used to aimlessly program.

4.3. STUDENTS ACTIVITY

1. Students try the usage example of DML query from fig. 4.1 to 4.32.
2. Students do exercise 4.4.

4.4. EXERCISE

Write a query MySQL to:

a. Show all items with its supplier.

b. Sort the result data of (a) based on supplier's name.

c. Show all items from supplier Meka Adi Pratama.

d. Do a group and count item based on item's supplier.

e. Delete all items supplied by Teruna Gema Nusa.

f. Delete all items having sellPrice between 100000 and 150000.

g. Show items with 'lu' in its name

h. Show suppliers that do not supply items to Car Service.

CHAPTER V
STORED ROUTINE: PROCEDURE

5.1 LEARNING OUTCOMES

1. Understand the concept of stored procedures and use of MySQL.
2. Knowing the syntax for declaring and calling the stored procedure.
3. Able to declare (create), delete, and call stored procedures in MySQL database

5.2 THEORY

Stored routine is a set of SQL commands that are stored on the server. When a stored routine has been made, the client does not need to set the order mejalanakan perbarisnya back but simply call a stored routine that has been stored. Stored routines can be either the procedure or function.

Stored procedure is a procedure (such as subprogram in the programming language) that is stored in the database. The advantages of using stored procedures, among others:

1. Fast, compilation done in the database so that the data traffic from / to the application program is reduced.
2. The application program becomes more simple and quick.
3. The form of the object in the database, thus eliminating dependence on the programming language used.
4. Characteristically portable, if the database can be installed anywhere, it can be ascertained that the stored procedure can also be executed anywhere.

General syntax of writing a stored procedure that is CREATE PROCEDURE nama_sp ([parameter_prosedur [, ...]]) isi_prosedur and closed with a delimiter. Delimiter is a marker of the end of an SQL statement, the default is a semicolon (;). Before creating a stored procedure delimiter should be replaced with symbols that are rarely used such as ^, #, and others.

Explanation of general syntax is as follows. Italicized part needs to be tailored to the needs. There are three parameters, namely procedures IN, OUT, and INOUT followed by variable names and data types. The square brackets mean the option, it can be used or not. While the sign [...] means it can be used more than one parameter procedure.

The three parameters mentioned procedure can be explained as follows:

1. IN (*default*): put the value from the memory to the SP.memasukkan nilai dari memori ke SP.
2. OUT: providing output value from the SP to the memory.
3. INOUT: put the value from the memory to the SP and also able to provide an output via the same parameter.

Another frequently used syntax is the syntax to view the status of a stored procedure, the syntax to view the content of stored procedures, and syntax to call the stored procedure. The syntax to view the status of a stored procedure is *SHOW PROCEDURE STATUS;*. The syntax to view the contents of Stores procedure is *SHOW CREATE PROCEDURE nama_sp;*. The syntax to call a stored procedure is *CALL nama_sp (value);*. While the syntax to delete a stored procedure is *DROP PROCEDURE nama_sp;*.

5.3 STUDENTS ACTIVITY

1. Scenario A: Making a *Hello World*!
   a. Step 1 : *Login* as *root*

      Open *command prompt/terminal* and then *login* to *MySQL* using "mysql -u root -p" syntax (without quotation mark). There will be a prompt asking for *password*.

   b. Step 2: Writing *stored procedure* syntax

      Because *stored procedures* are saved in the database, the first thing to do is go inside/use a database using *USE database_name* syntax. To create a one liner procedure, there's no need to change the delimiter value. Here's the syntax to create a *Hello World!* stored procedure:

```
CREATE STORED PROCEDURE helloworld ()
      SELECT "Hello World!";
```

   c. Step 3: Viewing *stored procedure*'s status

      View the status of *stored procedure* that's been created.

   d. Step 4: Viewing the content of *stored procedure*

      View the code line of *stored procedure* with the name *helloworld* that has been created.

   e. Step 5: Calling *stored procedure*

      Using CALL syntax, call the *stored procedure* to show *Hello World*! That's been created in step 2.

2. Scenario B: *Stored procedure* with IN parameter
   a. Step 1 : Create *stored procedure* with IN parameter

      Create a *stored procedure* to show student data with faculty major code as an *input*. Here's the syntax:

```
CREATE PROCEDURE spDataMhs (IN prodiID CHAR(3))
SELECT * FROM mahasiswa WHERE kode_prodi LIKE prodiID;
```

   b. Step 2: Calling *stored procedure*

      Using CALL syntax, call a *stored procedure* named *spDataMhs* that's been created in step 1 with with faculty major code as an *input*.

3. Scenario C : *Stored procedure* with OUT parameter
   a. Step 1 : Making *stored procedure* with OUT parameter

      Create a *stored procedure* to count and display the number of subject offered in table mata_kuliah. Here's the syntax:

```
CREATE PROCEDURE spJumlahSKS (OUT jumlah INT)
SELECT SUM(sks) INTO jumlah FROM mata_kuliah;
```

   b. Step 2 : Calling *stored procedure*

      Caling a stored procedure with OUT parameters is slightly different to the previous IN parameter. On the stored procedure with OUT parameters, CALL syntax used to store query results in the stored procedure in a particular variable, for example jum then to call those variables used SELECT syntax. The full syntax is as follows.

```
CALL spJumlahSKS (@jum);
SELECT @jum AS 'Jumlah SKS';
```

4. Scenario D: *Stored procedure* with IN and OUT parameters
   a. Step 1: Making *stored procedure* with IN and OUT parameters

      In this scenario, the contents stored procedure consists of several SQL statements that need to change the default delimiter beforehand. Create a stored procedure to view the student data with input birthplace then tampilkanlah number of students who were born in that place. To display the number of students used the syntax FOUND_ROWS () which will count the number of rows that result from the previous SELECT statement. The full syntax is as follows.

```
DELIMITER %%
CREATE PROCEDURE spTempatMhs (IN param1 varchar(20), OUT param2 INT)
BEGIN
     SELECT * FROM mahasiswa WHERE tempat_lahir LIKE param1;
     SELECT FOUND_ROWS() INTO param2;
END;%%
```

b. Step 2: Calling *stored procedure*

The calling of this stored procedure combines the calling parameters IN and OUT parameters so that there are two inputs such as 'W%' as input for parameter IN would ask the student data that was born in a place with the prefix letter 'W' and @jml as input for the parameter OUT will keep the number of students who were born in that place. The full syntax is:

```
CALL spTempatMhs('W%', @jml);
SELECT @jml;
```

5. Scenario E : *Stored procedure* with INOUT parameter

a. Step 1: Making *stored procedure* with INOUT parameter

Create a *stored procedure* to change a date format, from YYYY-MM-DD to DD-MM-YYY. Here's the syntax:

```
CREATE PROCEDURE spTanggal (INOUT tgl VARCHAR(10))
SELECT CONCAT(RIGHT(tgl,2) ,'-', SUBSTR(tgl,6,2), '-', LEFT(tgl,4)) INTO
tgl;
```

b. Step 2: Calling *stored procedure*

In this scenario, there is an addition to call the *stored procedure*, that is to give an input value using syntax SET. Here's the complete syntax:

```
SET @tanggal = '2020-10-20';
CALL spTanggal (@tanggal);
SELECT @tanggal;
```

6. Scenario F : Exercise making a *stored procedure*

a. Step 1 : Creating a *stored procedure*

Create *stored procedure* named *spFakultas* and faculty code and faculty name as *inputs*. Here are the conditions:

- If the faculty code is on the table, the *stored procedure* will do an UPDATE on faculty name.
- If the faculty code is not on on the table, the *stored procedure* will do an INSERT (the faculty data will be inserted).

b. Step 2 : Calling *stored procedure*

Call the *stored procedure* in two conditions: the first is the faculty code is on the table and the second condition is the faculty code is not on the table.

5.4 EXERCISE

1. Create a *stored procedure* to count the number of data in a table!
2. Create a *stored procedure* to calculate mean, max, min, and sum of a column with a numeric data type in a table!
3. What are the max number and the min number of parameter in *stored procedure*?

CHAPTER VI
STORED ROUTINE: FUNCTION

6.1 LEARNING OUTCOMES

1. Students understand the principle of a *function* in *MySQL*
2. Stuends are able to create a *function* in *MySQL* DBMS
3. Stuends are able to explain when to use a *function* in *MySQL*

6.2 THEORY

Stored function / user defined function (UDF) is a facility of MySQL starting with version 5.0. The difference between the procedures and functions in MySQL is similar to procedures and functions in programming languages. The function of a UDF is to return a scalar value and can be called in any other procedure or function statement, while procedure invoked through the command CALL and can return a value through a variable output.

6.3 STUDENTS ACTIVITY

Before creating a *function*, first make sure that the command line is already using a *database* (*use nama_database*).

1. *Create Function*

   The general form of *create function* command is:

```
CREATE FUNCTION function_name (function_parameter)
RETURNS tipedata isi_fungsi
```

   Description

   function_parameter : variable_name data_type

   isi_fungsi : *SQL* function *statements*

   Example :

```
Delimiter | (enter)
   CREATE FUNCTION coba (isi char(20))
   RETURNS concat('hello', ',s,'!'); |
(enter)
   Select hello ('world')| (enter)
   Alter Function
```

This function is can be used only to change the name of the procedure or function and changing its characteristics. Here's the general syntax form:

```
ALTER {procedure | function} nama [characteristic ...]
```

Alter function can not be used to change the program bodies, function parameters, and others. Therefore, if we want to replace a program agency procedure can be done by removing the procedure you want to change and then create a new procedure again.

2. *Drop Function*

   This following syntax can be used to delete a stored procedure or function that already exist in a database.

```
DROP {procedure | function} [if exist] nama
```

   if exist can be added to avoid the emergence of MySQL error messages for procedure / function specified does not exist.

3. *Show Function*

   If it's needed to see the syntax of a procedure or function that already exists, can be done with the following command:

```
SHOW CREATE {procedure | function} name
```

4. Call *Function*

   Function *call* can be used to call a *procedure*. *Procedure* or *function* can be called in SQL language or called directly using this function. Here is the syntax:

```
CALL nama procedure atau function
Badan program
BEGIN...END
Bentuk umumnya :
[begin_label : ] begin
statement1;
statement2;
...
statementN;
end [end label];
```

   The body of program is allowed to load more than one program block (begin ... end). However, note labels begin and end its label must be the same (if labeled).

5. *Declare*

Declare is defined in *begin...end* block and in the beginning of the line before other statements. Its function is to define local variable that apply in relevant block. There are three kind of *declare*, the first is *declare* to declare a variable; the second is *declare handler*; and the last is *declare cursor*. Here is the general form to declare a variable:

declare variable_name <data type> [DEFAULT value].

Example:

declare sum *int default* 5 or *declare* i,j *int*.

Here are some examples of *stored function*:

Here's how to show the number of data of each class from table *tb_siswa,* with nim, name and class as its fields.

```
DELIMITER $$
CREATE FUNCTION sf_tampilkan_siswa (siswa int)
RETURNS INT
      BEGIN
DECLARE jml INT;
SELECT COUNT(*) AS jml_kelas INTO jml
FROM tb_siswa WHERE kelas = siswa;
RETURN jml;
END $$
```

To show the data, use this following syntax:

```
SELECT sf_tampilkan_siswa ('2');
```

The data shown will be the number of students in the class 2.

Using IF

```
DELIMITER &&
CREATE FUNCTION tampil_gaji(gaji float(8,2))
RETURNS VARCHAR(20)
   BEGIN
      DECLARE uang_gaji varchar(20);
            if gaji < 4000 then set uang_gaji = 'Gaji Rendah';
                  else set uang_gaji = 'Gaji Tinggi';
                  end if;
    RETURN uang_gaji;
    END &&
DELIMITER ;
```

The following syntax can be used to show the data:

```
SELECT nama_depan, nama_belakang, tampil_gaji(gaji) FROM pekerja;
```

Then that will appear is the first name last name trade workers and the workers' salaries. If salaries above 4000 it will appear as a highly-paid workers, on the contrary if below 4000 then the salary is low.

Using CASE to handle conditions:

```
DELIMITER ^^
     CREATE FUNCTION sfFakultas (fakID CHAR(2)) RETURNS VARCHAR(20)
     BEGIN
          DECLARE namaFak VARCHAR(20);
          CASE fakID
               WHEN 'PA' THEN SET namaFak = 'Matematika dan Ilmu
               Pengetahuan                                    Alam';
               WHEN 'TK' THEN SET namaFak = 'Teknik';
               ELSE SET namaFak = 'Fakultas Tidak Terdaftar';
          END CASE;
          RETURN namaFak;
     END;
DELIMITER ^^
```

The syntax below can be used to show the data:

```
SELECT kode_jurusan, nama, sfFakultas(kode_fakultas) FROM jurusan;
```

It will show the department code, department name the department code follows the code in accordance with its CASE condition.

6.4 EXERCISE

1. Identify and explain the differences between stored procedures and user defined function!
2. Could a *function* have more than one return value?
3. Make a *function* to search student's NIM with highest IPK in academic database!

CHAPTER VII
TRIGGER

7.1 LEARNING OUTCOMES

1. Understand *trigger*.
2. Knowing syntax used in creating trigger in MySQL.

7.2 THEORY

Trigger is a database object such as a command line (statement) associated with the table and will be active / executed automatically when a particular event occurs in the table. Trigger can not be associated with a temporary table or a view. Here is the general syntax to create a trigger in MySQL DBMS:

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    TRIGGER trigger_name trigger_time trigger_event
    ON tbl_name FOR EACH ROW trigger_body
```

Description:

trigger_time is the time that is defined to activate the trigger. Its value can be either BEFORE or AFTER to indicate that the active trigger before or after a change in the table referred to as a trigger trigger.

trigger_event indicates the type of statement that makes the trigger runs. Can be one of these following statements:

1. INSERT: *Trigger* will be activated when there is a new line inserted into the table; as an example of the exercise of INSERT, LOAD DATA, or REPLACE.
2. UPDATE: *Trigger* will be activated when a row is modified; for example when executed UPDATE statement.
3. DELETE: Trigger will run when a row is deleted from the table; for example, if carried out executions DELETE and REPLACE statements. Please also note that DROP TABLE and TRUNCATE TABLE will not activate a trigger because it does not include using the DELETE statement.

SQL statement that follows the FOR EACH ROW defines the contents of the trigger, the SQL command line to be executed each time the trigger is triggered. Trigger will be activated / triggered when each line (row) affected by the statement triggers run. Command on the contents of the trigger starts with the keyword BEGIN and ends with END, unless the contents of only one command on the command line is not required to include the keywords BEGIN and END.

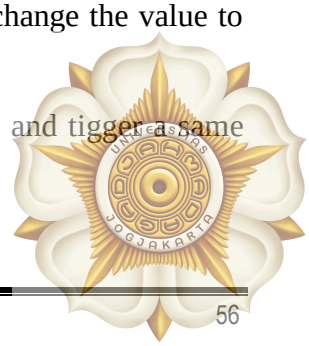
```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

The above example makes trigger named upd_check which will be active when there is SQL statements that make changes to the data in the table account. When there UPDATE statement is executed, the following line of code between BEGIN and END will be executed. As has been previously diunkap, keywords BEGIN ... END is used when the SQL statements that are executed more than one trigger.

In the BEGIN ... END syntax conditional and looping can be used. As well as stored procedures and stored functions, if you want to include more than one statement MySQL then have to redefine the statement delimiter in MySQL so that the characters; can be used in defining the trigger without breaking the flow of the trigger.

7.3 EXERCISE

1. Create a trigger that will store the data that removed any deletion of data rows in a table!
2. Create a trigger that would limit to a minimum of 0 and a maximum of 100 when the UPDATE or INSERT on a column of numeric data type! Example: if there is a data input -10 then the data entered will be 0, and if there is no input or change the value to more than 100, the value of 100 that will be saved.
3. Could two triggers (or more) on a database have a same trigger time and tigger a same event?



CHAPTER VIII
TRANSACTION

8.1 LEARNING OUTCOMES

1. Be able to understand the principles of transaction and when the transaction needs to be used.
2. Knowing the syntax used in the transaction.

8.2 THEORY

Transaction Transaction is a set of SQL statements that are treated as a single unit by Relational Database Management System (RDBMS). SQL statements are used as one unit as the transaction is the statements that have a specific purpose and interdependence between the statements of the other statement. Addiction is meant transaction is only considered successful if the whole statement was successfully executed.

An RDBMS that supports transaction must satisfy a rule called ACID rules. ACID (Atomicity, Consistency, Isolation, Durability) rules contain basic principles that must be met for the RDBMS can implement a secure transaction.

Atomicity: a transaction should be treated as one entity, and any commands that make up the transaction must be successfully executed entirely then the execution of the transaction is considered a success. When a failure occurs, the system must be restored to the state before the transaction is executed.

Consistency: when the transaction completes, the system must remain consistent with no mistakes, with each constraint suggests integrity of the tables in the database. Each rule contained in the database prior to the exercise of the transaction must be fulfilled after the executable transaction.

Isolation: changes resulting from the transaction must be isolated from the other transaction when the transaction is running. If a transaction is not isolated alias can be interrupted by the transaction or any other SQL statement, the consistency of the RDBMS is not guaranteed.

Durability: when a transaction completes, the changes made must be remembered / recorded by the system even in the event of system failure. So there is no guarantee of error occurs when a transaction is not successfully executed.

In order to perform the transaction, autocommit feature should be in an inactive state. In the standard setting MySQL, autocommit are in an active condition. This may imply that when a SQL statement is executed to change (update) table then MySQL will save it to disk so that changes occur permanently. To disable autocommit can be done two ways: implicitly (by running transaction) or explicitly (by turning off autocommit feature).

Statement-SQL statement that is used to control the transaction include BEGIN, COMMIT, ROLLBACK, and SAVEPOINT. Statement BEGIN or START TRANSACTION is used to initiate a transaction or implicitly used to disable autocommit. COMMIT statement is used to execute the transaction so that changes are made to be remembered / recorded by the system. ROLLBACK statement is used to cancel the whole transaction has been made. SAVEPOINT statement is used to create a secure point to the name of a specific identifier so that it can be done ROLLBACK to the safe point without aborting the entire transaction. Explanation statement SQL-statement can be illustrated by Figure 1.

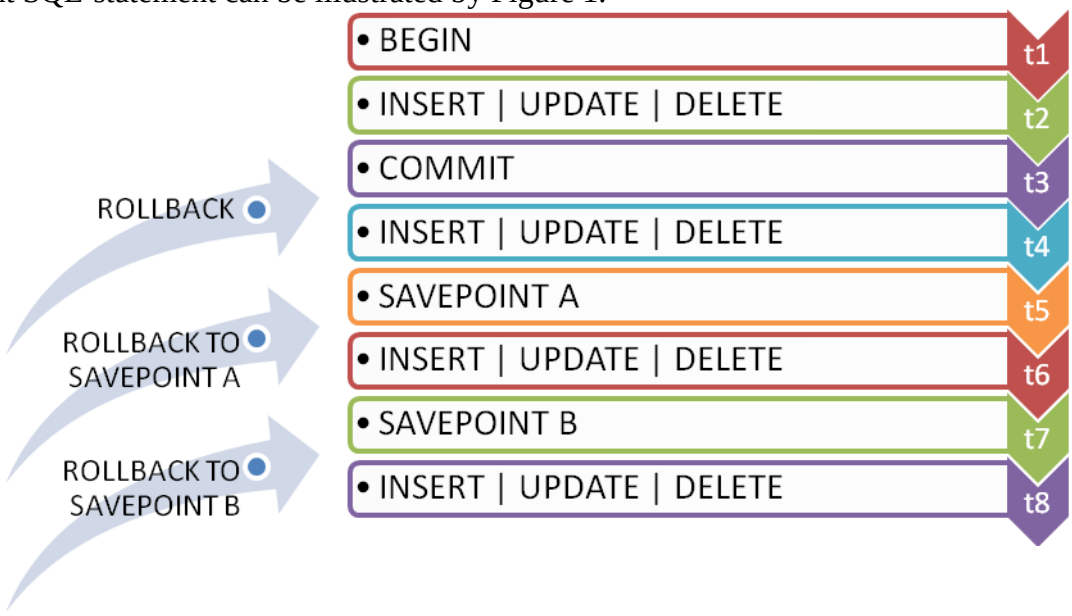


Fig. 8. 1 SQL statements to control *Transaction*

On the table type that uses a storage engine that supports transactions such as InnoDB, can not be nested transactions, which means that if the BEGIN statement is executed before the transaction is completed (COMMIT Statement), the COMMIT statement is automatically executed first. Also on the table type, transaction arranged by the client, so that if at the time the transaction occurs between a client connection problem with the server, then MySQL will automatically cancel the entire transaction has been made, the same as the concept ROLLBACK..

8.3 STUDENTS ACTIVITY

1. Scenario A: Manipulating *Autocommit* feature.
   a. Step 1: Checking the state of *autocommit*

      Conditions autocommit default value is 1, which means that the transaction will be recorded / stored in the system automatically. Check and record the condition autocommit by writing the following syntax.

      ```
      SELECT @@AUTOCOMMIT;
      ```

   b. Step 2: Change the value of *autocommit*

      Change the default autocommit condition with a value of 0 using the following syntax:

      ```
      SET AUTOCOMMIT = 0;
      ```

   c. Step 3: Make sure the state of *autocommit* has been changed.

      Recheck and make sure autocommit after its value has changed in accordance with the conditions where it is possible to do statement COMMIT, ROLLBACK, and SAVEPOINT.

2. Scenario B: *statement* COMMIT
   a. Step 1 : Open two sessions of *MySQL* client

      Open a command prompt / terminal and log in using the MySQL root user, we call this session as the first client session. Open a command prompt / new terminal and log on using another user, the session is what we call the second client session.

   b. Step 2 : Checking the same table's data on both the MySQL client session

      Enter into the database with the syntax of the USE nama_basisdata then, look at the table with the syntax SELECT * FROM table_name on both the client session.

   c. Step 3: Start the transaction and add a new row to the first client session

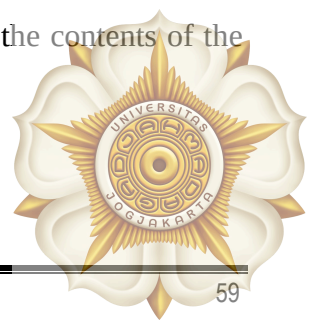
      Start the transaction on the first client session with the syntax BEGIN or START TRANSACTION then add a new row to the table with the syntax INSERT INTO table_name VALUES (data); then take a look at the contents of the table change.

   d. Step 4: Viewing the tables on both the client session and recorded the results

      Use the syntax SELECT * FROM *table_name* on both the client session and then record the results.

   e. Step 5: Perform COMMIT statement at the first client session

      Write the COMMIT syntax on the client session first and then look at the contents of the table change.



f. Step 6: Viewing the tables on both the client session and recorded the results 6

   Gunakan kembali sintak SELECT * FROM *nama_tabel* pada sesi klien kedua kemudian catatlah hasilnya.

g. Step 7: Comparing the results in Step 4 and Step 6 and conclude utilization COMMIT statement

   Compare the contents of the table in Step 4 (before COMMIT) and Step 6 (after COMMIT) and then analyze and summarize the use of the COMMIT statement

3. Scenario C: *Statement* ROLLBACK

   a. Step 1: View and record the contents of the table before the transaction

      Before the transaction begins, view and record the contents of the table prior to the syntax SELECT * FROM table_name;.

   b. Step 2: Start transaction and modify the contents of certain columns in the table

      Start the transaction with BEGIN syntax, and then modify the contents of one of the columns in the table with the syntax UPDATE table_name SET column_name = isi_kolom_baru WHERE kolom_primary_key = isi_kolom_yang_akan_diubah; then take a look at the contents of the table changes that occur.

   c. Step 3: Delete specific rows in the table

      Remove one row in the table with DELETE FROM table_name WHERE syntax kolom_primary_key = isi_kolom_yang_akan_dihapus; then look at the contents of the table changes that occur.

   d. Step 4: Perform a ROLLBACK statement

      ROLLBACK syntax Write the contents of the table and then look at the changes that occur

   e. Step 5: Compare the contents of a table before and after a ROLLBACK statement is executed and concluded utilization ROLLBACK statement.

      Compare the contents of a table before and after the ROLLBACK statement ROLLBACK statement later analysis and conclude utilization ROLLBACK statement..

4. Scenario D: *Statement* SAVEPOINT

   a. Step 1: View and record the contents of the table before the transaction

      S Before the transaction begins, view and record the contents of the table prior to the syntax SELECT * FROM table_name;..

   b. Step 2: Start transaction and modify the contents of certain columns in the table

      Start transaction with START TRANSACTION syntax, and then modify the contents of one of the columns in the table with the syntax UPDATE table_name SET column_name = isi_kolom_baru WHERE kolom_primary_key = isi_kolom_yang_akan_diubah; then take a look at the contents of the table changes that occur..

c. Step 3: Delete specific rows in the table.

   Remove one row in the table with DELETE FROM table_name WHERE syntax kolom_primary_key = isi_kolom_yang_akan_dihapus; then look at the contents of the table changes that occur..

d. Step 4 : Perform SAVEPOINT statement with the name A

   Write the statement SAVEPOINT A; which means that it has been made safe point with the name/identifier A.

e. Step 5: Add a new row to the table

   Add a new row to the table with the syntax INSERT INTO table_name VALUES (data); then take a look at the contents of the table change

f. Step 6: Doing the SAVEPOINT statement with the name B

   Write the statement SAVEPOINT B; which means that it has been made safe point with the name / identifier B.

   Step 7: Perform TO SAVEPOINT A ROLLBACK statement

   Write the syntax ROLLBACK TO SAVEPOINT A; then view and record the contents of the table after the syntax executed.

g. Step 8: Summing utilization SAVEPOINT statement.

   Based on some of these steps, simpulkanlah utilization SAVEPOINT statement..

8.4 EXERCISE

1. Combine the use of statements COMMIT, SAVEPOINT, ROLLBACK, and ROLLBACK TO SAVEPOINT in a transaction!
2. Do the transaction with three or more sessions a client! What happens if there is a client to make changes to the data when a client transaction others first did not end the transaction and / execute COMMIT or ROLLBACK?

MySQL INSTALLATION PROCESS

In October 2005, MySQL AB MySQL as the manufacturer has finally announced that the MySQL 5.0 was released. Until now, the most recent version of MySQL is still being developed. To find the latest version of MySQL has been released, please visit his official website at http://www.mysql.com.

Download

To be able to install and use MySQL, we must first download the software MySQL website http://www.mysql.com. Next, select the appropriate platform for the installation.

Installing MySQL Step-by-Step

After done downloading the MySQL software, the following steps to install MySQL: (In this book, using MySQL 5.0)

1. After completion of the installation package MySQL for Windows. Next go to the directory where to save the MySQL installation package. Run the file called **setup.exe**.

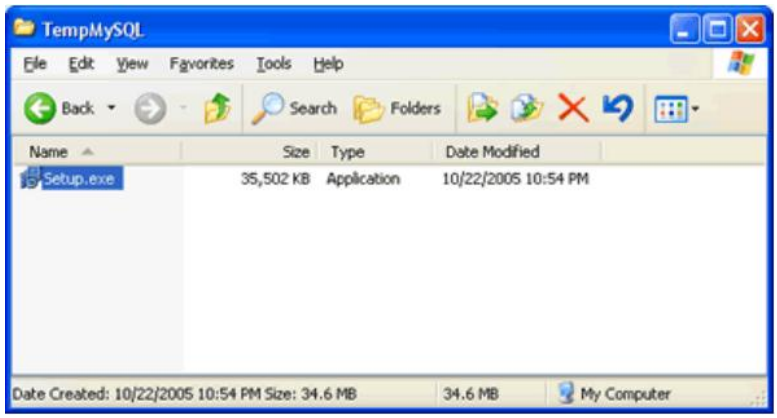


Fig 1 Running MySQL setup file

2. To continue the installation process and click **Next**>

Fig 2 The beginning MySQL installation process

3. The next step is to choose the type of installation (setup). There are 3 types of installations, namely, Typical, Complete, and Custom. Type typical mean putting common features of MySQL. Type Complete mean putting all the features of MySQL, of course, would require a larger disk space. Costum mode means you can choose what programs will be installed. In this tutorial, select the type of typical.

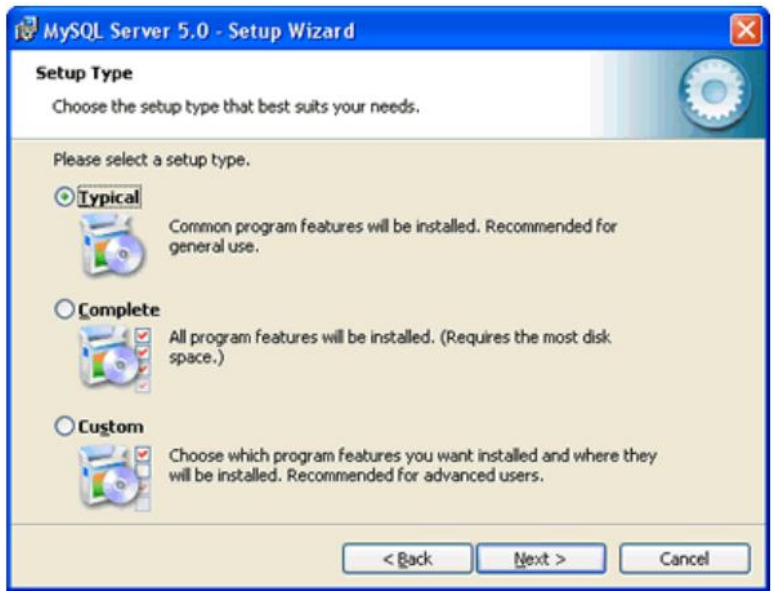


Fig 3 Choosing type of installation setup

4. The next step is to confirm the MySQL installation directory is C: \ Program Files \ MySQL \ MySQL Server 5.0 \. Installation directory can not be changed because it uses typical installation type. Click Install to begin the installation.

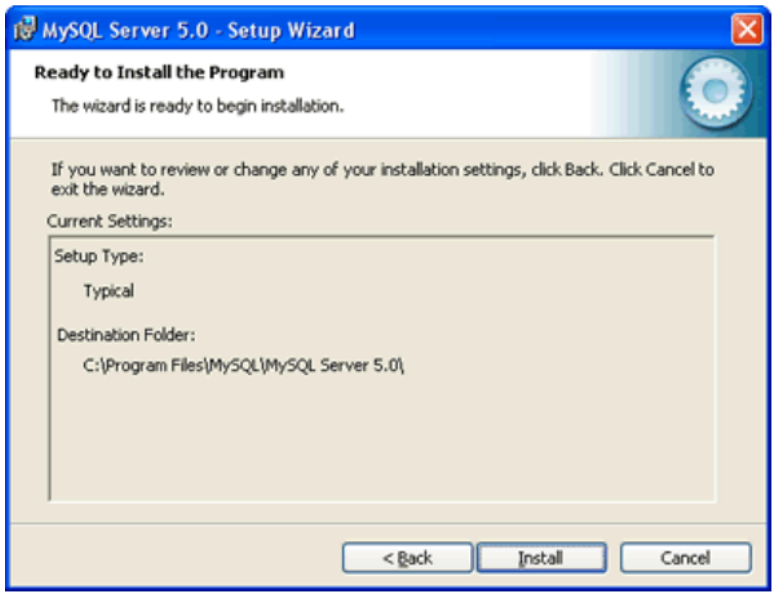


Fig 4 Installation location of MySQL

Then the installation process will take place.

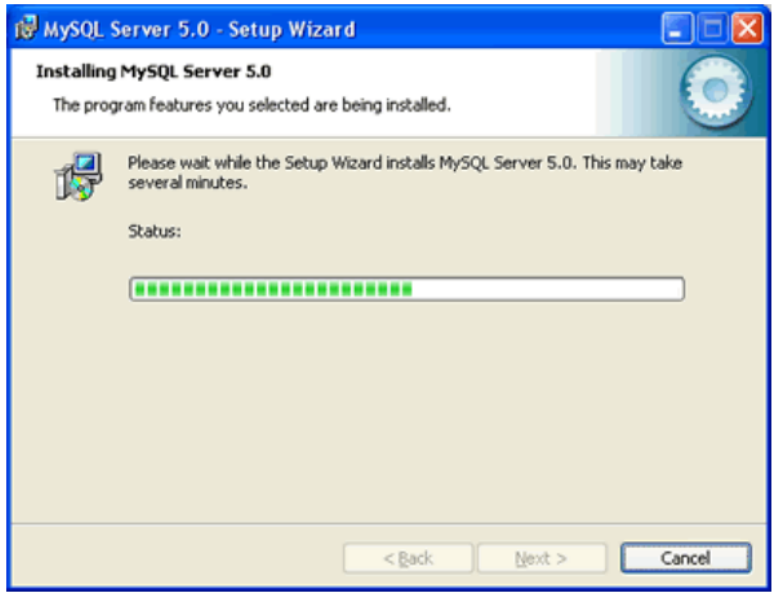


Fig 5 MySQL installation process

5. Then, MySQL offers you to join for free a member in MySQL.com. If you are interested in joining, please select create a new free account MySQL.com. If you've ever had an account on MySQL.com, please select the second row (Login to MySQL.com), or Skip Sign-Up. In this module we select only the third row. (Skip Sign-Up). Then continue by pressing the Next button>.

Fig 6 Sign-Up MySQL Account

6. Wait a while until the installation is complete. Then (recommended) to proceed with the MySQL server configuration (Configure the MySQL Server now). And you can press the Finish button on this stage (but will continue with the configuration process).

Fig 7 Confirmation of MySQL server configuration

7. Click **Next** to continue with configuration

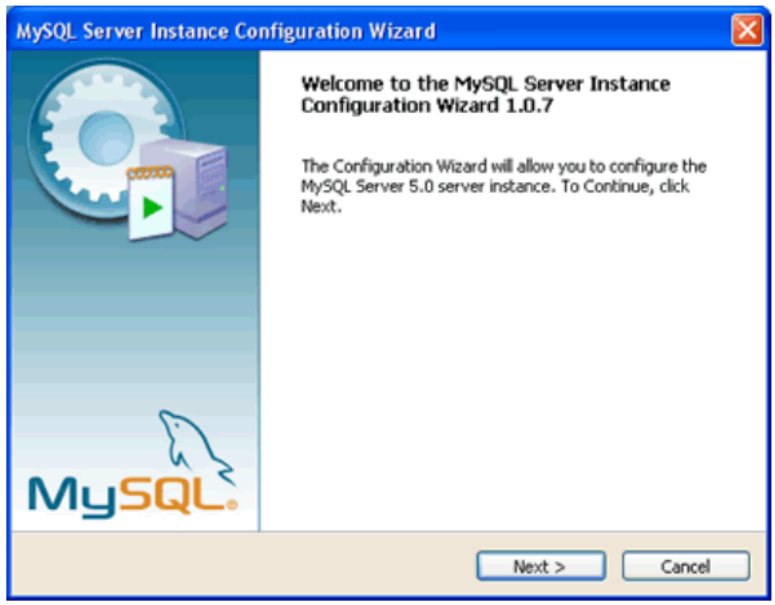


Fig 8 Firs step of configuring MySQL server

8. There are two configuration options, Detailed Configuration and Standard Configuration. In this module, we use the Detailed Configuration, because it makes the server becomes more optimal. Click on the Next> button to continue.

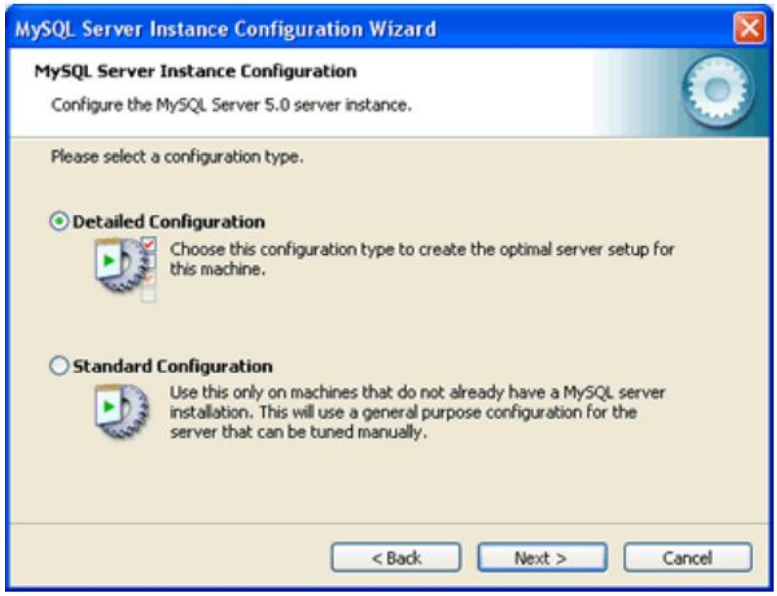


Fig 9 Configuration type of MySQL server

9. The next step, there are 3 options, namely server types Developer Machine, Server Machine and Dedicated MySQL Server Machine. In this module we choose Developer Machine. Thus MySQL will not monopolize or burdensome work of other systems on the computer. Click on the Next> button to continue.

Fig 10 Server type

10. The next step is choosing the usability of MySQL. There are three options, Multifunctional Database, Transactional Database Only or Non-Transactional Database Only. In this module, use the Multifunctional Database. Click on the Next> button to continue.

Fig 11 Database usability choices

11. This stage displays the drive to be installed InnoDB. Starting MySQL 4.1.x, InnoDB has been included in a single package. InnoDB allows you to do a transaction in the database. Where the MyISAM table (which is the default table type in MySQL) does not support the Transaction. For this look use a selection of existing standards, and proceed by clicking the Next button.

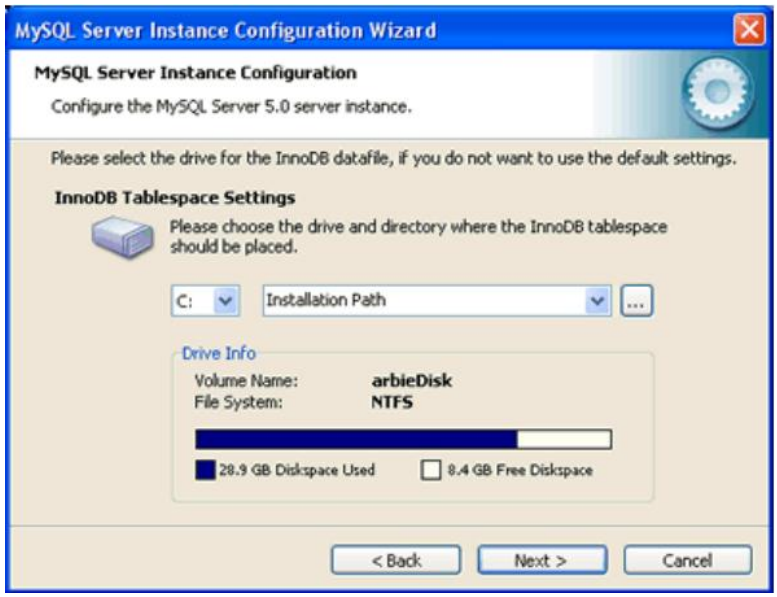


Fig 12 Setting InnoDB

12. The next step is to choose the maximum number of connections allowed on the MySQL server, which Decission Support (DSS) / OLAP, Online Transaction Processing (OLTP), or Manual Setting. In this module been Decission Support (DSS) / OLAP. Click Next> to continue the configuration.

Fig 13 Connection setting on MySQL server

13. The next step, determine whether to take advantage of TCP / IP to access the MySQL server or not. Here there is little difference compared with the installation process MySQL 4.1.x. There is an additional option Enable Strict Mode. Enable the option Enable Strict Mode in accordance with the amount recommended by the system. Click Next> to continue.

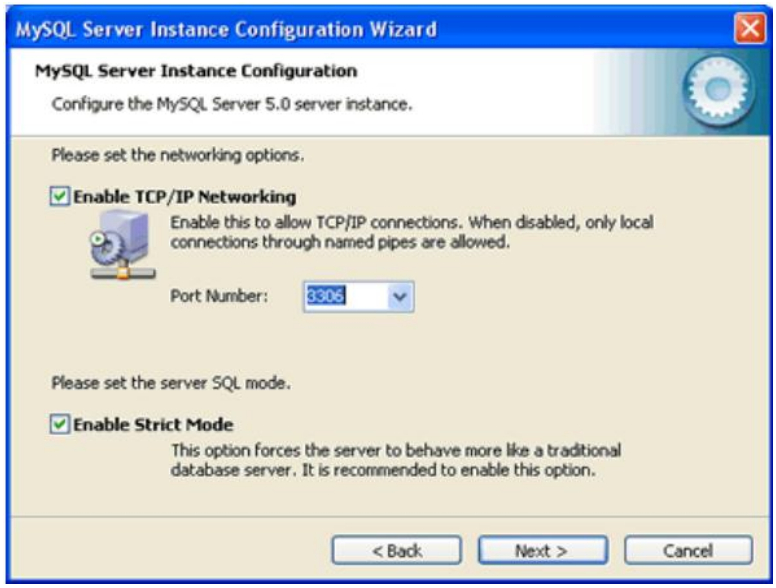


Fig. 14. Setting TCP/IP Networking

14. The next choice regarding the type of characters used. Select standard characters (*Standard Character Set*), then Next.

Fig 15. Setting default character set

15. The next screen, enable the Install as a Windows Service and Launch the MySQL Server automatically. With this option, every time the computer is turned on, the program automatically MySQL server will dijalankan.Begitupun should enable the option Include Bin directory in the Windows Path. MySQL programs are usually stored in the directory C: \ Program Files \ MySQL \ MySQL Server 5.0 \ Bin. By enabling this option, then you can run a MySQL program from a DOS / Command Prompt.

Fig 16. Setting Windows Service

16. The next display is the MySQL server security issues. You should give a special password as Root, and not allowing people to enter your system without a password. Then activate the Modify Security Settings option and enter your root password carefully. However, turn off selection Create An Anonymous Account. Thus not careless people can log in using your MySQL server.

    One more thing, it is advisable also turn off the option Enable root access from remote machines. This is to prevent cracks that can be entered by people who are not responsible for sneaking into our system. Continue by pressing the Next button> .

Fig 17. Setting Security

17. When you are sure to continue, click on the Execute button. And you can relax a little bit while waiting setting process is completed.

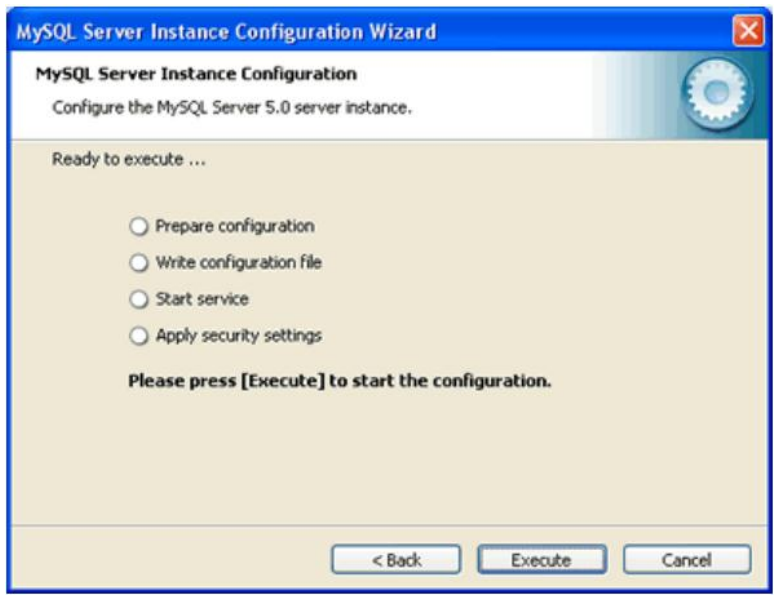


Fig 3.18 Final step of configuration

18. If there are no obstacles whatsoever, then the whole process of installing and setting MySQL 5.0.x program has been completed. Click the **Finish** button.

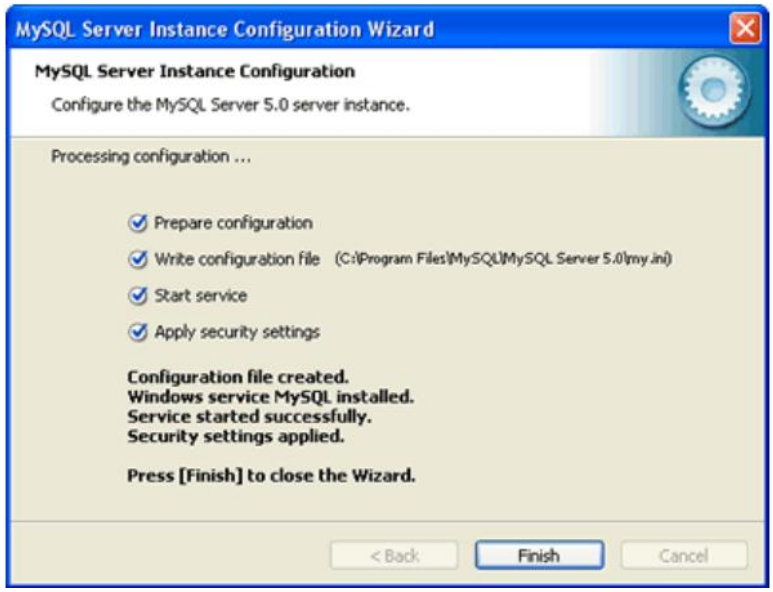


Fig 19 Final step of MySQL installation

19. Now please do a test to access MySQL from the Command Line. type:

    C:>mysql -u root -p

    Enter password:*******

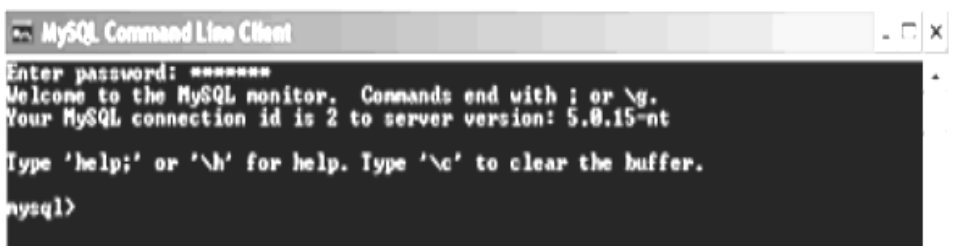


Fig 20 Accessing MySQL via Command Line

20. Also try to include some MySQL commands like show databases, and so on. If it can function properly. MySQL 5.0.x installation has been completed.