NMK20303 Database Management System LAB MODULE 3

SQL Application

FACULTY OF ELECTRONIC ENGINEERING

TECHNOLOGY **Universiti Malaysia Perlis**

Objectives

1. To understand the Structured Query Language that query or data management language.
2. To use the MySQL in the application of database management system.
3. To apply the MySQL in the database application.

Equipment/software

1. XAMPP Apache + MariaDB + PHP
2. Netbeans

1 SQL – SELECT STATEMENT

The advantages of database system compare to other data storing (e.g. text file of directory) is the ease of data searching. By separating the data into rows and columns, data can be refer easily either by direct searching to the record or search the related records. In the last laboratory we have look on how to create a database, create a table and inserting data. In this laboratory we will continue with how to view data and how to query a data.

1.1 Select Statement

SELECT <column> FROM <table> WHERE <filter_condition>

Command line divide into three parts:-

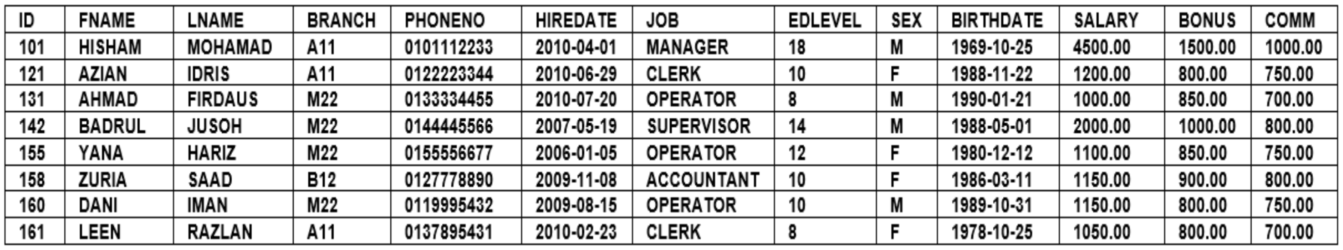
1. **SELECT <column>**
   Command tell the database the name of the column which holds the query data.

2. **FROM <table>**
   Command tells the database the source table of the data column.

3. **WHERE <filter_condition>**
   The **WHERE** clause allow you to filter the query in order to limit the number of rows return. Commands tell the database to display data that had fulfilled the filter condition only. From all three parts above, only commands **SELECT** and **FROM** are compulsory to execute data selecting. Command where only needed if you want to select rows which have certain criteria.

Table 3.1 : EMPLOYEE's Table

EMPLOYEE



ID	FNAME	LNAME	BRANCH	PHONENO	HIREDATE	JOB	EDLEVEL	SEX	BIRTHDATE	SALARY	BONUS	COMM
101	HISHAM	MOHAMAD	A11	0101112233	2010-04-01	MANAGER	18	M	1969-10-25	4500.00	1500.00	1000.00
121	AZIAN	IDRIS	A11	0122223344	2010-06-29	CLERK	10	F	1988-11-22	1200.00	800.00	750.00
131	AHMAD	FIRDAUS	M22	0133334455	2010-07-20	OPERATOR	8	M	1990-01-21	1000.00	850.00	700.00
142	BADRUL	JUSOH	M22	0144445566	2007-05-19	SUPERVISOR	14	M	1988-05-01	2000.00	1000.00	800.00
155	YANA	HARIZ	M22	0155556677	2006-01-05	OPERATOR	12	F	1980-12-12	1100.00	850.00	750.00
158	ZURIA	SAAD	B12	0127778890	2009-11-08	ACCOUNTANT	10	F	1986-03-11	1150.00	900.00	800.00
160	DANI	IMAN	M22	0119995432	2009-08-15	OPERATOR	10	M	1989-10-31	1150.00	800.00	750.00
161	LEEN	RAZLAN	A11	0137895431	2010-02-23	CLERK	8	F	1978-10-25	1050.00	800.00	700.00

Syntax:

```
mysql>
CREATE TABLE EMPLOYEE
(ID VARCHAR(15),
FNAME VARCHAR(20),
LNAME VARCHAR(20),
BRANCH VARCHAR(20),
PHONENO INT(15),
HIREDATE DATE,
JOB VARCHAR(20),
EDLEVEL VARCHAR(40),
SEX VARCHAR(6),
BIRTHDATE DATE,
SALARY INT(15),
BONUS INT(15),
COMM INT(15)) ;
```

From the above command:-
1. If you want to display all the employee id (ID) from table EMPLOYEE:-

```
Mysql >> SELECT ID FROM EMPLOYEE;
```

2. And if you want to list all the phone number (PHONENO):-

```
Mysql >> SELECT PHONENO FROM EMPLOYEE;
```

1.2 SQL Statement: SELECT <n_column> FROM <table>

To choose more than one column in one table, you have to list each column you want to display after command SELECT and separate each column with a comma like below:-

```
SELECT <attribute 1>, <attribute 2>, ..., <attribute n> FROM <table>
```

Syntax:
Mysql >> SELECT FNAME, LNAME, BRANCH FROM EMPLOYEE;

The result of command line above, system will display contain of columns FNAME, LNAME, and BRANCH.

1.3 SQL Statement: Simple SELECT

Inside the table Employee used above contains 13 different columns, in certain case we may need to display all the columns.

Syntax:
Mysql >> SELECT ID, FNAME, LNAME, BRANCH, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, COMM FROM EMPLOYEE;

Writing a long command line may cause a spelling error or missing column name. Solution to these problems is by using "simple SELECT statement".

SQL Statement,
SELECT * FROM <table_name>

Syntax:
Mysql >> SELECT * FROM EMPLOYEE

The command will display all the columns inside the table. The symbol asterisk '*' representing all the columns inside the table EMPLOYEE.

2 ROW SELECTION

2.1 Arithmetic Comparison Operator In Filter Condition

The comparison operators compare one value to another, based on the relational operators. The simplest form of this predicate is represented by two expressions connected by one operator.

1. Example, to find all the rows contain data of workers work in the branch (BRANCH) numbered 'A11' from table 'EMPLOYEE';

Mysql >> SELECT * FROM EMPLOYEE WHERE BRANCH='A11';

2. Example, to find all the rows contain data of workers who had 18 years of education level (EDLEVEL) from table 'EMPLOYEE';

Mysql >> SELECT * FROM EMPLOYEE WHERE EDLEVEL>18;

3. Example, to list all the rows of workers data except workers that work as DESIGNER;

Mysql >> SELECT * FROM EMPLOYEE WHERE JOB <> 'OPERATOR';

2.2 Predicates in Filter Condition

A predicate in a **WHERE** clause is a search condition that may be true, false, or unknown. A predicate use operators, expressions, and constants to specify the search condition. The main predicates are, **BETWEEN**, **NULL**, **EXISTS**, **LIKE**, **IN**.

The LIKE Predicate

This predicate searches for string of text that belongs to the specified default. It can only use in **char** or **varchar** data type:-
Example, to find all the rows that contain workers last name (LNAME) starting with letter 'S' from table EMPLOYEE:-

Syntax:
Mysql >> SELECT * FROM EMPLOYEE WHERE LNAME LIKE 'S%';

String pattern 'S%' tell the SQL system to find all the rows which start with letter 'S' and the symbol '%' will let SQL system to take any character following after letter 'S'. Table 3.2 show the list of string pattern.

Table 3.2: List of String Pattern

String Pattern	**Comment**
% [percent]	Any letter, unlimited length of character
- [underscore]	Any letter, length 1 letter
S%	Any data that start with the letter S
%N	Any data that end with the letter N
%A	Any data that contain letter A at any position
'S____	Any data that start with letter S and have 5 letters long
S%T%	Any data that start with letter S and have letter T at any position
S__T%	Any data that start with letter S and have letter T at fourth position
%A%E_	Any data that contain letter E at 2 position from behind and lead by letter A at any position

3 COMBINING FILTER CONDITION

There is a time when we need to state more than one filter condition in SQL command to get more precious data.

3.1 Boolean Operator: AND

True when both expressions are true. Basic syntax:-

SELECT <column_name> FROM <table_name> WHERE <column_name> <operator> <filtercondition> AND <column_name> <operator> <filtercondition>

Example:

To find all the rows that contains data of workers position (JOB) as ‘OPERATOR’ and have 12 year education level (EDLEVEL) from table EMPLOYEE;

Syntax:

Mysql >> SELECT * FROM EMPLOYEE WHERE JOB = ’OPERATOR’ AND EDLEVEL > 12

3.2 Boolean Operator: OR

True when one expression is true. Basic syntax:-

SELECT <column_name> FROM <table_name> WHERE <column_name> <operator> <filtercondition> OR <column_name> <operator><filtercondition>

Example:

To find a rows that contain data workers first name (FIRSTNAME) stating with letter D or last name is ‘SMITH’;

Syntax:

Mysql >> SELECT * FROM EMPLOYEE WHERE FIRSTNAME LIKE ‘D%’ OR LASTNAME=’IDRIS’

3.3 Joining Boolean Operator

In searching more specific data, we can joining several filter condition using several Boolean operator as example,

Syntax:

Mysql >> SELECT * FROM EMPLOYEE WHERE JOB=’MANAGER’ OR SALARY >2000 AND EDLEVEL>=15

4 SET OPERATION FILTER CONDITION

4.1 Set Operation: IN and NOT IN Statement

SQL IN statement can be used with where clause to list a set of matching records of a table. We can use SQL in query with NOT combination also to filter out some records.

Syntax:

Mysql >> SELECT <column_name> FROM <table_name>
WHERE <column_name> IN (<column_list>) ;

Example:

Syntax:

Mysql >> SELECT * FROM EMPLOYEE WHERE FIRSTNAME IN ('AZIAN', 'BADRUL', 'ZURIA', 'DANI') ;

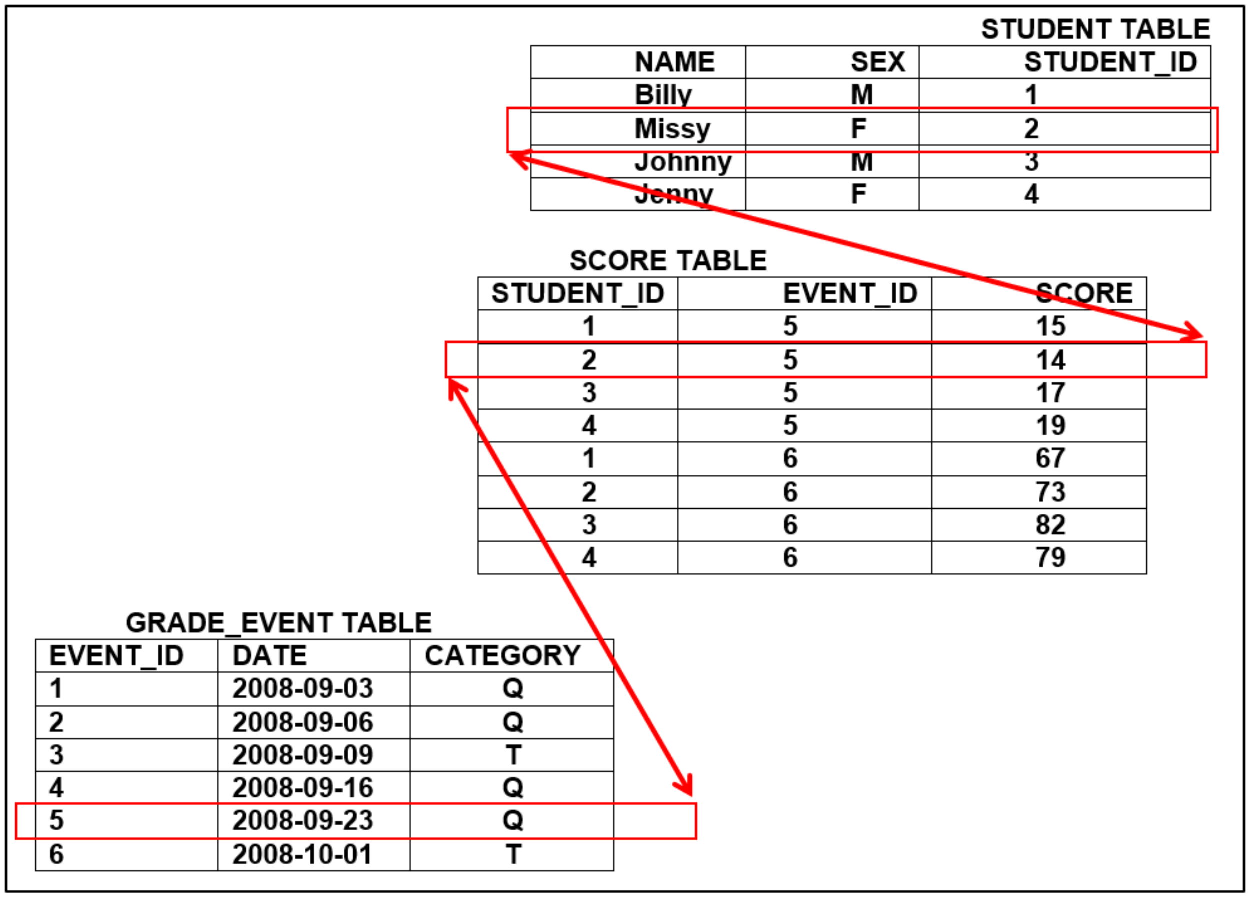
NOT IN statement is a compliment of **IN** statement. We can apply **NOT IN** SQL query also likes below to remove some records in display.

Syntax:

Mysql >> SELECT * FROM EMPLOYEE
WHERE FIRSTNAME NOT IN ('AZIAN', 'BADRUL', 'ZURIA', 'DANI') ;

5 JOINING MULTIPLE TABLES AND SETTING KEYS

Figure 3.1 showed the joining operation between student table, score table, and grade_event table. In student table, primary key is set to student_id and connected with score table with student_id (as foreign key). Meanwhile, grade_event table set event_id as primary key and connected to another score table as foreign key (event_id). Score table was function as enrollment table because it has two (2) foreign keys for student_id and event_id. Table 3.3



STUDENT TABLE

NAME	SEX	STUDENT_ID
Billy	M	1
Missy	F	2
Johnny	M	3
Jenny	F	4

SCORE TABLE

STUDENT_ID	EVENT_ID	SCORE
1	5	15
2	5	14
3	5	17
4	5	19
1	6	67
2	6	73
3	6	82
4	6	79

GRADE_EVENT TABLE

EVENT_ID	DATE	CATEGORY
1	2008-09-03	Q
2	2008-09-06	Q
3	2008-09-09	T
4	2008-09-16	Q
5	2008-09-23	Q
6	2008-10-01	T

showed the absence table.

Figure 3.1 SCORE, STUDENT AND GRADE_EVENT tables linked on STUDENT ID and EVENT ID

ABSENCE TABLE

STUDENT_ID	DATE
2	2008-09-02
4	2008-09-15
2	2008-09-20

Table 3.3 ABSENCE TABLE

PART 1: CREATING A NEW DATABASE

• Create a new database named <Student_Mgmt>.
• Use <Student_Mgmt>.

PART 2: CREATING A STUDENT TABLE

Create student table:-

Syntax:

```
Mysql >>
CREATE TABLE student
( name VARCHAR(20) NOT NULL,
sex ENUM('F','M') NOT NULL,
student_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (student_id)
) ENGINE = InnoDB;
```

Description :-

Values in an ENUM column need not be just a single character. The sex column could have been defined as something like ENUM('female','male') instead. If you omit the ENGINE clause, MySQL picks a default engine, which usually is MyISAM. "ISAM" stands for "indexed sequential access method," and the MyISAM engine is based on that access method with some MySQL-specific stuff added. InnoDB offers something called "referential integrity" through the use of foreign keys. That means we can use MySQL to enforce certain constraints on the interrelationships between tables.

• Score rows are tied to grade events and to students: We don't want to allow entry of rows into the score table unless the student ID and grade event ID are known in the student and **grade_event** table.

• Similarly, absence rows are tied to students: We don't want to allow entry of rows into the absence table unless the student ID is known in the **student** table.

To enforce these **constraints**, we'll set up foreign key relationships. "**Foreign**" means in another tables, and "**foreign key**" indicates a key value that must match a key value in that other table. Insert data into student table as shown in Figure 3.1

PART 3: CREATING A GRADE_EVENT TABLE

Create grade_event table with definition:-

Syntax:

```
Mysql >>
CREATE TABLE grade_event
( date DATE NOT NULL,
category ENUM('T','Q') NOT NULL,
event_id INT UNSIGNED NOT NULL AUTO_INCREMENT,
PRIMARY KEY (event_id)
) ENGINE = InnoDB;
```

• The date column holds a standard MySQL DATE value, in 'YYYY-MM-DD' (year-first) format. category represents score category.

• Like sex in the student table, category is an enumeration column. The allowable values are 'T' and 'Q', representing "test" and "quiz."

• event_id is an **AUTO_INCREMENT** column that is defined as a **PRIMARY KEY**, similar to student_id in the student table.

• Using **AUTO_INCREMENT** enables us to generate unique event ID values easily. As with the student_id column in the student table, the particular values are less important than that they be unique.

Insert data into grade_event table as shown in Figure 3.1.

PART 4: CREATING A SCORE TABLE

The score table including:-

Syntax:

```
Mysql >>
CREATE TABLE score
( student_id INT UNSIGNED NOT NULL,
event_id INT UNSIGNED NOT NULL,
score INT NOT NULL,
PRIMARY KEY (event_id, student_id),
INDEX (student_id),
FOREIGN KEY (event_id) REFERENCES grade_event
(event_id),
FOREIGN KEY (student_id) REFERENCES student
(student_id) ) ENGINE = InnoDB;
```

The score column is an INT to hold integer score values. If you wanted to allow scores such as 58.5 that have a fractional part, you'd use one of the data types that can represent them, such as DECIMAL or FLOAT.

The student_id and event_id columns are integer columns that indicate the student and event to which each score applies. By using them to link to the corresponding ID value columns in the student and grade_event tables, we'll be able to look up the student name and event date. There are a couple important points to note about the student_id and event_id columns:.

We've made the combination of the two columns a **PRIMARY KEY**. This ensures that we won't have duplicate scores for a student for a given quiz or test. Note that it's the combination of event_id and student_id that is unique. In the score table, neither value is unique by itself.

There will be multiple score rows for each event_id value (one per student), and multiple rows for each student_id value (one for each quiz and test) taken by the student.

• For each ID column, a **FOREIGN KEY** clause defines a constraint. The **REFERENCES** part of the clause indicates which table and column the score column refers to. The constraint on event_id is that each value in the column must match some event_id value in the grade_event table. Similarly, each student_id value in the score table must match some student_id value in the student table.

• The **PRIMARY KEY** definition ensures that we won't create duplicate score rows. The **FOREIGN KEY** definitions ensure that we won't have rows with bogus ID values that don't exist in the grade_event or student tables.

Why is there an index on student_id?

• The reason is that, for any columns in a FOREIGN KEY definition, there should be an index on them, or they should be the columns that are listed first in a multiple-column index.

• For the FOREIGN KEY on event_id, that column is listed first in the PRIMARY KEY.

• For the FOREIGN KEY on student_id, the PRIMARY KEY cannot be used because student_id is not listed first.

• So, instead,we create a separate index on student_id. InnoDB actually will create an index on columns.

Insert data into score table as shown in Figure 3.1

PART 5: CREATING AN ABSENCE TABLE

The absence table for recording lapses in attendance looks like this:-

Syntax:

```
Mysql >>
CREATE TABLE absence
(
student_id INT UNSIGNED NOT NULL,
date DATE NOT NULL,
PRIMARY KEY (student_id, date),
FOREIGN KEY (student_id) REFERENCES student
(student_id)
) ENGINE = InnoDB;
```

The absence table also includes a foreign key relationship, defined to ensure that each student_id value matches a student_id value in the student table. The inclusion of foreign key relationships in the grade-keeping tables is meant to enact constraints at data entry time:We want to insert only those rows that contain legal grade event and student ID values. However, the foreign key relationships have another effect as well. They set up dependencies that constrain the order in which you create and drop tables:-

• The score table refers to the grade_event and student tables, so they must be created first before you can create the score table. Similarly, absence refers to student, so student must exist before you can create absence.
• If you drop (remove) tables, the reverse is true.You cannot drop the grade_event table if you have not dropped the score table first, and student cannot be dropped unless you have first dropped score and absence.

Insert data into absence table as shown in Table 3.3.

PART 6: OTHERS SQL STATEMENT

<u>DISTINCT</u>

With DISTINCT keyword, you can eliminate the duplicated result from SELECT statement. For example, to find how many student_id of all student in score table, we use DISTINCT keyword in SELECT statement like below:

<u>Syntax:</u>
```
Mysql >> SELECT DISTINCT student_id FROM score;
```

1. Write down the output.

<u>GROUP BY</u>

If you need to find number of employee who hold each job, you can use GROUP BY clause. GROUP BY clause allows use to retrieve rows in group. Here is the query example:

<u>Syntax:</u>
```
Mysql >> SELECT count(*), student_id
FROM score
GROUP BY student_id;
```

2. Write down the output.

<u>HAVING Clause</u>

HAVING clause usually use with GROUP BY clause to selecting a particular of group.

For example:

Syntax:

```
Mysql >> SELECT count(*), student_id
FROM score
GROUP BY score
HAVING score = 73;
```

3. Write down the output.

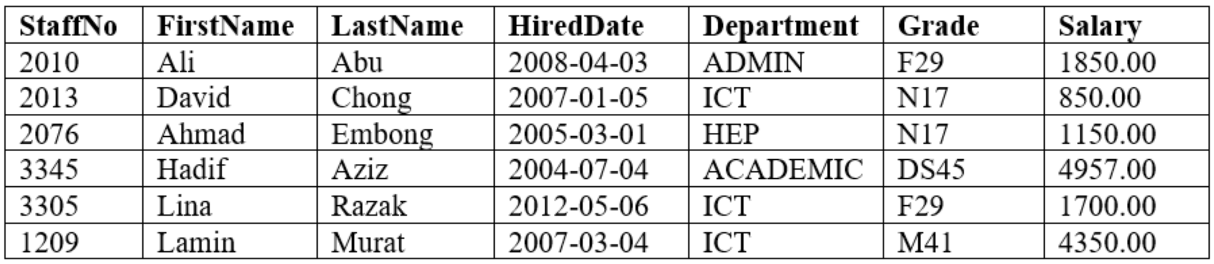
Sorting with ORDER BY

The ORDER BY clause allows you to sort the result set on one or more column in ascending or descending order. To sort the result set in ascending order you use ASC and in descending order you use DESC keywords. By default, the ORDER BY will sort the result set in ascending order. For example, to sort the name of employee on the first name and job title you can execute the following query:

Syntax:

```
Mysql >> SELECT name, sex
FROM student
ORDER BY name DESC;
```

4. Write down the output.



StaffNo	FirstName	LastName	HiredDate	Department	Grade	Salary
2010	Ali	Abu	2008-04-03	ADMIN	F29	1850.00
2013	David	Chong	2007-01-05	ICT	N17	850.00
2076	Ahmad	Embong	2005-03-01	HEP	N17	1150.00
3345	Hadif	Aziz	2004-07-04	ACADEMIC	DS45	4957.00
3305	Lina	Razak	2012-05-06	ICT	F29	1700.00
1209	Lamin	Murat	2007-03-04	ICT	M41	4350.00

Table 3.4

6 Exercise

Write down the complete SQL command to solve the tasks below:

A : Create database name STAFF and create the table 3.4.

STAFF_PROFILE

i. Syntax:
ii. Output:

B: Construct a SQL (Structured Query Language) statement to answer the following queries.

a) List all the staff 'NOT IN' ADMIN.
   i. Syntax:
   ii. Output:
b) List all the staff firstname which start with letter 'J'.
   1. Syntax:
   2. Output:
c) Find the staff who work in ADMIN and have salary below RM5000.00.
   1. Syntax:
   2. Output:

4. List all the employee except who worked in Department ICT.
   1. Syntax:
   3. Output:

5. List all the StaffNo, FirstName, LastName and Grade which salary are less than salary RM2000.00.
   1. Syntax:
   2. Output:

6. Use SQL statement to find Grade in STAFF_PROFILE,avoid duplicated result
   1. Syntax:
   2. Output:

C: Write down a simple discussion of what you have learnt from this lab session.