☑ CC105: INFORMATION MANAGEMENT

| SQL DATABASE

In the previous lesson, we focused on Data Definition Language (DDL). You learned how to create database, create tables, you even learned how to manipulate or ALTER tables. In this lesson, we will learn some Data Manipulation Language (DML) which will allow you to add data in the database, edit or update save data, delete data or even extract data from the database.

This lesson will contain examples of INSERT, UPDATE, DELETE, and SELECT statements. Follow the examples and observe the use of each queries. But to be able to go through this lesson, I want you to create a database named: *Accommodation*. SQL code is provided below.

```
CREATE DATABASE Accomodation;

CREATE TABLE customer(ID INT NOT NULL AUTO_INCREMENT PRIMARY KEY, Name
VARCHAR(50) NOT NULL, Age INT NOT NULL, Gender CHAR(1) NOT NULL,
Status VARCHAR(10) NOT NULL, Contact VARCHAR(11) NOT NULL);

CREATE TABLE Accomodation(ID INT NOT NULL PRIMARY KEY, CustID INT NOT
NULL REFERENCES customer(ID), RoomID INT NOT NULL REFERENCES
Room(RoomID), TransactionDate DATE NOT NULL, CheckIn DATE NOT NULL,
CheckOut DATE NOT NULL, Deposit DOUBLE NOT NULL, Total DOUBLE NOT
NULL);

CREATE TABLE Room(RoomID INT NOT NULL PRIMARY KEY, RoomType
VARCHAR(20) NOT NULL, RoomNo INT NOT NULL, NoBeds INT NOT NULL, Status
VARCHAR(10) NOT NULL, Rate FLOAT NOT NULL);


CREATE TABLE Billing(BillID INT NOT NULL PRIMARY KEY, TransactionID
INT NOT NULL REFERENCES Accommodation(ID), PaymentDate DATE NOT NULL,
AmountPaid DOUBLE NOT NULL);

INSERT INTO Customer VALUES(1,'John Doe', 24, 'M', 'Single',
'09234567865');
INSERT INTO Customer VALUES(2, 'Tom Bin', 35, 'M', 'Married',
'09204661129');
INSERT INTO Customer VALUES(3, 'Ann Milby', 20, 'F', 'Single',
'09188007211');
INSERT INTO Customer VALUES(4, 'Jane Mosby', 45, 'F', 'Married',
'09162003000');
INSERT INTO Customer VALUES(5, 'Peter Paul', 30, 'M', 'Separated',
'09221176500');
```

```
INSERT INTO Room VALUES(1, 'Twin', 101, 2, 'Available', 1250);
INSERT INTO Room VALUES(2, 'Deluxe', 102, 2, 'Available', 1500);
INSERT INTO Room VALUES(3, 'Suite', 103, 1, 'Occupied', 2000);
INSERT INTO Room VALUES(4, 'Single', 201, 1, 'Occupied', 750);
INSERT INTO Room VALUES(5, 'Suite', 202, 1, 'Available', 2000);
INSERT INTO Room VALUES(6, 'Deluxe', 203, 2, 'Occupied', 1500);
INSERT INTO Room VALUES(7, 'Deluxe', 301, 2, 'Available', 1500);
INSERT INTO Room VALUES(8, 'Single', 302, 1, 'Occupied', 750);
INSERT INTO Room VALUES(9, 'Twin', 303, 2, 'Available', 1250);
INSERT INTO Room VALUES(10, 'Single', 304, 1, 'Available', 750);

INSERT INTO Accomodation VALUES(1, 2, 3, '2009-01-25', '2009-01-30',
'2009-02-02', 1000, 6000);
INSERT INTO Accomodation VALUES(2, 4, 4, '2009-01-26', '2009-02-15',
'2009-02-17', 500, 1500);
INSERT INTO Accomodation VALUES(3, 1, 6, '2009-01-26', '2009-02-07',
'2009-02-10', 2000, 4500);
INSERT INTO Accomodation VALUES(4, 5, 8, '2009-01-27', '2009-01-30',
'2009-02-05', 1000, 4500);

INSERT INTO billing VALUES(1, 1, '2009-02-02', 5000);
INSERT INTO billing VALUES(2, 2, '2009-02-17', 1000);
INSERT INTO billing VALUES(3, 3, '2009-02-10', 2500);
INSERT INTO billing VALUES(4, 4, '2009-02-05', 3500);
```

Take note of the SQL keyword `REFERENCES` in table description (declaration). `REFERENCES` is used to show relationship between table. Using this, we follow the syntax: `REFERENCES [table_name (column_name)]`. For example the code, `REFERENCES customer(ID)` in creating *accommodation*, means that *accommodation* has a relationship with *customer* through the column *ID*.

INSERT INTO STATEMENT

The `INSERT INTO` statement is used to save information in the database. You have tried this statement in module 2. This time you will discover two ways of using INSERT statement. You can user either of the following syntax:

```
INSERT INTO [table_name] (column_name1,…) VALUES (value, …);
```

or

```
INSERT INTO [table_name] VALUES (values, …);
```

The first syntax specifies the column to be affected. In the second syntax, it is assumed that all columns in the table should be filled. Either of the two, `VALUES` of the insert statement must be in order. Take note of the given code above. It follows the second `INSERT INTO` statement.

Try to use the two syntax to save two additional customers in customer. Take a screenshot of your code and output and share it on our Facebook Group Chat.

UPDATE STATEMENT

To edit a column value, we use the SQL command `UPDATE`. You can use the `UPDATE` command with or without using a `WHERE` clause. Take a look at this syntax:

```
UPDATE [table_name] SET [column_name] = [new_value];
```

Following this syntax will update the whole column with the new value. For an instance, we have table student

```
+----+------------+-----+--------+-----------+-------------+
| ID | Name       | Age | Gender | Status    | Contact     |
+----+------------+-----+--------+-----------+-------------+
|  1 | John D.C   |  19 | M      | Enrolled  | 09237567835 |
|  2 | Paul Cruz  |  21 | M      | Enrolled  | 09208662129 |
|  3 | Anne Lu    |  20 | F      | Enrolled  | 09186007201 |
+----+------------+-----+--------+-----------+-------------+
```

and we want to set all values in *status* to dropped we use

```
UPDATE students SET status = 'dropped';
```

On the other hand, we can update a specific item. To do this, we will use WHERE clause. In this case, it is done as follows.

```
UPDATE students SET status = 'dropped' WHERE id = 3;
```

This command will only update Anne Lu's *status* given the *condition id = 3.*

DELETE STATEMENT

To delete an item in a table we use:

```
DELETE FROM [table_name] WHERE [condition];
```

To delete ALL data from an existing table we use:

```
DELETE FROM [table_name];
```

or

```
TRUNCATE TABLE  [table_name];
```

SELECT – FROM STATEMENT

The `SELECT - FROM` statement retrieves saved data or information from the database. The syntax for this statement is: `SELECT [column_name] FROM [table_name];` see example below:

```
SELECT * FROM customer;
```

Note the use of asterisk in the given SQL command. This means ALL, that is all values of the columns in table customer will be retrieved. The result of this command is as follows:

```
+----+------------+-----+--------+-----------+-------------+
| ID | Name       | Age | Gender | Status    | Contact     |
+----+------------+-----+--------+-----------+-------------+
|  1 | John Doe   |  24 | M      | Single    | 09234567865 |
|  2 | Tom Bin    |  35 | M      | Married   | 09204661129 |
|  3 | Ann Milby  |  20 | F      | Single    | 09188007211 |
|  4 | Jane Mosby |  45 | F      | Married   | 09162003000 |
|  5 | Peter Paul |  30 | M      | Separated | 09221176500 |
+----+------------+-----+--------+-----------+-------------+
```

Let's try to retrieve Names and Genders of all the customers. We can do this by specifying the column name(s) you want to retrieve. **Note that if you will retrieve multiple columns, column names must be separated by a comma.** Let's execute this code and take note of the result.

```
SELECT name, gender FROM customer;
```

Share the result of this command in our Facebook Group Chat with a brief explanation of the result.

WHERE Clause

Now let us talk about the WHERE clause. This is used together with the SELECT statement. This clause specifies a condition. This serves as conditional statement in a query. The syntax is: `SELECT [column_name] FROM [table_name] WHERE [condition];` Take a look at the example below.

```
SELECT * FROM customer WHERE age > 30;
```

In this case, the clause `WHERE age > 30` specifies which information is to be fetched from the database. Note that *age* is a column in the table *accommodation*. This command retrieves customer's information which age is greater than 30. This command will give us the result shown below.

```
+----+------------+-----+--------+---------+-------------+
| ID | Name       | Age | Gender | Status  | Contact     |
+----+------------+-----+--------+---------+-------------+
|  2 | Tom Bin    |  35 | M      | Married | 09204661129 |
|  4 | Jane Mosby |  45 | F      | Married | 09162003000 |
+----+------------+-----+--------+---------+-------------+
```

Always remember that `WHERE` is used to specify condition. We use the following symbols together with the `WHERE` clause to form our condition.

SYMBOLS	MEANING
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>	Not equal
LIKE	*See note below*

The `LIKE` pattern matching operator can also be used in the conditional selection of the where clause. Like is a very powerful operator that allows you to select only rows that are "like" what you specify. The percent sign "`%`" can be used as a *wild card* to match any possible character that might appear before or after the characters specified. For example:

```
SELECT * FROM customer WHERE name LIKE ‘J%’;
```

This SQL statement will match any names that start with 'J'. Strings must be in single quotes.

Or you can specify,

```
SELECT * FROM customer WHERE name LIKE '%y';
```

This statement will match any names that end in a 'y'.

ORDER BY Clause

The `ORDER` clause is used to arrange data in ascending (ASC) or in descending (DESC). This is usually used together with the `SELECT` statement. Try to execute the SQL command below. Have a screenshot of the output, share it on our GC with a short explanation.

```
SELECT * FROM customer WHERE ORDER BY age DESC;
```

COUNT and GROUP BY Clause

Use the accommodation database and execute the code below. In our GC, share a screenshot of the result and briefly explain the use of `COUNT` and `GROUP BY` in:

```
SELECT gender, COUNT(gender) FROM customer GROUP BY gender;
```

SUM Clause

`SUM` allows you to retrieve a sum of a particular column. See the sample code below:

```
SELECT SUM(Deposit) as 'Total Deposit' FROM accomodation;
```

Take note of the command segment, `as 'Total Deposit'`, this sets the result column name to Total Deposit. This will give us an output:

```
+---------------+
| Total Deposit |
+---------------+
|          4500 |
+---------------+
```

IN, NOT IN, and BETWEEN Clause

`IN`

The `IN` conditional operator is really a set membership test operator. That is, it is used to test whether or not a value (stated before the keyword `IN`) is "in" the list of values provided after the keyword `IN`.

Take a loot at this example.

```
SELECT name FROM customer
     WHERE name IN ('John Doe','Ann Lu','Ann Milby');
```

This statement will select the *name* from the *customer* table where the *name* is equal to either: John Doe, Ann Lu, or Ann Milby. It will return the rows if it is ANY of these values. This code will give as a result as shown below:

```
+------------+
| name       |
+------------+
| John Doe   |
| Ann Milby  |
+------------+
```

NOT IN

You can also use `NOT IN` to exclude the rows in your list. Study the result of the command below:

```
SELECT id, name FROM customer
     WHERE id NOT IN (SELECT custID FROM accomodation);
```

This statement will give as a result:

```
+----+------------+
| id | name       |
+----+------------+
|  3 | Ann Milby  |
+----+------------+
```

Take note that I used another SELECT statement inside the NOT IN clause. The inner SELECT statement returns a set of values where the outside SELECT value will compare. You can apply this method in an IN clause.

BETWEEN

The `BETWEEN` conditional operator is used to test to see whether or not a value (stated before the keyword `BETWEEN`) is "between" the two values stated after the keyword `BETWEEN`.

```
SELECT id, name FROM customer WHERE age BETWEEN 20 and 30;
```

This statement will select the *id*, and *name* from the *customer* table where the age is between 20 and 30. This statement's result will be:

```
+----+------------+
| id | name       |
+----+------------+
|  1 | John Doe   |
|  3 | Ann Milby  |
|  5 | Peter Paul |
+----+------------+
```

SQL JOINT

All of the queries up until this point have been useful with the exception of one major limitation - that is, you've been selecting from only one table at a time with your `SELECT` statement. It is time to introduce you to one of the most beneficial features of SQL & relational database systems - the `Join`.

The SQL `Joins` clause is used to combine records from two or more tables in a database. A `JOIN` is a means for combining fields from two different tables by using values common to each. `JOINS` allow you to link data from two or more tables together into a single query result--from one single `SELECT` statement.

There are different types of joins available in SQL:

1. INNER JOIN – returns rows when there is a match in both tables.
2. LEFT JOIN – returns all rows from the left table, even if there are no matches in the right table.
3. RIGHT JOIN – returns all rows from the right table, even if there are no matches in the left table.
4. FULL JOIN – returns rows when there is a match in one of the tables.
5. SELF JOIN – is used to join a table to itself as if the table were two tables, temporarily renaming at least one table in the SQL statement.

Syntax for a join is show below:

```
SELECT [table1].[columnname], [table2.column] FROM
     [table1] [JOIN or Any type of join] [table 2]
     ON [table1].[column] = [table2].[column];
```

Let's have an example. If we want to get customer's name, and the amount of his or her deposit, and his or her rooms number, we will use the SQL Join to link three different tables:

customer which contains the name, *accommodation* which contains roomID and deposit, and *rooms* which contains the room number. Study the command below.

```
SELECT customer.name, accomodation.deposit, room.roomNo
     FROM customer JOIN accomodation
     ON customer.id = accomodation.custID
     JOIN room ON room.roomID = accomodation.roomID;
```

The statement tells us the cutomer table has a relation between accommodation through id and custID. On the other hand, accommodation has a relation with room through roomID. This relation allows us to join tables. Take note of the result on the following page.

See how we retrieved three columns from different tables in one result.

```
+------------+---------+--------+
| name       | deposit | roomNo |
+------------+---------+--------+
| John Doe   |    2000 |    203 |
| Tom Bin    |    1000 |    103 |
| Jane Mosby |     500 |    201 |
| Peter Paul |    1000 |    302 |
+------------+---------+--------+
```

Let us now talk about **ALIAS** in SQL. Observe how the previous code is modified. We set ***c*** as an alias of *customer* table, ***a*** as an alias of *accommodation*, and ***r*** for *room*.

```
SELECT c.name, a.deposit, r.roomNo
     FROM customer c  JOIN accomodation a
     ON c.id = a.custID JOIN room r ON r.roomID = a.roomID;
```

This statement will give us the same result.

SQL UNION

The SQL `UNION` clause/operator is used to combine the results of two or more `SELECT` statements without returning any duplicate rows.

To use this `UNION` clause, each `SELECT` statement must have

1. The same number of columns selected
2. The same number of column expressions
3. The same data type and
4. Have them in the same order

But they need not have to be in the same length. The basic syntax of a `UNION` clause is as follows

```
SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]

UNION

SELECT column1 [, column2 ]
FROM table1 [, table2 ]
[WHERE condition]
```

Let's have an example. Consider the following two tables.

CUSTOMERS Table is as follows.

```
+----+----------+-----+-----------+----------+
| ID | NAME     | AGE | ADDRESS   | SALARY   |
+----+----------+-----+-----------+----------+
|  1 | Ramesh   |  32 | Ahmedabad |  2000.00 |
|  2 | Khilan   |  25 | Delhi     |  1500.00 |
|  3 | kaushik  |  23 | Kota      |  2000.00 |
|  4 | Chaitali |  25 | Mumbai    |  6500.00 |
|  5 | Hardik   |  27 | Bhopal    |  8500.00 |
|  6 | Komal    |  22 | MP        |  4500.00 |
|  7 | Muffy    |  24 | Indore    | 10000.00 |
+----+----------+-----+-----------+----------+
```

ORDERS Table is as follows.

```
+-----+---------------------+-------------+--------+
|OID  | DATE                | CUSTOMER_ID | AMOUNT |
+-----+---------------------+-------------+--------+
| 102 | 2009-10-08 00:00:00 |           3 |   3000 |
| 100 | 2009-10-08 00:00:00 |           3 |   1500 |
| 101 | 2009-11-20 00:00:00 |           2 |   1560 |
| 103 | 2008-05-20 00:00:00 |           4 |   2060 |
+-----+---------------------+-------------+--------+
```

Now, let us join these two tables in our SELECT statement as follows

```
SQL> SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   LEFT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID
UNION
   SELECT  ID, NAME, AMOUNT, DATE
   FROM CUSTOMERS
   RIGHT JOIN ORDERS
   ON CUSTOMERS.ID = ORDERS.CUSTOMER_ID;
```

This would produce the following result

```
+------+----------+--------+---------------------+
| ID   | NAME     | AMOUNT | DATE                |
+------+----------+--------+---------------------+
|    1 | Ramesh   |   NULL | NULL                |
|    2 | Khilan   |   1560 | 2009-11-20 00:00:00 |
|    3 | kaushik  |   3000 | 2009-10-08 00:00:00 |
|    3 | kaushik  |   1500 | 2009-10-08 00:00:00 |
|    4 | Chaitali |   2060 | 2008-05-20 00:00:00 |
|    5 | Hardik   |   NULL | NULL                |
|    6 | Komal    |   NULL | NULL                |
|    7 | Muffy    |   NULL | NULL                |
+------+----------+--------+---------------------+
```