MS SQL

Lecturer: Dr Millicent Agangiba

About SQL

- **SQL**- A command line tool used to manipulate tables and other database objects in a database.

What can SQL do?

- SQL can
  - Execute queries against a database
  - Retrieve data from a database
  - Insert records in a database
  - Update records in a database
  - Delete records from a database
  - Create stored procedures
  - Set permissions on tables, procedures and views

Data Definition Language

- **Data Definition Language** (DDL) statements are used to define the database structure or schema. Some examples:
- CREATE - to create objects in the database
- ALTER - alters the structure of the database
- DROP - delete objects from the database

Data Manipulation Language

- **Data Manipulation Language** (DML) statements are used for managing data within schema objects. Some examples:
- SELECT - retrieve data from the a database
- INSERT - insert data into a table
- UPDATE - updates existing data within a table
- DELETE - deletes all records from a table, the space for

Data Control Language

- **Data Control Language** (DCL) statements. Some examples:
- GRANT - gives user's access privileges to database
- DENY to disallow specified users from performing specified tasks.
- REVOKE - withdraw access privileges given with the GRANT command

Transaction Control Language

- **Transaction Control language** (TCL) statements are used to manage the changes made by DML statements. It allows statements to be grouped together into logical transactions.
- COMMIT - save work done
- SAVEPOINT - identify a point in a transaction to which you can later roll back
- ROLLBACK - restore database to original since the last COMMIT

- DATA DEFINITION LANGUAGE
- SOME COMMANDS AND EXAMPLES

Creating a Table

- A table is made up of one or more columns (also called attributes in relational theory).
- Each column is given a name and a data type that reflects the kind of data it will store. MSSQL supports several data types:
- **NVARCHAR**(**50**) is a column that can store up to **50** characters (using up to 100 bytes), but it can store any number of characters less than **50** as well without adding trailing spaces.
- **VARCHAR (50)** - it can be any number of bytes up to the maximum. The additional bytes are the count of the number of bytes currently used, generally. So **varchar**(**50**) could hold 0 to **50** characters, and would take 52 bytes to store

Creating a Table

- **NUMBERIC (18,0)-** The first value is the precision and the second is the scale, so **18,0** is essentially **18** digits with **0** digits after the **decimal** place. If you had **18,**2 for example, you would have **18** digits, two of which would come after the **decimal**
- **INT** is a data type in the database - an **integer** (whole number). What it **means** depends on the database you use - in **SQL** Server the 4 specifies the field precision. However, this will always be the size of an **int in SQL** Server

Creating a Table

- **MONEY** - The **money data type** is an abstract **data type. Money** values are stored significant to two **decimal** places. These values are rounded to their amounts in dollars and cents or other **currency** units on input and output, and arithmetic operations on the **money data type** retain two-**decimal**-place precision
- **CHAR(10)** is a data type in the database - CHAR allows you to store a string of 10 characters.

Creating a Table

- **DATE** - Date and Time data type. Can contain a date and time portion in the format: DD-MON-YY HH:MI:SS. No additional information is needed when specifying the DATE data type. If no time component is supplied when the date is inserted, the time of 00:00:00 is used as a default. The output format of the date and time can be modified to conform to local standards.

Example Creating a Table

- To create a new table to hold employee data, we use the CREATE TABLE statement:

```
CREATE TABLE Employee
(Empid INT IDENTITY(1,1) ,
first_name NVARCHAR(50) ,
last_name NVARCHAR(50),
bdate DATE,
City NVARCHAR(20),
salary MONEY);
```

Adding a new column (field)

- In order to add a new field use the command ALTER
- For example to add a new column called contact

```
ALTER TABLE Employee
ADD  contact NVARCHAR(15);
```

Add primary key

Identify the field to be used as primary key

Use ALTER command

Example make the field ,Empid the primary key

```
ALTER TABLE Employee
Add constraint pk_employee
Primary key (Empid) ;
```

Add a foreign key

- Foreign key helps link one table to the other ie. Establish relationship between the two tables. Create second table called Department

```
Create table department
(deptno INT NOT NULL,
dept_name NVARCHAR(20),
location NVARCHAR(20),
Contact NVARCHAR(20),
PRIMARY KEY(deptno));
```

Add a foreign key

```
Alter table Employee
Add constraint
   fk_emp_department
FOREIGN KEY (deptno)
references
   department (deptno);
```

Create table wit auto increment for id with primary key:

```
 CREATE TABLE Persons (
Personid int IDENTITY(1,1) PRIMARY KEY,
LastName varchar(255) NOT NULL,
FirstName varchar(255),
Age int );
```

- DATA MANIPULATION LANGUAGE
- SOME COMMANDS AND EXAMPLES

Using SELECT

Retrieving all records

```
SELECT *
FROM Employee
```

Example:

```
SELECT *
From Employee;
```

SELECT with Conditions

Retrieving specific fields from a Table

```
SELECT column1, column2, column3
From Table_name
WHERE condition;
```

Example:

```
SELECT firstname, lastname, salary
FROM Employee
WHERE  lastname= ' Agangiba ';
```

Using logical operators

SELECT *

FROM Table_name

WHERE Condition1 AND Condition2; or

Example:

SELECT *

FROM Employee

WHERE City= 'Accra' AND salary between 2000 and 3000;

Limit number of records

```
SET ROWCOUNT number_of_records
SELECT * FROM Table_name
WHERE  Condition;
```

Example:

```
SET ROWCOUNT 3
SELECT * FROM Employee
WHERE  EmployeeID < 105;
```

Limit number of records

```
SELECT TOP number_of_records  *
FROM Table_name ;
```

Example: To Display the 4 top records

```
SELECT TOP 4 * FROM Employee;
```

Sorting records using SELECT

```
SELECT column1, column2, column3
FROM Table_name
ORDER BY column_name;
```

Example:

```
SELECT firstname, lastname, salary
From Employee
ORDER BY lastname;
```

Sorting records using SELECT

```
SELECT *
FROM Table_name
WHERE  column IN (values);
```

Example:

```
SELECT *
From Employee
WHERE lastname IN ('Agangiba', 'Asare');-----multiple  values
```

```
SELECT *
From Employee
WHERE lastname LIKE 'Agangiba%’----single  value
```

Sorting records by ascending or descending order

```
SELECT column1, column2, column3
FROM Table_name
ORDER BY DESC or ASC;
```

Example:

```
SELECT firstname, lastname, salary
From Employee
ORDER BY salary DESC; or ORDER BY salary ASC
```

Using functions

To count the number of records

COUNT FUNCTION IGNORES NULL VALUES

```
SELECT COUNT (*)
FROM Table_name
```

Example:

```
SELECT COUNT (*)
FROM Employee
```

```
SELECT COUNT (*) AS TOTAL
FROM Employee
```

Using functions

MAX RETURNS THE HIGHEST VALUE

```
SELECT MAX (column_name)
FROM Table_name
```

Example:

```
SELECT MAX (salary)
FROM Employee
```

Using functions

MIN RETURNS THE MINIMUM VALUE

```
SELECT MIN (column_name)
FROM Table_name
```

Example:

```
SELECT MIN (salary)
FROM Employee
```

Using functions

AVG RETURNS THE AVERAGE VALUE

```
SELECT AVG (column_name)
FROM Table_name
```

Example:

```
SELECT AVG (salary)
FROM Employee
```

Using functions

SUM RETURNS THE TOTAL VALUE

```
SELECT SUM (column_name)
FROM Table_name
```

Example:

```
SELECT SUM (salary)
FROM Employee
```

SELECT distinct values

```
SELECT DISTINCT column1
FROM Table_name
```

Example:

```
SELECT DISTINCT lastname
From Employee
```

SELECT INTO

- The following SQL statement creates a backup copy of Customers
- SELECT * INTO CustomersBackup
  FROM Customers;

INSERT INTO SELECT

- The following SQL statement copies "Suppliers" into "Customers" (the columns that are not filled with data, will contain NULL)
- INSERT INTO Customers (CustomerName, City, Country)
  SELECT SupplierName, City,
  Country FROM Suppliers;

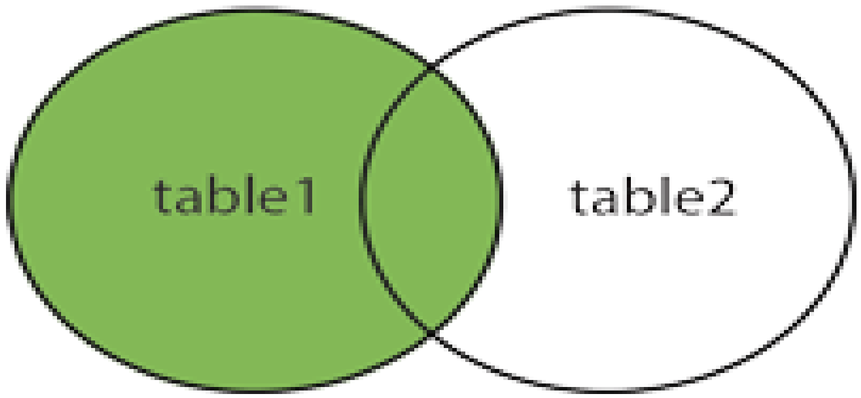
RETRIEVING From 2 or more tables

2 tables:
```
SELECT EmpName, Surname, Salary,
DeptName
FROM Employee INNER JOIN Department
ON Department. DeptID= Employee.DeptID;
```

```
SELECT EmpName, Surname, Salary, DeptName
FROM Employee, Department
WHERE  Department. Dept_ID= Employee.Dept_ID;
```

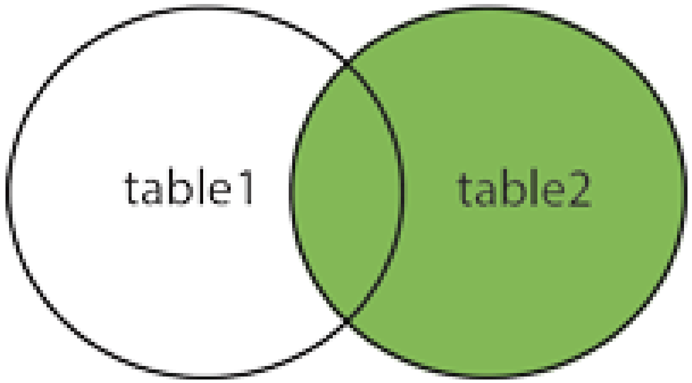
LEFT JOIN (LEFT OUTER JOIN)

- SELECT Customers.CustomerName, Orders.OrderID
- FROM Customers
- LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;



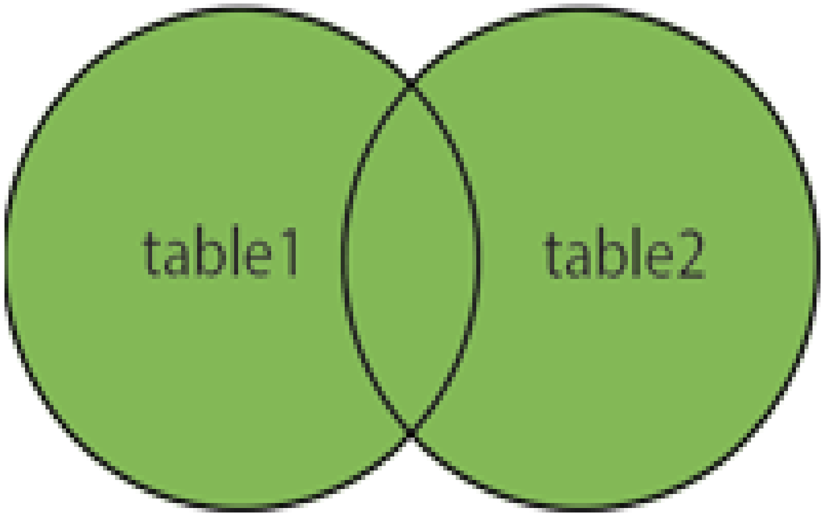
RIGHT JOIN (RIGHT OUTER JOIN)

- SELECT Orders.OrderID, Employees.LastName, Employees.FirstName
  FROM Orders
  RIGHT JOIN Employees ON Orders.EmployeeID = Employees.EmployeeID;



FULL OUTER JOIN

- SELECT Customers.CustomerName, Orders.OrderID
  FROM Customers
  FULL OUTER JOIN Orders ON Customers.CustomerID
  =Orders.CustomerID;



Enter Values Into Table

- To insert new data into the employee table, we use the INSERT statement: insert the same order as fields appear.

INSERT INTO table-name (column-names)

VALUES (values) – enter specific fields

- Or

INSERT INTO table-name VALUES- enter all fields

Example to insert record into Employee

INSERT INTO Employee VALUES ('Tony', 'Abban', '24-JAN-54','M', 1000);

Enter Multiple Rows Into Table

```
INSERT INTO table_name(column1,column2…)
VALUES (value1,value2,…), (value1,value2,…)
```

Example:

```
INSERT INTO Employee(Firstname, Lastname, Title)
 VALUES('Milcah', 'Agangiba', 'Engineer'),
            ('Dora', 'Asare', 'Manager’);
```

Enter Multiple Rows Into Table

// To view what values were modified

```
INSERT INTO Employee (last_name,first_name,salary)
OUTPUT inserted.first_name,inserted.last_name,
inserted.salary
VALUES ('Mart','Martha',3100), ('Faa', 'Grace',1200)
```

Updating records

- To update records use the command UPDATE

```
UPDATE Table_name
Set column_ name
WHERE conditions
```

For example increase the SALARY of all employees by 20%

```
UPDATE Employee
Set SALARY=SALARY*1.2;
```

Do an increment of salary of a particular department

```
UPDATE Employee
Set SALARY=SALARY*1.2
WHERE department= ‘Marketing’;
```

Deleting records

- In order to delete a particular or group of records use the DELETE
- DELETE from employee
- Where (state condition)

```
DELETE From Employee
Where salary=2000
```

```
DELETE From Employee
OUTPUT deleted.first_name,deleted.last_name, deleted.salary

Where salary=2000
```

EXERCISE

EXERCISE

- Customer Service Office Takes Details For The Following On Customers:
- Customer Number, customer Name, City, Contact Number
- The Following Information On each Order placed:
- Order Number, Order Date, Delivery Mode, Delivery Date
- NOTICE: A CUSTOMER MAKES MORE THAN ONE ORDER

EXERCISE CONT'D

- In Sql Interface Create The Tables For Customers And Order
- Enter Four Values For Both Tables
- Add Primary Key For Both Table
- Add The Foreign Key
- Make A Query For The Following Details:

1. Retrieve Customer Name, City, Contact Number
2. Retrieve Customer Name, Contact Number with a specified City
3. Retrieve Customer Name, Contact, Order date, Delivery Date

SQL FUNCTIONS

- Go to database
- Go to programmability
- Go to system functions
- All functions within SQL are listed

SQL FUNCTIONS

- Examples:
- SELECT @@SERVERNAME – the current admin name
- SELECT @@VERSION – version of SQL server in use
- SELECT @@CONNECTIONS –number of connections to the server
- SELECT @@SERVERNAME, @@VERSION, @@CONNECTIONS

STRING FUNCTIONS

- Examples: to change the column Lastname in Employee table to upper or lower case
- SELECT UPPER (Lastname) or SELECT LOWER(Lastname)
- FROM Employee

- Example: find the length of a column
- SELECT LEN (Lastname)
- FROM Employee

DATE FUNCTIONS

- Examples: displays current date and time
- SELECT GETDATE()
- SELECT DAY (GETDATE())-displays day
- SELECT MONTH (GETDATE())-displays month
- SELECT YEAR (GETDATE())-displays year
- SELECT DATENAME (WEEKDAY, GETDATE())- displays the day of the week

DATE FUNCTIONS

- Examples: displays current date and time
- SELECT DATEADD (DAY, 15,GETDATE())
- Difference between 2 dates
- SELECT DATEDIFF(DAY, '2019/10/25', '2019/09/15') AS REMAINING;
- SELECT DATEDIFF(YEAR, '2019/10/25', '2019/09/15') AS REMAINING;
- SELECT DATEDIFF(MONTH, '2019/10/25', '2019/09/15') AS REMAINING;

PRACTICE

- Design a database for a car rental company to keep track of its Cars, Customers and Employee. One employee handles several transactions for a number of customers. Also a customer can rent more than one car at a time
- Information for Customer details: Customer first name, surname, contact, date of hiring, returned date, amount due
- Employee information: Employee first name, surname, date of birth, date of employment, contact
- Car details: Car number, car type, manufacturer, colour, status (hired or garaged) charge per day
- Design a query to show the names, age and the duration each worker has been with company
- Design a query to show how many a customer rented a specified car, amount paid for hiring the numbered days

READINGS

- READ ON DATABASE ACID RULES
- CONCURRENCY CONTROL IN DATABASES