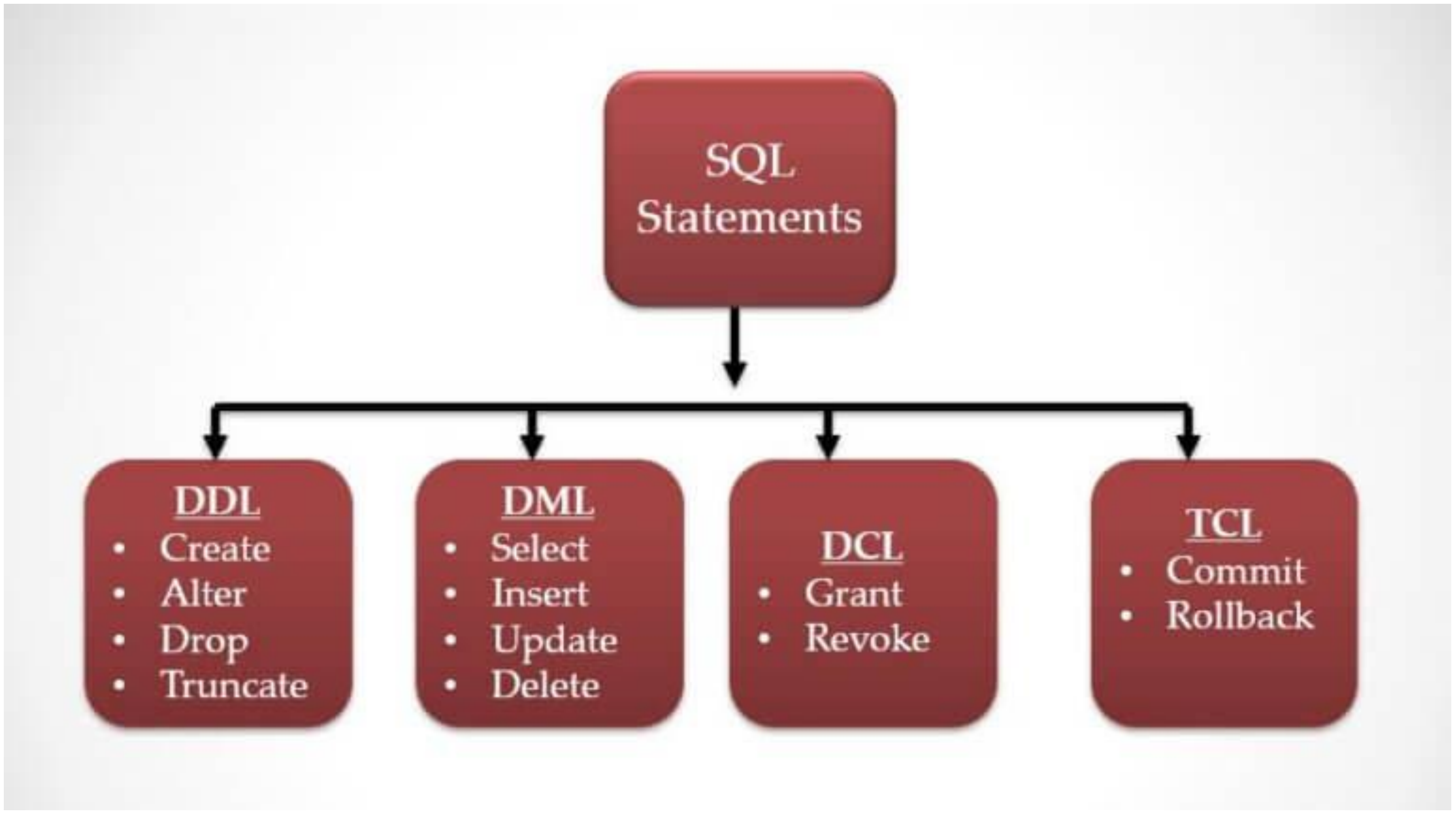
SQL – Part 2

NGU.CCAS.4.3

Learning Outcomes

- Understand the meaning of Data Manipulation Language(DML)
- Implement different SQL statements that support the DML

SQL Statements

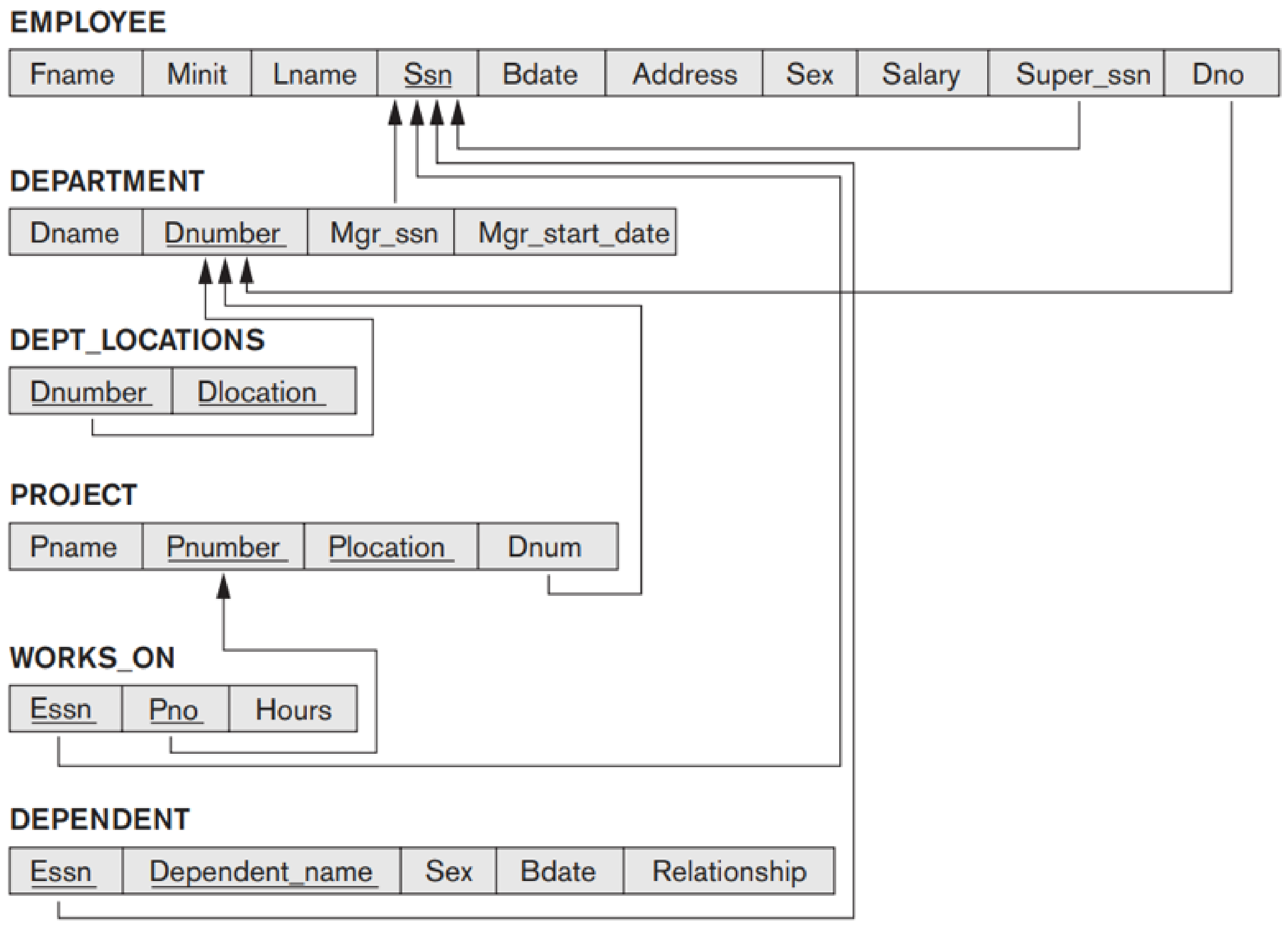


Data Manipulation Language (DML)

Data Manipulation Language(DML)

- Data Manipulation Language or DML is a subset of operations used to insert(add), delete, and update data in a database.

COMPANY Relational Database Schema



DML: INSERT

- ***INSERT statement*** is used to insert values into tables.
- Attribute values should be listed in the same order as the attributes were specified in the CREATE TABLE command.
  - `INSERT INTO` table_name `VALUES` (Value_lists);

    -
      ```
      INSERT INTO Department VALUES
      (‘R&D',467,‘1992/08/07');
      ```

- An alternate form of INSERT specifies explicitly the attribute names that correspond to the values in the new tuple
  - Attributes with NULL values can be left out.
    - `INSERT INTO` table_name (column_lists)
      `VALUES` (Value_lists);

      -
        ```
        INSERT INTO Department (Dname,Dnumber)
        VALUES ('HR', 46);
        ```

DML: INSERT (cont’d)

- ***INSERT*** allows insertion of multiple tuples resulting from a query into a relation.

```
INSERT INTO Department VALUES ('Finance', 249,'1992/11/13'),
                              (‘HR’, 46,'1995/07/08’);
```

Show the values after ***INSERT*** statement

```
INSERT INTO Department (Dname, Dnumber) OUTPUT
inserted.Dname, inserted.Dnumber VALUES (HR’, 46);
```

Example

Insert a row into the Dept_Locations table with values 249, "Zamalek"

DML: UPDATE

- ***UPDATE statement*** is used to modify attribute values of one or more selected tuples.

  - UPDATE table_name
    SET ***c1=v1, c2=v2, ……., cn=vn***
    [WHERE ***condition];***

  You can add more than 1 condition

  - UPDATE table_name
    SET ***c1=v1, c2=v2, ……., cn=vn***
    [WHERE ***condition or/and condition];***

    -
      ```
      UPDATE Department
        set Dnumber='259'
        Where Dname='HR';
      ```

DML: DELETE

- ***DELETE*** command removes tuples from a relation.

  -
    ```
    DELETE FROM Table_name
     [Where condition];
    ```

    -
      ```
      DELETE FROM Department
      where Dnumber='123';
      ```

      Delete multiple tuples

    -
      ```
      DELETE FROM Customer
      where customer_id between 6 and 9;
      ```

    -
      ```
      DELETE Top(5) FROM Department;
      ```

      Delete 3% from the records

    -
      ```
      DELETE TOP(3) Percent FROM Department;
      ```

DML: SELECT

- ***SELECT*** statement is used to retrieve data from the database.
- There are many options and flavours to the SELECT statement in SQL.

```
SELECT *FROM table_name;
```

Retrieve all data in the relation

```
SELECT attribute_list
FROM table_name;
```

The columns you need to retrieve

```
Example:
        SELECT Fname,
               Lname
        FROM Employee;


        SELECT Fname + ' ' + Lname AS 'Employee Name'
        FROM Employee;
```

The + sign is called concatenate. So it will combine the first name with last name

Alias

The name of the generated column

DML: SELECT (cont'd)

- The basic form of the ***SELECT*** statement, sometimes called a mapping or a select-from-where block, is formed of the three clauses SELECT, FROM, and WHERE and has the following form:

```
SELECT attribute_list
FROM   table_name
Where  Condition
```

```
Example:
       SELECT Fname, Lname
       FROM Employee
       Where Sex ='Female';
```

DML: SELECT (Operators)

Example: Retrieve the first and last name of female employees whose salary > 6000

```
SELECT Fname, Lname
FROM Employee
Where Sex ='Female’ AND Salary > 6000;
```

Example: Retrieve the first and last name of female employees or employees who work for department number 4

```
SELECT Fname, Lname
FROM Employee
Where Sex ='Female' OR Dno= 4;
```

DML: SELECT (Operators)

- In SQL, the basic logical comparison operators for comparing attribute values with one another and with literal constants are ***=, <=, >, >=,*** and ***<>.***
- SQL allows queries that check if a value is ***NULL*** (missing or undefined or not applicable)
- SQL uses ***IS*** or ***IS NOT*** to compare ***NULLs*** because it considers each NULL value distinct from other NULL values, so equality comparison is not appropriate.
  - `Example:` Retrieve all employees who do not have a phone number.

```
SELECT * FROM Employee
where Phone is  NULL;
```

DML: SELECT (IN operator)

- Example: Retrieve all employees who live in TX, CA, or NY.

IN operator instead of using multiple ORs

```
SELECT * FROM Employee
Where Address in ('TX','CA','NY’);
```

- Example: Retrieve all employees that are not located in TX, CA, or NY.

```
SELECT * FROM Employee
Where Address not in ('TX','CA','NY');
```

DML: SELECT (Between operator)

- Example: Retrieve the first and last name of all employees whose salary is between 2000 and 4000.

```
SELECT Fname, Lname
FROM Employee
Where Salary >=2000 and Salary <= 4000;
```

- **OR**

```
SELECT Fname, Lname
FROM Employee
Where Salary between 2000 and 4000;
```

- We can also use ***NOT BETWEEN.***

DML: SELECT (DISTINCT values)

- SQL does not treat a relation as a set; duplicate tuples can appear
- To eliminate duplicate tuples in a query result, the keyword ***DISTINCT*** is used.

```
SELECT Dno FROM Employee;
```

May have duplicate Dno values

- Instead, we can use:

```
SELECT Distinct Dno FROM Employee;
```

Doesn't have any duplicate values

```
SELECT Distinct Dno,Salary FROM Employee;
```

The both values together will not be duplicated

DML: SELECT (LIKE operator)

- ***LIKE*** is a logical operator that determines if a character string matches a specific pattern.
  - The ***LIKE*** comparison operator is used to compare partial strings
  - Two reserved characters are used: '%' (or '*' in some implementations) replaces an arbitrary number of characters, and '_' replaces a single arbitrary character.
- Example: Find the names of employees whose last name starts with letter ‘h’

```
SELECT Fname,Lname
FROM Employee
Where Lname LIKE 'h%';
```

- Example: Find the names of employees whose last name ends with letter ‘m’

```
SELECT Fname,Lname
FROM Employee
Where Lname LIKE '%m';
```

DML: SELECT (LIKE operator)

- Example: Find the names of employees whose last name contains ‘ha’.

```
SELECT Fname,Lname
FROM Employee
Where Lname LIKE '%ha%';
```

- Example: Find the employees whose first name consists of 4 letters only.

```
SELECT * FROM Employee
Where Fname LIKE '____';
```

- Example: Find the employees whose first name consists of 4 letters and starts with letter ‘Z’.

```
SELECT * FROM Employee
Where Fname LIKE 'Z___';
```

DML: SELECT (LIKE operator)

- Example: Find the employees who have “@gmail.com” email.

```
SELECT * FROM Employee
Where Email LIKE ‘% @gmail.com’;
```

- Example: Find the employees whose first name starts with M or Z.

```
SELECT * FROM Employee
Where Fname LIKE '[MZ]%’;
```

- Example: Find the employees whose first name starts from A to M.

```
SELECT * FROM Employee
Where Fname LIKE '[A-M]%’;
```

- We can also use ***not like***

```
SELECT * FROM Employee
Where Fname not LIKE '[A-M]%';
```

DML: SELECT (cont'd)

- Example: Find the Products in which the price is around 300.

```
SELECT * FROM Product
WHERE Price LIKE '3__.%';
```

- Example: Retrieve all employees who were born during the 1992s.

```
SELECT * FROM Employee
Where Bdate LIKE '__9%';
```

DML: SELECT (ORDER BY)

- ***ORDER BY*** clause is used to sort the tuples in a query result based on the values of some attributes.

- The default order is in ascending order of values. We can specify the keyword ***DESC*** if we want to see the result in a descending order of values. The keyword ***ASC*** can be used to specify ascending order explicitly.

```
SELECT attribute_list
FROM   table_name
ORDER BY
```

DML: SELECT (ORDER BY)

- Example: Sort the employees list by the first name in ascending order.

```
SELECT Fname,Lname
FROM Employee
ORDER BY Fname;
```

- Example: Sort the employees list by the first name in descending order.

```
SELECT Fname,Lname
FROM Employee
ORDER BY Fname  DESC;
```

DML: SELECT (ORDER BY)

- Sort a result set by multiple column.
    - Example: Sort the employees list by the Address, then by the first name.

```
SELECT Fname,Lname, Addres
FROM Employee
ORDER BY Address,
         Fname;
```

    - Example: Sort the employees list by the Address in descending order, then by the first name in ascending order.

```
SELECT Fname,Lname, Addres
FROM Employee
ORDER BY Address DESC,
         Fname;
```

DML: SELECT (Inner Join Tables)

- ***Inner Join*** the default type of join in a joined table, where a tuple is included in the result only if a matching tuple exists in the other relation. The foreign key is used to join the tables.

```
SELECT Fname,Lname
FROM Employee , Department
WHERE Dname='HR' and Dnumber=Dno;
OR
SELECT Fname,Lname
FROM Employee E, Department D
WHERE D.Dname='HR' and D.Dnumber=E.Dno;
OR
SELECT Fname,Lname
FROM Employee E join Department D
on D.Dname='HR' and D.Dnumber=E.Dno;
```

DML: SELECT (Outer Join Tables)

- ***Outer join*** is used for matched tuples along with un matched tuples, as well as from one or both of the tables.
- There are three types of outer join
  - ***Left outer join (Left join):***
    - Returns only unmatched tuples from the left table, as well as matched tuples in both tables.
  - ***Right outer join (Right join):***
    - Returns only unmatched tuples from the right table, as well as matched tuples in both tables.
  - ***Full outer join:***
    - Returns unmatched tuples from both tables, as well as matched tuples in both tables.

DML: SELECT (Outer Join Tables)

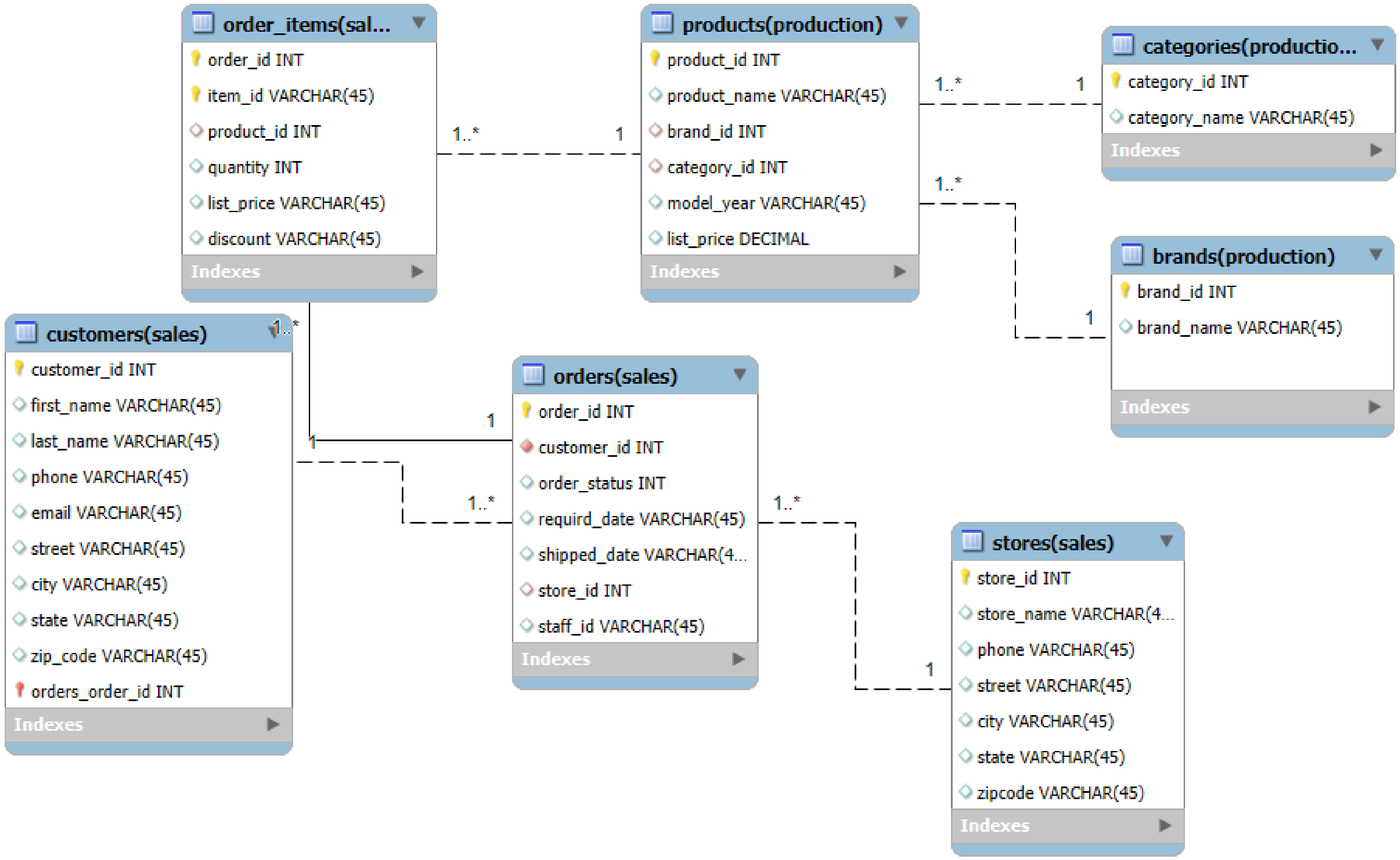
Examples:

```
        SELECT Fname,Lname
        FROM Employee E  left outer join Department D
        on D.Dnumber=E.Dno;


        SELECT Fname,Lname
        FROM Employee E  right outer join Department D
        on D.Dnumber=E.Dno;


        SELECT Fname,Lname
        FROM Employee E  full outer join Department D
        on D.Dnumber=E.Dno;
```

DML: SELECT (Join more than 2 tables)



DML: SELECT (Join more than 2 tables)

NEWGIZA UNIVERSITY

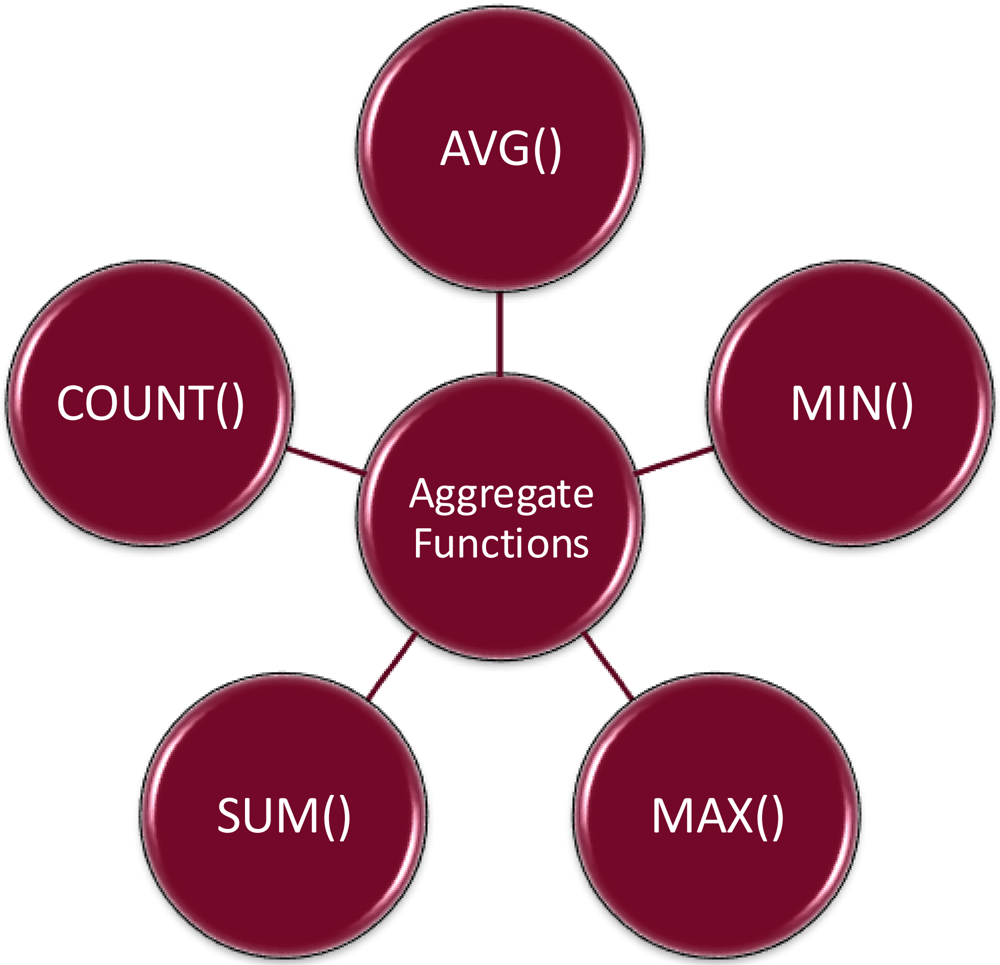
Example: Retrieve the customer first and last name, store details as well as order id.

```
SELECT first_name, last_name, store_name, order_id
FROM Customer C, Orders O, Store S
WHERE C.customer_id=O.customer_id and
O.store_id=S.store_id;
```

OR

```
SELECT first_name, last_name, store_name, order_id
FROM Customer C join Orders O on
C.customer_id=O.customer_id join Store S
on O.store_id=S.store_id;
```

DML: Aggregate Function



DML: Aggregate Function (cont'd)

NEWGIZA UNIVERSITY

Example: Find the maximum salary of employees.

```
SELECT  MAX(Salary) AS max_salary
FROM Employee;
```

Example: Find the maximum, minimum, and average salary of employees.

```
SELECT  MAX(Salary) AS max_salary, MIN (Salary) AS
min_salary, AVG(Salary) as avg_salary
FROM Employee;
```

DML: Aggregate Function (cont'd)

Example: Retrieve the number of employees in the company.

```
      SELECT COUNT (*) AS number_of_employees
      FROM Employee;
OR
      SELECT COUNT (Emp_id) AS number_of_employees
      FROM Employee;
```

Example: Retrieve the number of employees in the HR department.

```
      SELECT COUNT(*) AS number_of_employees
      FROM Employee, Department
      WHERE Dname='HR' and Dnumber=Dno;
```

DML: Grouping

- In many cases, we want to apply the aggregate functions to subgroups of tuples in a relation.
- Each subgroup of tuples consists of the set of tuples that have the same value for the grouping attribute(s).
- The function is applied to each subgroup independently.
- SQL has a **GROUP BY**-clause for specifying the grouping attributes, which must also appear in the SELECT-clause.

DML: Grouping (cont'd)

Example: For each department, retrieve the department number, the number of employees in the department, and their average salary.

```
SELECT Dno, COUNT (*) As number_of_employees, AVG (Salary) As
average_salary
FROM Employee
Group by Dno;
```

Example: For each project, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT Pnumber, Pname, COUNT(*)
FROM Project, Work_on
WHERE Pnumber=Pno
Group by Pnumber, Pname;
```

DML: Having-clause

- Sometimes we want to retrieve the values of these functions for only those groups that satisfy certain conditions

- The ***HAVING***-clause is used for specifying a selection condition on groups (rather than on individual tuples)

Example: For each project on which more than two employees work, retrieve the project number, project name, and the number of employees who work on that project.

```
SELECT Pnumber, Pname, COUNT(*)
FROM Project, Work_on
WHERE Pnumber=Pno
Group by Pnumber, Pname
Having COUNT(*) >2;
```

DML: Nested Queries (sub queries)

NEWGIZA UNIVERSITY

- A complete SELECT query, called a nested query, can be specified within the WHERE-clause of another query, called the outer query.

Example: Retrieve the student_number who achieved the highest mark

```
SELECT Stdno
FROM register
WHERE mark= (SELECT MAX(mark) FROM register);
```

Example: Retrieve the students' info who studied with Khalid

```
SELECT Stdno, Fname, Lname
FROM Student S join register R
on S.Stdno=R.Stdno
WHERE courseid in (SELECT courseid FROM Student S join
register R on S.Stdno =R.Stdno
WHERE Fname='Khalid');
```

DML: All & Any Operator

Any operator returns a boolean value as a result. Returns TRUE if ANY of the subquery values meet the condition.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ANY
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

All operator returns a boolean value as a result. Returns TRUE if ALL of the subquery values meet the condition.

```
SELECT column_name(s)
FROM table_name
WHERE column_name operator ALL
  (SELECT column_name
  FROM table_name
  WHERE condition);
```

Example

Employee (employee_name, street, city)
Works (employee_name, company_name, salary)
Company (company_name, city)
Manages (employee_name, manager_name)

1. Find the names, street address, and cities of residence for all the employees who work for ‘First Bank’ and earn more than $10,000.

2. Find the names of all employees in the database who live in the same cities as the companies for which they work.

3. Find the names of all employees who do not work for ‘ First Bank’.

Example (Cont'd)

4. Find the names of all employees who earn more than every employee of ‘Small Bank’.

5. Find the names of all employees who earn more than the average salary of all employees of their company.

6. Find the names of all companies that have the smallest payroll.

Summary

- Understand what is Data Manipulation Language
- Different Examples of DML
  - Insert
  - Update
  - Delete
  - Select
  - Aggregate Functions
  - Having Clause