Information
Technology
Institute

Structured Query Language (SQL)

Lecturer: Manar Ayman   Mail: mafathi@iti.gov.eg   Room: 3005

Made by: Shahinaz S. Azab   Edited by: Mona Saleh , Rana Salah, Manar Ayman

Structured Query Language (SQL)

Data Definition Language (DDL)

Data Manipulation Language (DML)

Data Control Language (DCL)

Database Transaction

- A transaction is an executing program that forms a logical unit of database actions.
- It includes one or more database access operations such as insert, delete and update.
- The database operations that form a transaction can either be embedded within an application program or they can be specified interactively via a high-level query language such as SQL.

Database Transaction Properties

ACID Properties

1. Atomicity

A transaction is atomic, meaning it's treated as a single, indivisible unit. It's either fully completed or fully rolled back if any part fails.

2. Consistency

Transactions bring the database from one consistent state to another. Data must satisfy predefined integrity constraints.

3. Isolation

Transactions are isolated from each other, ensuring that concurrent transactions do not interfere with each other.

4. Durability

Once a transaction is committed, its changes are permanent and survive system failures.

Database Schema

A schema is a group of related objects in a database.
There is one owner of a schema who has access to manipulate the structure of any object in the schema.
A schema does not represent a person, although the schema is associated with a user that resides in the database.

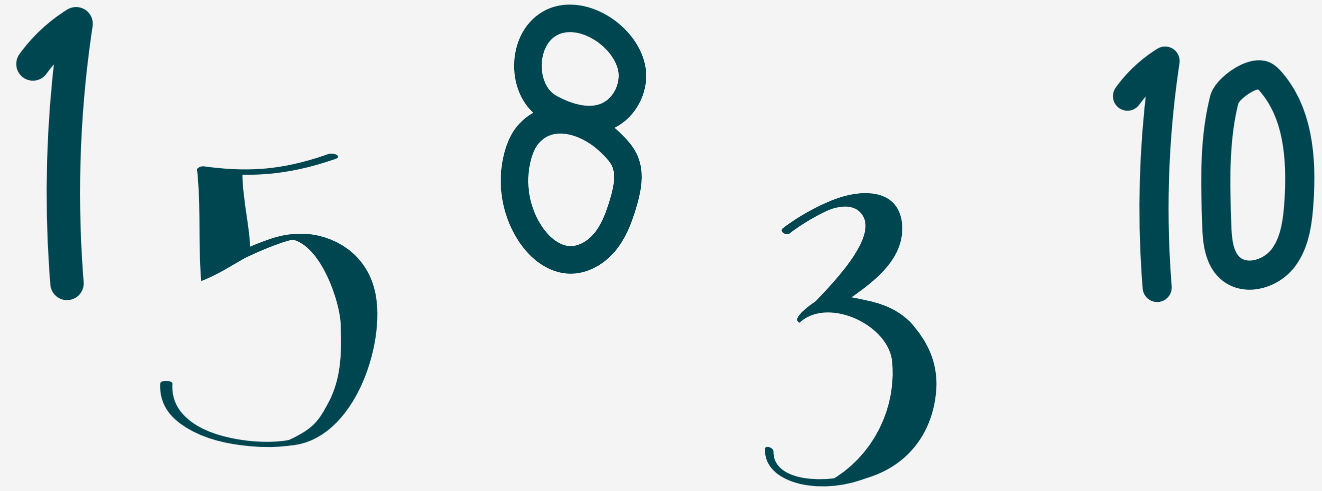
Data Types

A data type determine the type of the data that can be stored in a database column. The most Commonly used data types are:

Alphanumeric

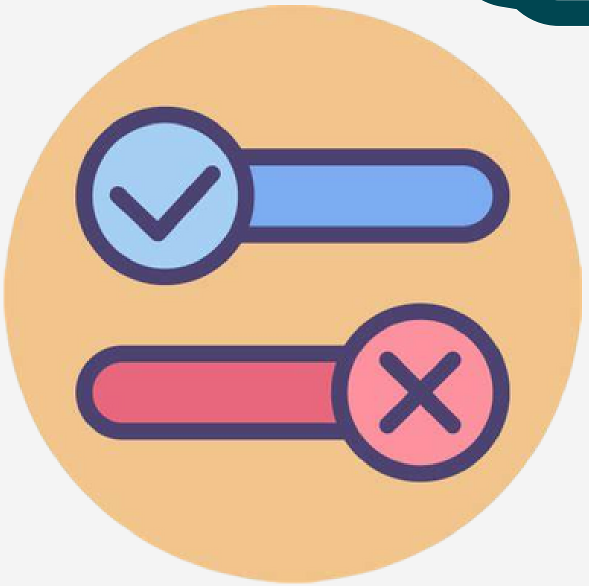
Data types used to store characters, numbers, special characters, or nearly any combination.

Numeric



Date and Time

Boolean



Database Constraints

- Primary Key (Not Null + Unique)
- Not Null
- Unique Key
- Referential Integrity (FK)
- Check

Data Definition Language (DDL)

- **Creating Database Objects:**
  - Like tables, Views, Indexes, and schemas
- **Modifying Database Objects:**
  - Such as Altering table structures or adding constraints.
- **Deleting Database objects:**
  - Delete unwanted database objects using (Drop or Truncate).

Data Definition Language (DDL) (Cont’d)

- **Data Integrity:**
  - DDL helps enforce data integrity by defining constraints such as primary keys, foreign keys, and unique constraints.
- **Data Dictionary:**
  - DDL statements are typically stored in a data dictionary, which is a repository of metadata about the database.
- **Security Implications:**
  - DDL commands are powerful and should be used with caution, as they can impact the entire database structure.

Create Command

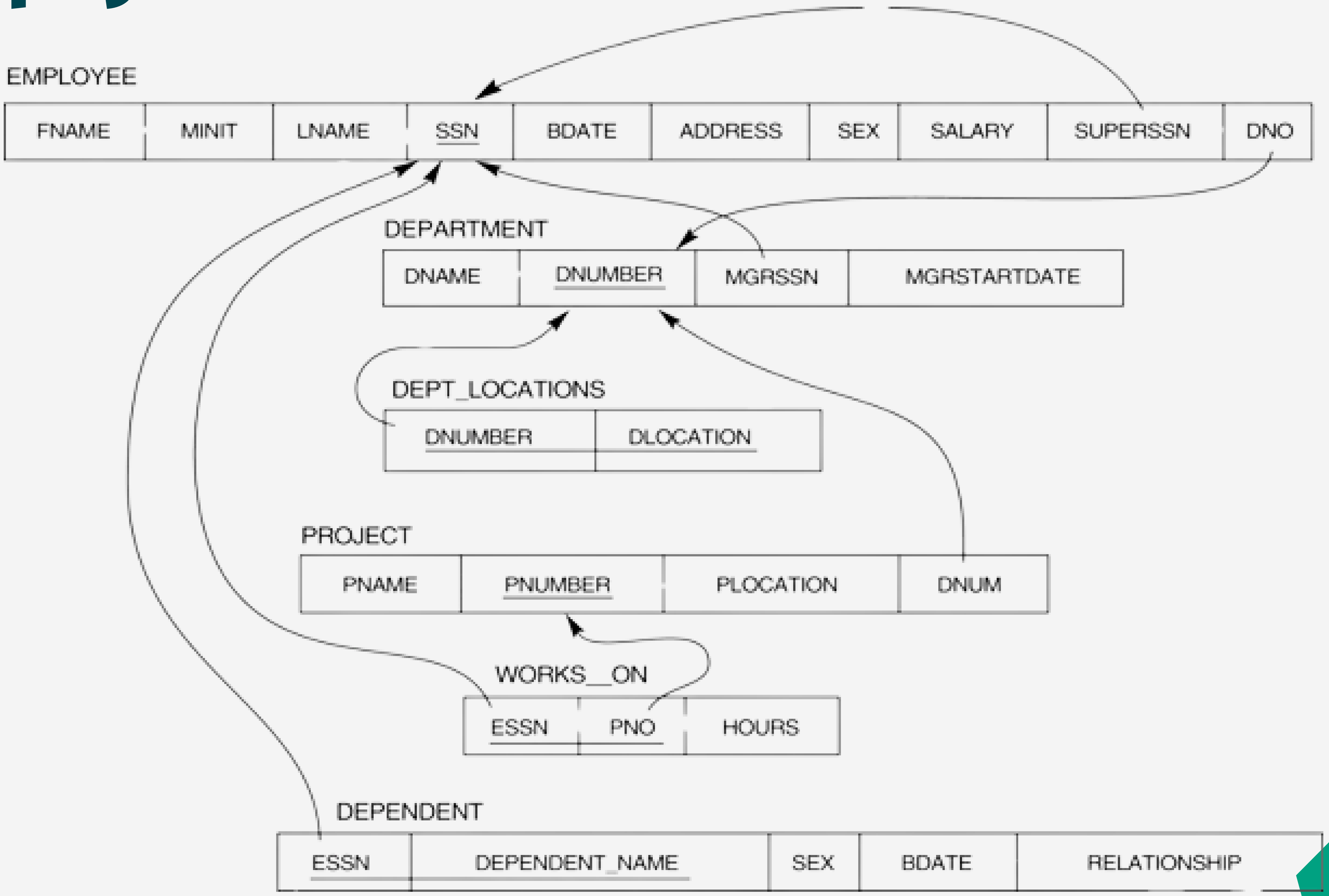
- **Syntax:**

```
CREATE TABLE table_name
(column1 DATA_TYPE [CONS_TYPE CONS_NAME],
 column2 DATA_TYPE [CONS_TYPE CONS_NAME],.....)
```

- **Example:**

```
CREATE TABLE Students
(
  ID NUMBER(15) PRIMARY KEY,
 First_name CHAR(50) NOT NULL,
 Last_name CHAR(50),
 City CHAR(50),
 Birth_Date DATE
);
```

Apply on Create Command



EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT_LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS__ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEPENDENT_NAME	SEX	BDATE	RELATIONSHIP

Alter Command

- **Syntax:**
  ```
  ALTER TABLE table_name
  ADD column_name DATA_TYPE;
  / DROP COLUMN column_name;
  ```
- **Example:**

  -
    ```
    ALTER TABLE Students
    ADD CONSTRAINT Found_LName NOT NULL (Last_name);
    ```

  -
    ```
    ALTER TABLE Students
    ADD country CHAR(50);
    ```

  -
    ```
    ALTER TABLE Students
     DROP COLUMN City;
    ```

Apply on ALter Command

- Add Department location as new column on Department table.
- Add constraint on Department location to be not null.
- Add Age (new column) at Dependent table.
- then delete the Age column.

Truncate Command

- **Syntax:**
  - TRUNCATE TABLE table_name;
- **Example:**
  - TRUNCATE TABLE Students;

Drop Command

- **Syntax:**
  - DROP TABLE table_name;
- **Example:**
  - DROP TABLE Students;

- **Truncate** used to delete the data from the table.
- **Drop** used to delete the table structure with its data.

Data Manipulation Language (DML)

- **Retrieving Data:**
  - Retrieve data from one or more database tables using queries (select Command).
- **Inserting Data:**
  - Insert new records or rows into database tables (Insert Command).
- **Updateing Data:**
  - Updating existing data, making it useful for modifying records (Update Command).
- **Deleting Data:**
  - Delete records or rows from database tables (Delete Command).

Data Manipulation Language (DML)

- **Data Filtering:**
  - DML allows you to filter data using conditions specified in the queries (e.g., WHERE clause).
- **Data Transformation:**
  - DML supports data transformation through functions and calculations within queries.
- **Data Integrity:**
  - DML statements should adhere to data integrity constraints defined using DDL (Data Definition Language).

Insert Command

- **Syntax:**
  a. **Spacify both the column name and the values to be inserted:**

```
INSERT INTO table_name
(column1, column2, column3, .....)
VALUES ( value1 , value2, value3,......);
```

- **Example:**

```
INSERT INTO Customers
(Customer_Name, Contact_Name, Address, City,
Postal_Code, Country)
VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21',
'Stavanger', '4006', 'Norway');
```

Insert Command

- **Syntax:**

  b. **If you are adding values for all the columns of the table, you do not need to specify the column names in the SQL query:**

  ```
  INSERT INTO table_name
  VALUES (value1, value2, value3, ...);
  ```

- **Example:**

  ```
  INSERT INTO Customers
  VALUES ('Cardinal', 'Tom B. Erichsen', 'Skagen 21',
  'Stavanger', '4006', 'Norway');
  ```

Insert Command

- **Syntax:**
    c. **Insert multiple rows:**
- **Example:**
    - INSERT INTO Customers (CustomerName, ContactName, Address, City, PostalCode, Country) VALUES
    - ('Cardinal', 'Tom B. Erichsen', 'Skagen 21', 'Stavanger', '4006', 'Norway'),
    - ('Greasy Burger', 'Per Olsen', 'Gateveien 15', 'San
      '4306', 'Norway');

Update Command

- **Syntax:**
  - UPDATE table_name
    SET column1= value1, column2= value2, ...
    WHERE condition;
- **Example:**
  - **Update Single record:**
  - UPDATE Customers
  - SET ContactName = 'Alfred Schmidt', City= 'Frankfurt'
  - WHERE CustomerID = 1;

Update Command

- **Update multiple record:**
  - UPDATE Customers
  - SET ContactName='Juan'
  - WHERE Country='Mexico';
- **Update Warning:**
  - Be carful when updating records. If you omit the where clause, All records will be updated!
  - **Example:**
  - UPDATE Customers
  - SET ContactName='Juan';

Delete Command

- **Syntax:**
  - ○ DELETE FROM table_name WHERE condition;
- **Example:**
  - ○ DELETE FROM Customers WHERE CustomerName='Alfreds Futterkiste';
- **Delete all records:**
  - ○ DELETE FROM table_name;
- **Example:**
  - ○ DELETE FROM Customers;

Truncate vs Delete

Truncate	Delete
Delete the content of table	Delete the content of table
It’s a DDL Command	It’s a DML Command
Can’t use where clause	Can use where clause to delete spaecific records
The data page is locked before removing the table data	A tuple is locked before removing it

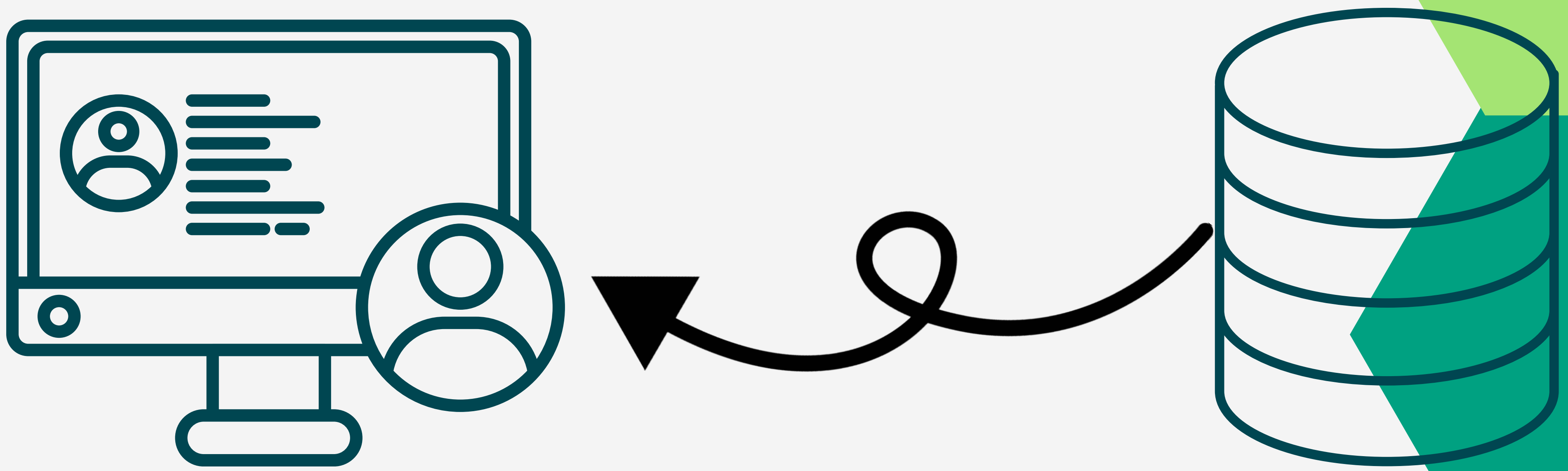
Truncate vs Delete

Truncate	Delete
Faster than delete	Slower than delete
Need alter permision on the table	need delete permision on the table
Can’t be used with indexed view	can be use with indexed view
Can’t active trigger	can active trigger
occupies less transaction spaces than delete	occupies more transaction spaces than truncate

Simple Queries

- **Syntax:**

  -
    ```
    SELECT <attribute list>
    FROM <table list>
    WHERE <condition>
    ORDER BY <attribute list>;
    ```



Simple Queries

- **Select all data:**
  - SELECT *
    FROM departments;
- **Select specific data:**
  - SELECT emp_id , emp_name, dept_id
    FROM employees
    WHERE location = “Cairo”;

Distict Keyword

- **It’s a row keyword that displays unique rows:**
- **Example:**
  - SELECT DISTINCT DNO FROM employees;

EmpNO	Name	DNO	JobID
100	Ahmed	2	Sales_Rep
200	Mai	2	It_PROG
300	Ali	3	Sales_Rep

DNO
2
3

Distict Keyword

- It’s a row keyword that displays unique rows:
- Example:
  - ○ SELECT DISTINCT DNO,JobID FROM employees;

EmpNO	Name	DNO	JobID
100	Ahmed	2	Sales_Rep
200	Mai	2	It_PROG
300	Ali	3	Sales_Rep

DNO	JobID
2	Sales_Rep
2	IT_Prog
3	Sales_Rep

Comparison & Logical Operators:

- **= , > , < --------> Single Row Operator:**
  - WHERE Salary >= 1500 and Salary <>2500;
  - WHERE Super_SSN = 321 OR Super_SSN = 321;
- **ALL, ANY, IN --------> Multi-row Operator:**
  - WHERE Super_SSN IN (321 , 223);
- **AND , OR ---------->Combained more than 1 condition**
  - WHERE Salary between 1500 and 2500;
- **Like ---------> Compare based on pattern**
  - WHERE fname like ‘?o*’;
  - ? ---------> 1 char
  - * ---------> Zero or more char

Arithmetic Expressions

- SELECT last_name , Salary , Salary+300 FROM Employees;
- Order of Precedence: *, /, +, -
  - You can enforce priority by adding parantheses
- SELECT last_name , Salary ,10* (Salary+300) as proposed salary FROM Employees;

$$\frac{x^a}{x^b} = x^{a-b}$$

Order by Clause

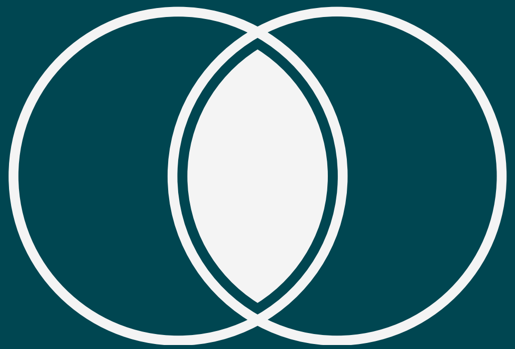
Used to sort the result-set in Ascending or Descending order (Asc order is the defult)

- **Syntax:**

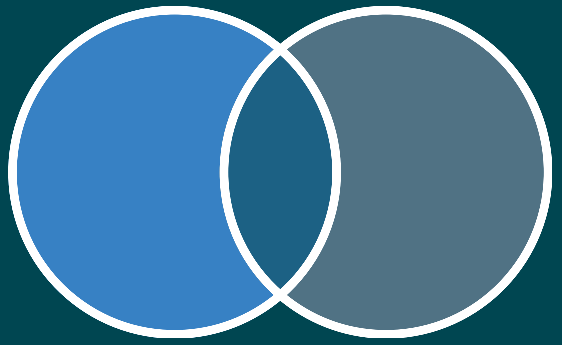
  -
    ```
    SELECT column1, column2, ...
     FROM table_name
     ORDER BY column1, column2, ... ASC|DESC;
    ```
- **Example:**

  -
    ```
    SELECT * FROM Products
     ORDER BY Price;
    ```

Types of Join



- **INNER JOIN:**
  - Returns rows when there is at least one match in both tables.
- **OUTER JOIN:**
  - Returns matched rows as well as rows when there is no match in one of the tables.



- **CARTESIAN PRODUCT:**
  - Returns all possible combination of rows.

Apply on Join

- Retrieve the name, address of all employees who work for Research Department

```
SELECT fname, Lname, Address
FROM Employee , Department
WHERE Dname= “research” and
Department.number = employee.Dno;
```

Table Alias:

- You can add alternative name (Alias) for Table name.
- Alias can be useful for long or complex table names.
- You can use table alias instead of table name to resolve ambiguity.

```
SELECT e.ID , d.name , e.name , e.Salary
FROM  department d , employee e
WHERE  d.id = e.deptid
ORDER BY  d.name;
```

Self Join:

- Is a join in which a table is joined with itself (Unary relationship), Specially when the table has a foreign key which referenced its own PK.
- To Join a table itself means that each row of the table is combined with itself and with every other row of the table.
- The self-join can be viewed as a join of 2 copies of the same table.
- Example:

  -
    ```
    SELECT e.name Employee_Name , s.name Supervisor
    FROM employee e , employee s
    WHERE e.supervisorID = s.ID;
    ```

Outer Join

Left Join

Return all rows from the tabl, even if there are no matches in the right table.

Right Join

Return all rows from the right table, even if there are no matches in the left table.

Full Outer

Returns rows when there is a match in one of the tables.

Apply on Outer Join:

- Display all departments' information even if they have no employees assigned.
- SELECT e.name AS Employee, d.dept_id, d.name AS Department FROM employees e RIGHT OUTER JOIN departments d ON e.dept_id = d.id;

Emp_name	Dept ID	Dept
Ahmed Ali	100	IT
Amr Samir	200	Marketing
Mona Selim	100	IT
	300	HR

Equi and Non-Equi Join

- **Equi join:** when the join condition is based on the equality operator (Ex. D.dno=e.deptno)

=

- **Non-Equi join:** when the join condition is based on any operator rather than the equality operator

> <

Non-Equi Join Example

- Display employees information (name , salary , title) with salary grade for each employee.

-
  ```
  SELECT e. name, e. salary, j.grade
   FROM employees e, job_ grades j
   WHERE  e. salary BETWEEN j.lowsal
   AND j.highsal;
  ```

LowSal	HighSal	Grade
1000	4000	B
4000	7000	A

Sub Queries

- Sub-Query (Nested Query): is a complete SELECT query inside another SELECT
- The Inner query is executed first then the outer query
- The inner query is usually placed in the WHERE or HAVING clauses
- Sometimes it is placed in the FROM clause and called “inline view”

Example

- Find the names of employees whose working locations are Giza

```
SELECT     name
FROM       Employee
WHERE      Dno     IN     (
                            SELECT Dnumber
                            FROM   Dept
                            WHERE Location = “Giza”
                          );
```

Example (Cont'd)

- Display department name for the highest paid employee

```
SELECT   dname
FROM     Dept
WHERE    deptno =    (
                      SELECT deptno
                      FROM   emp
                      WHERE sal = (
                                   SELECT MAX (sal)
                                   FROM emp
                                  )
                     );
```

Example (Cont'd)

- Find the names of employees whose salary is greater than the salary of the employees in department 5

```
SELECT      Lname , Fname
FROM        employee
WHERE       salary > ALL   (
                            SELECT salary
                            FROM   employee
                            WHERE Dno = 5
                           );
```

Union Operator:

- UNION operator is one of the set operators that unifies the output of two select statements into one result.
- Two conditions must apply for the two selects:
  - Same number of columns
  - Same data type of columns
- The result displays the names of the first query.
- If one row is contained in the result of the two selects it is displayed only once.
- UNION ALL is functionally equivalent to UNION operator, except that it doesn’t eliminate repetition in the result set.

Union Operator Example

- Find the departments numbers whose location in GIZA or manager no = 10

```
SELECT Dnumber
FROM depatment
WHERE MGRSSN = 10;
```

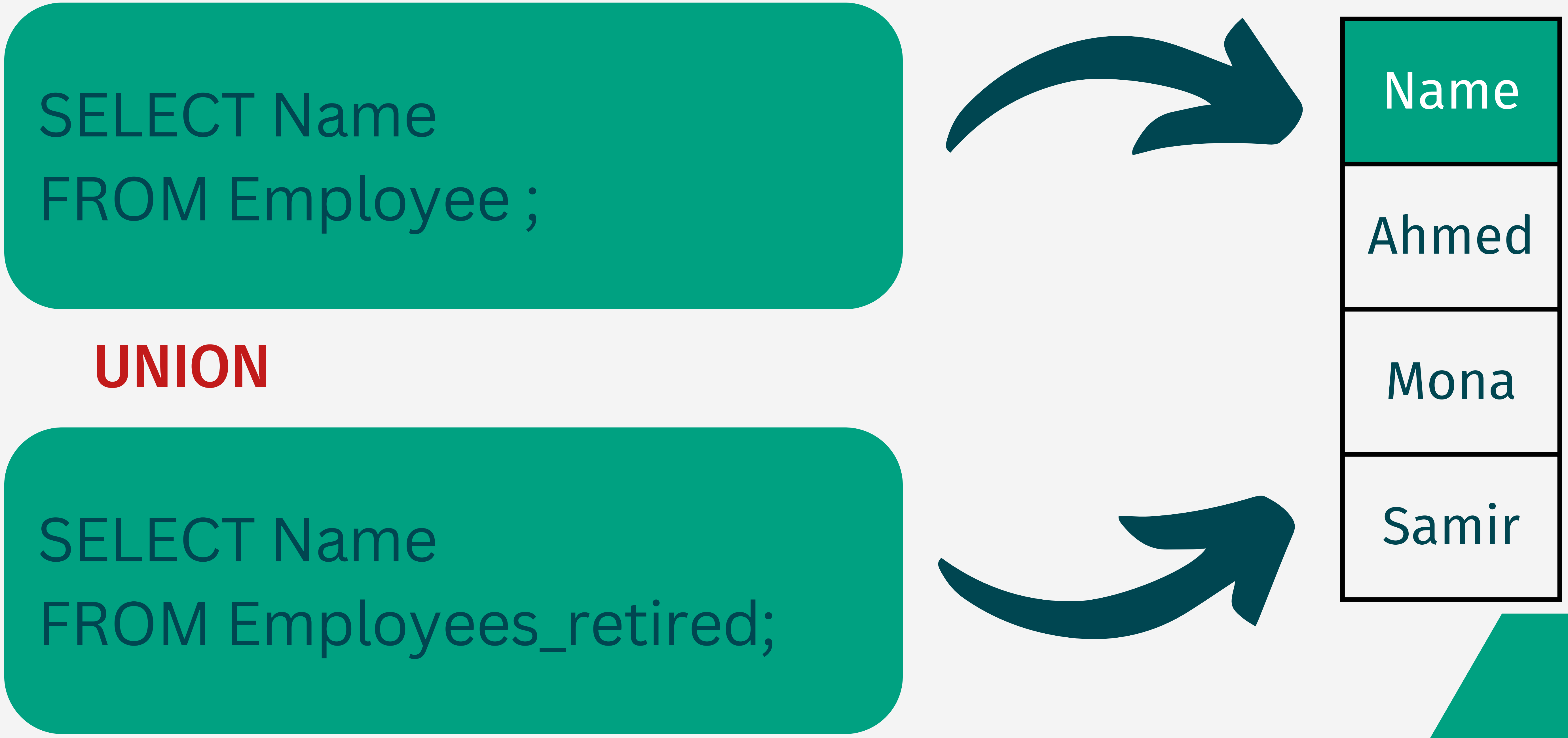
UNION

```
SELECT Dnumber
FROM dept_locations
WHERE dlocation = “GIZA”;
```

One Result

Union Operator Example

- Display names of current employees as well as previous employees



```
SELECT Name
FROM Employee ;
```

UNION

```
SELECT Name
FROM Employees_retired;
```

Name
Ahmed
Mona
Samir

Correlated Sub-Query:

- When the inner query references one of the attributes of the outer query.
- The inner query is executed once per row of the outer query
- **Example:**

  -
    ```
    SELECT product_name, list_price, category_id
    FROM production.products p1
    WHERE list_price IN ( SELECT MAX (p2.list_price)
                          FROM production.products p2
                          WHERE p2.category_id = p1.category_id
                          GROUP BY p2.category_id )
    ORDER BY category_id, product_name;
    ```

Exists Keyword

- The EXISTs keyword is used with correlated sub-queries

- The EXISTS condition is considered "to be met" if the sub-query returns at least one row.

- **Syntax**
    - SELECT columns
      FROM tables
      WHERE EXISTS (sub-query);

Exists Keyword Example

- Display suppliers' information who has orders.

  -
    ```
    SELECT *
    FROM suppliers
    WHERE EXISTS (
                    SELECT *
                    FROM orders
                    WHERE
                        suppliers.supplier_id = orders.supplier_id
                   );
    ```

Exists Keyword Example

- Retrieve the name of employees who have no dependents

  -
    ```
    SELECT name
    FROM employee
    WHERE  NOT EXISTS (
                        SELECT *
                        FROM dependent
                        WHERE ssn=Essn
                       );
    ```

Exists Condition with DML

-
  ```
  DELETE FROM suppliers
  WHERE NOT EXISTS (
                      SELECT *
                      FROM orders
                      WHERE
                         suppliers.supplier_id =    orders.supplier_id
                     );
  ```

Aggregate Functions:

- Aggregate Functions (group functions): perform a specific operation on a number of rows and return one result per group

- Example:
  - COUNT, SUM, MAX, MIN, and AVG

- Group Functions ignore null values in the columns.

Aggregate Functions Example:

- Find the sum, maximum, minimum and average salaries of all employees.

```
SELECT SUM (salary), MAX (salary), MIN (salary), AVG (salary)
FROM Employees;
```

- Find the total number of employees in the company?

```
SELECT COUNT(*)
FROM employees;
```

- Find the number of employees in the research department?

```
SELECT COUNT(*)
FROM employee , department
WHERE dno=dnumber AND dname ='Research';
```

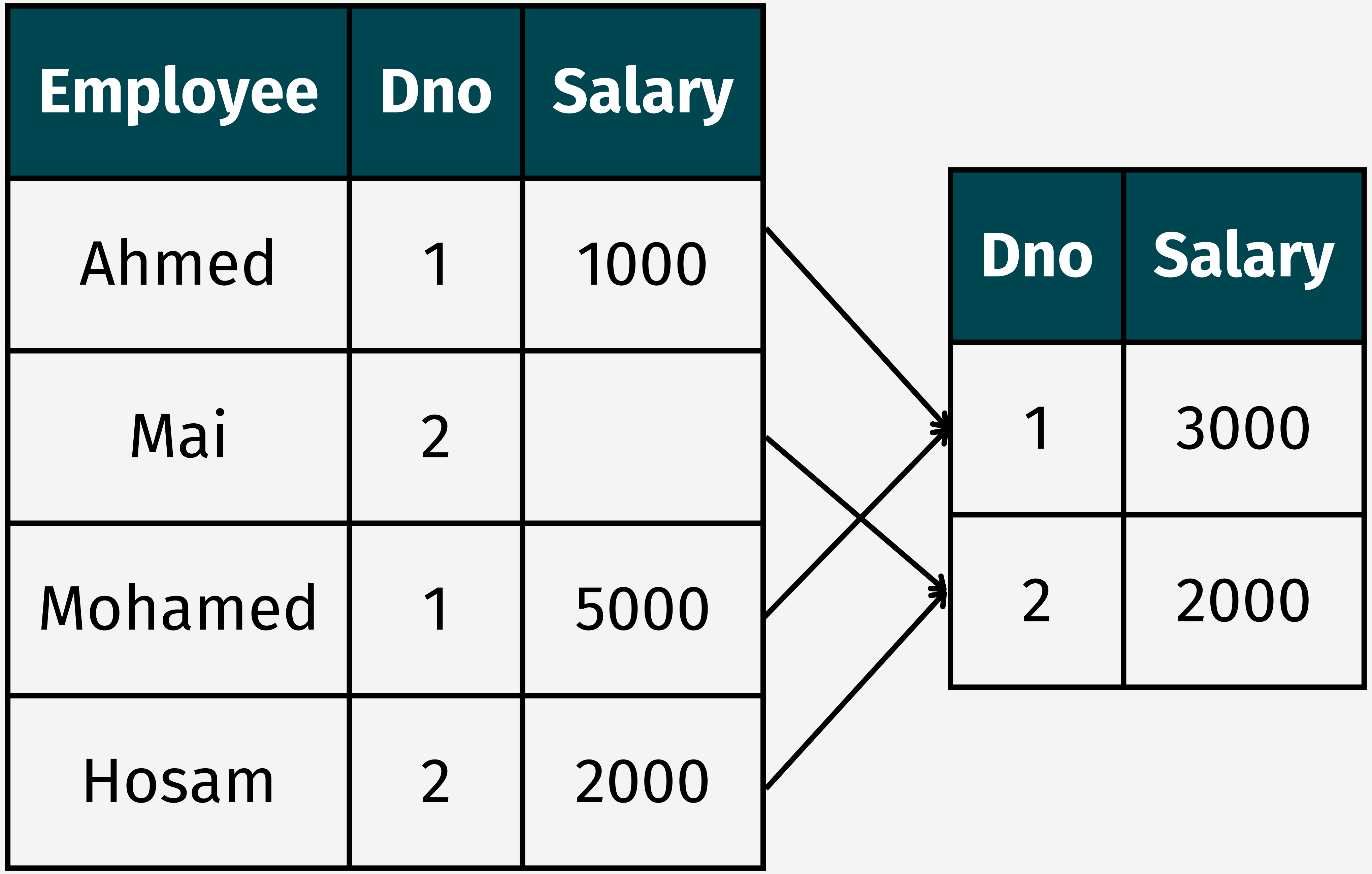
Grouping

- If you want to apply aggregate functions to subgroups of tuples, use GROUP BY clause
- GROUP BY clause must be used in conjunction with aggregate functions
- You can filter group results using HAVING clause
- HAVING clause is used for applying conditions on group functions

Grouping Examples

- For each department, retrieve the department number, the number of employees in the department, and their average salaries.

```
SELECT  dno , COUNT (*)
, AVG (salary)
FROM    employee
GROUP BY dno
```



Employee	Dno	Salary
Ahmed	1	1000
Mai	2	
Mohamed	1	5000
Hosam	2	2000

Dno	Salary
1	3000
2	2000

Grouping Examples

- For each project on which more than two employees work, retrieve the project number, the project name , and the number of employees who work on the project

```
SELECT pnumber, pname ,count (works_on.pno)
FROM project , Works_on
WHERE Pnumber = Pno
GROUP BY pnumber, pname
HAVING COUNT (*) > 2
ORDER BY Pnumber;
```

Data Control Language

- **Access Control:**
  - DCL is primarily concerned with managing who can access specific data or perform certain actions within a database.
  - It defines and enforces security and authorization rules.
- **Two Main DCL Statements:**
  - There are two main DCL statements in SQL: GRANT and REVOKE.
  - These statements are used to grant or revoke specific privileges to users or roles.

Data Control Language

Grant

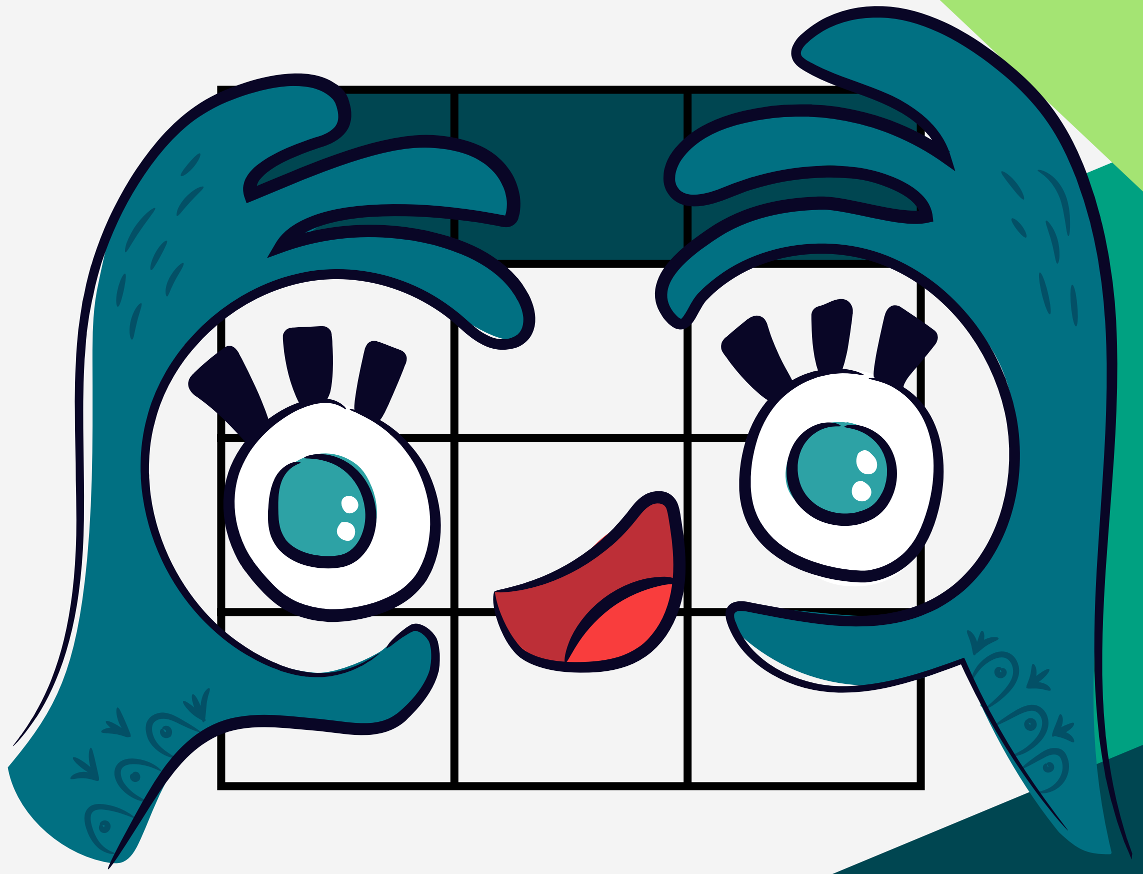
- GRANT SELECT ON TABLE employees TO Ahmed;
- GRANT ALL ON TABLE department TO Mary, Ahmed;
- GRANT SELECT ON TABLE employees TO Ahmed WITH GRANT OPTION;

Revoke

- REVOKE UPDATE ON TABLE department FROM Mary;
- REVOKE ALL ON TABLE department FROM Mary, Ahmed;

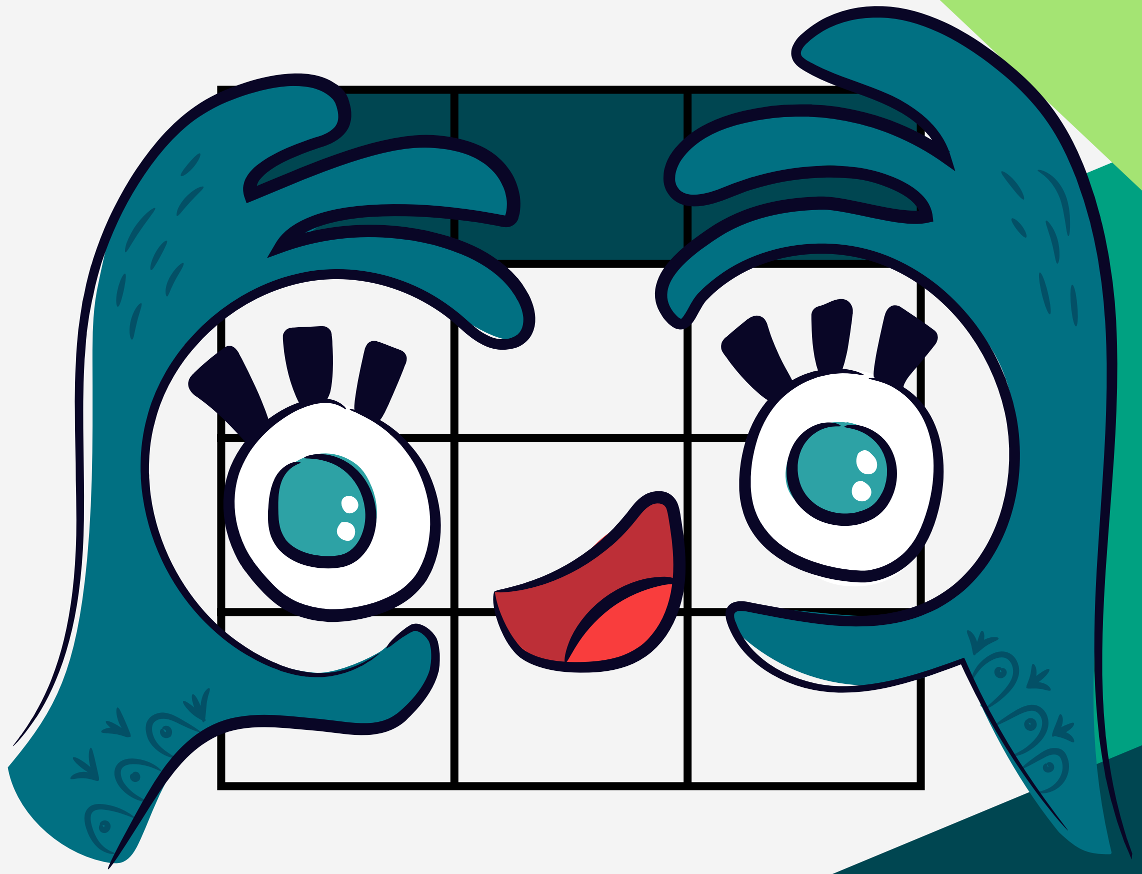
Views:

- A view is a logical table based on a table or another view
- A view contains no data of its own, but is like a window through which data from tables can be viewed or changed
- The tables on which a view is based are called base tables
- The view is stored as a SELECT statement in the data dictionary.



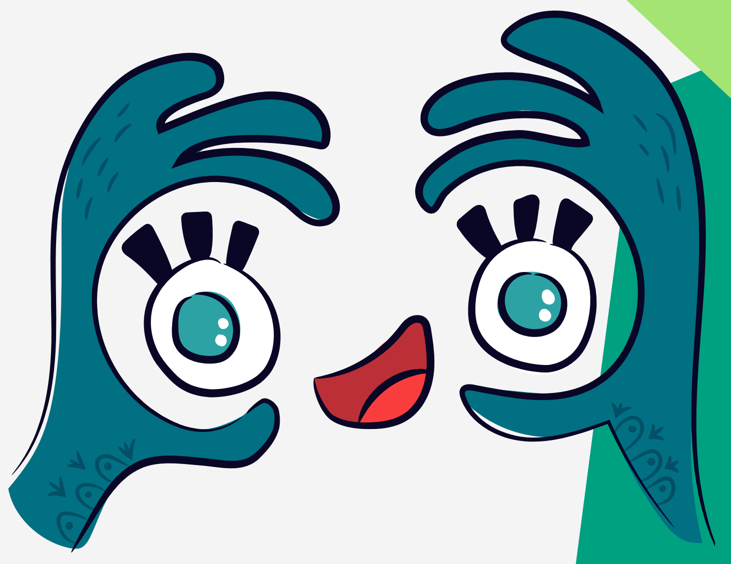
Advantages of Views:

- Restrict data access
- Make complex queries easy
- Provide data independence
- Present different views of the same data



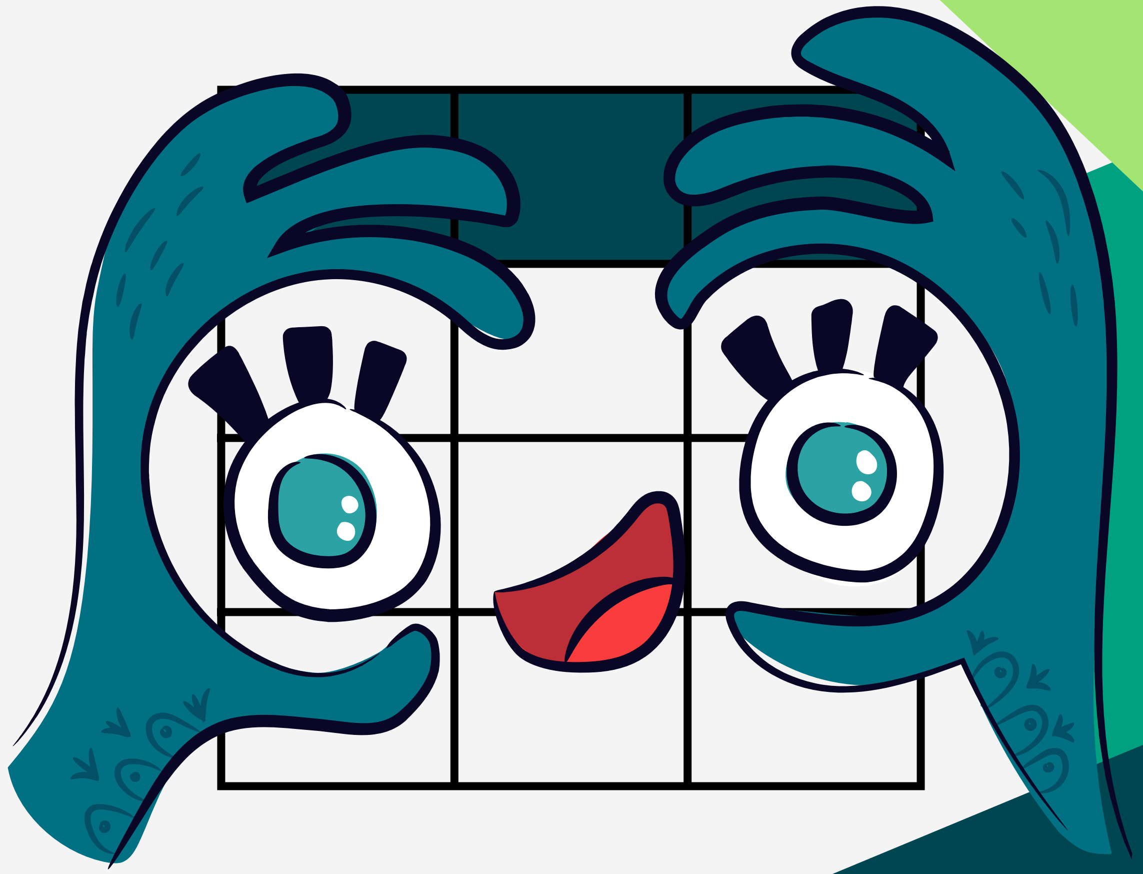
Simple Views and Complex Viwes:

Feature	Simple View	Complex View
Number of Tables	One	One or more
Contain Functions	✗	✓
Contain groups of data	✗	✓
DML operations through a view	✓	Not always



Creating Views:

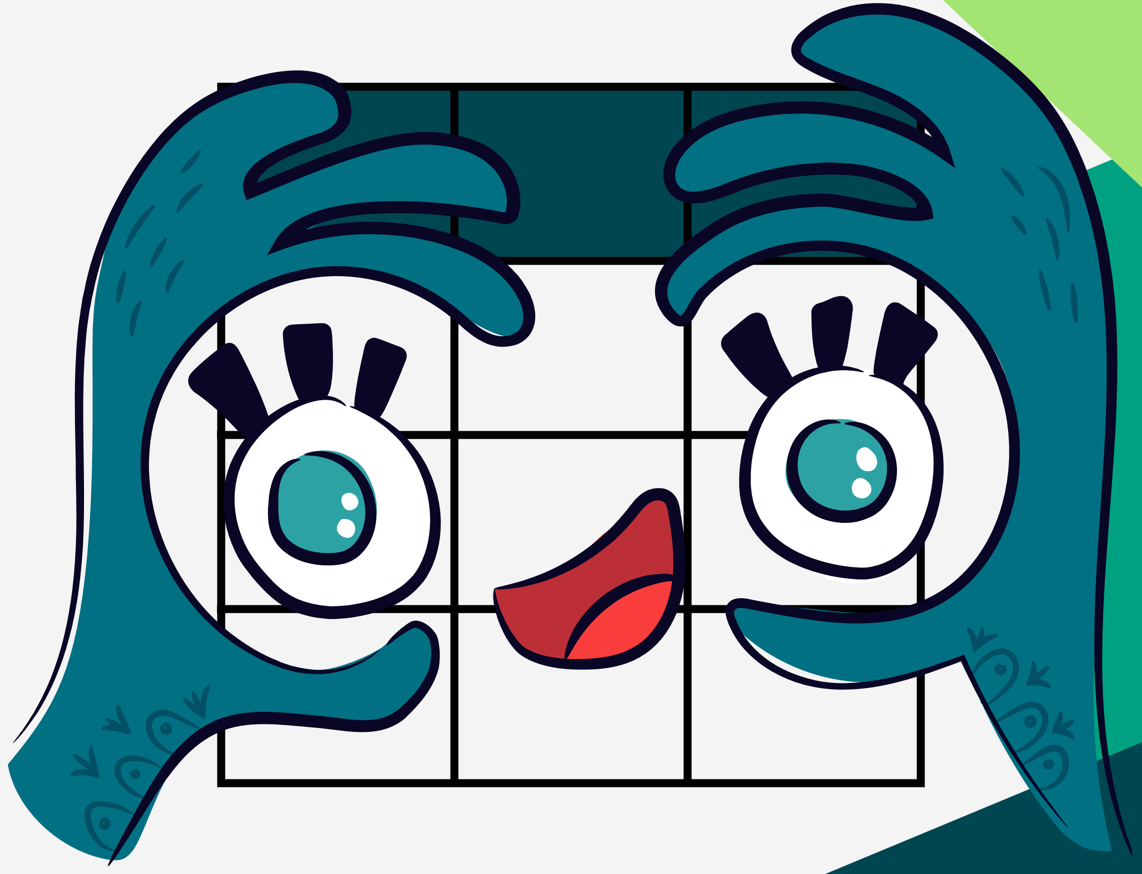
- CREATE VIEW view [(column 1 [, column2] ...)]
  AS subquery
  [With Check Option] ;



Creating Views (Cont'd):

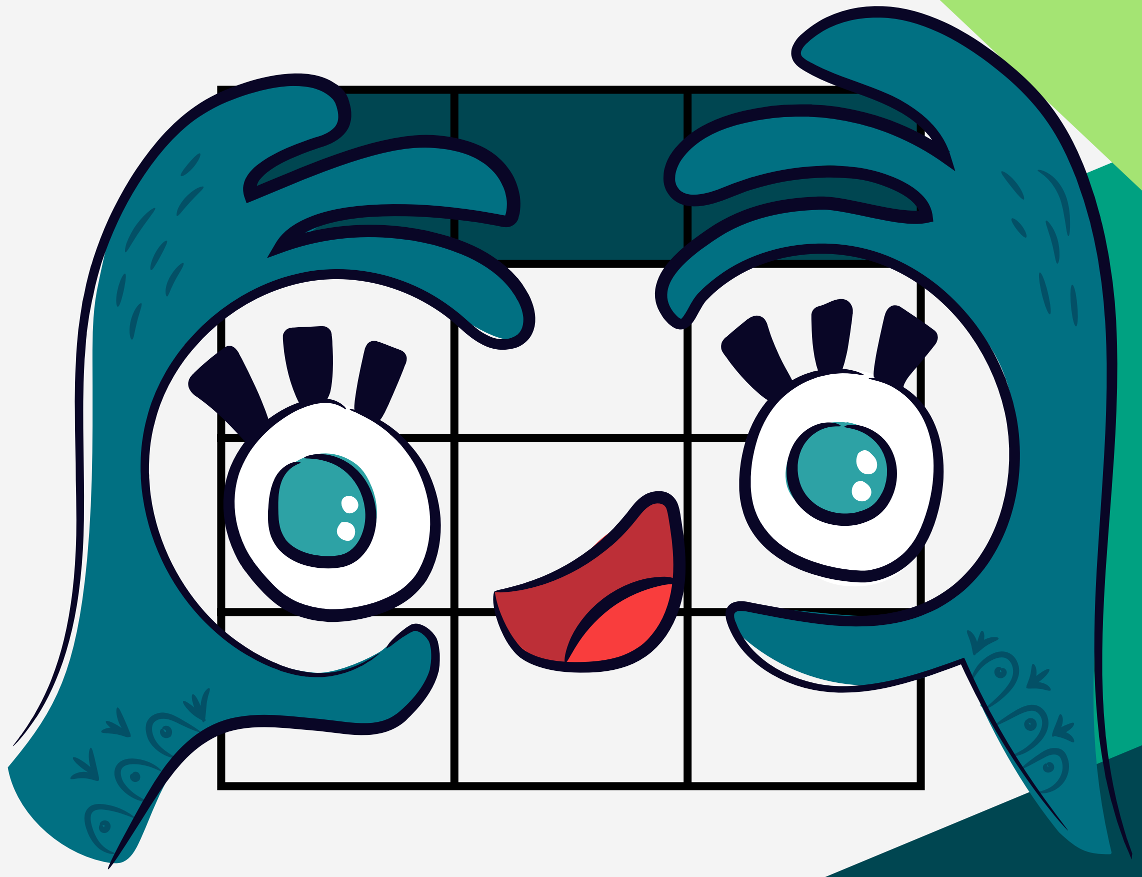
- Create a view to display employee names and total hours employee worked on a project

```
CREATE VIEW vw_work_hrs
AS
SELECT Fname , Lname , Pname , Hours
FROM Employee, Project , Works_on
WHERE SSN=ESSN AND PNO=PNUMBER;
```



Views with Check Option:

-
  ```
  CREATE VIEW Suppliers
   AS
  SELECT *
  FROM suppliers
  WHERE status > 15
  WITH CHECK OPTION;
  ```



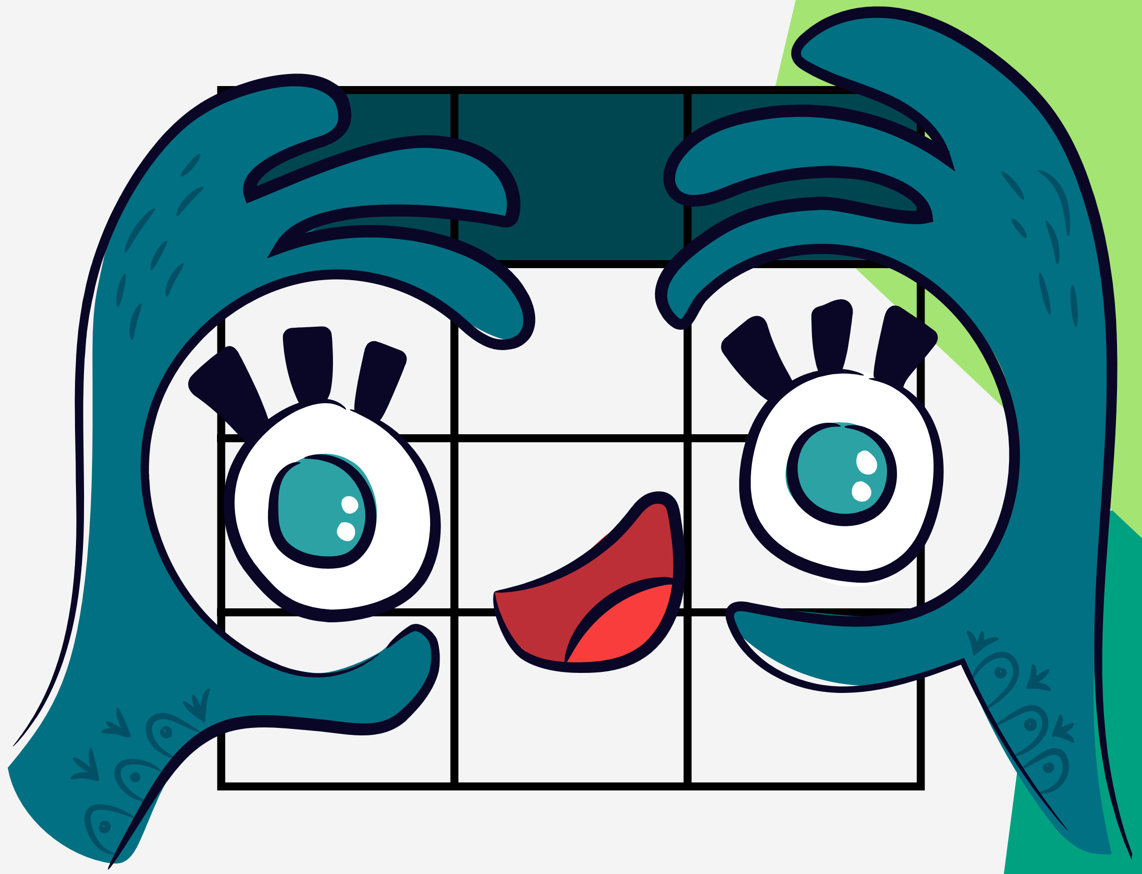
Modifying a View:

- **Syntax:**

  -
    ```
    CREATE OR REPLACE VIEW view_name
    AS
    Sub-query
    ```

- **Example:**

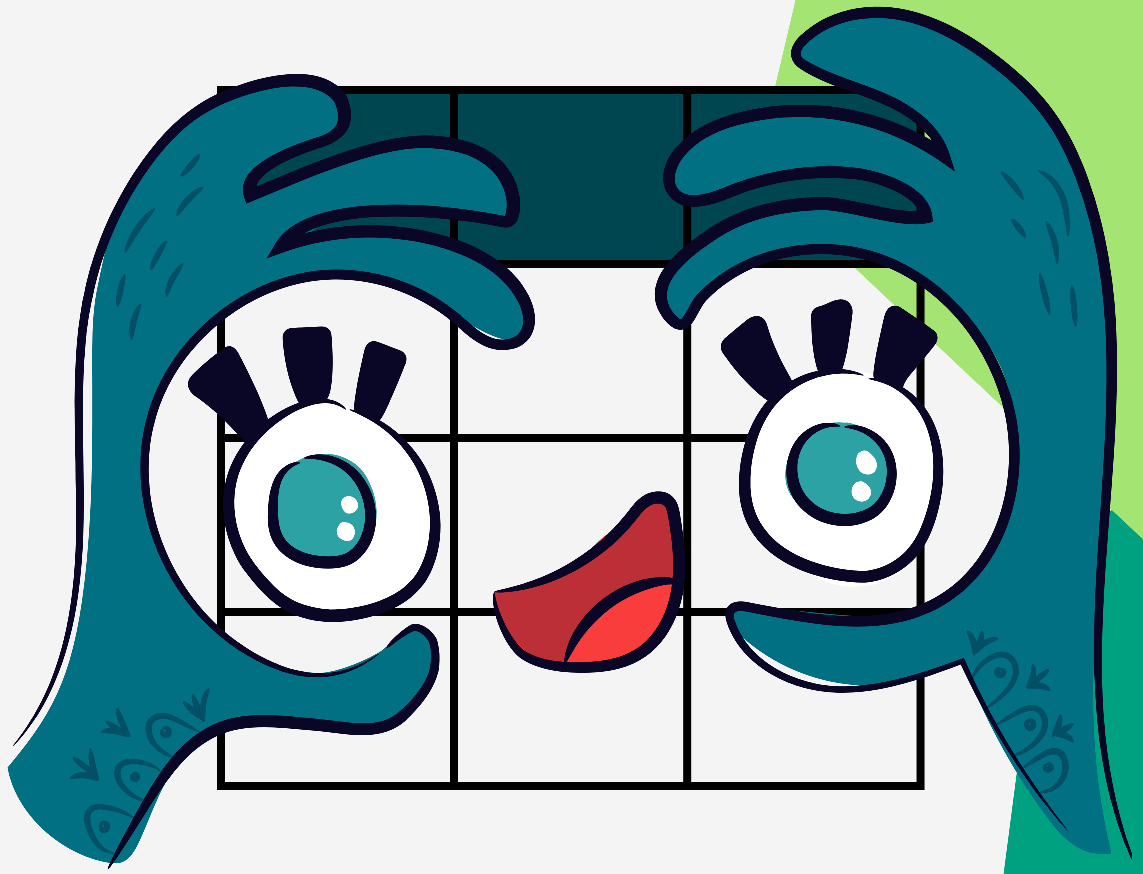
  -
    ```
    CREATE OR REPLACE VIEW vw_work_hrs
    AS
    SELECT Fname , Lname , Pname , Hours
    FROM Employee, Project , Works_on
    WHERE SSN=ESSN AND PNO=PNUMBER AND Dno = 5;
    ```



Removing a View:

- **Syntax:**
  - DROP VIEW view_name;

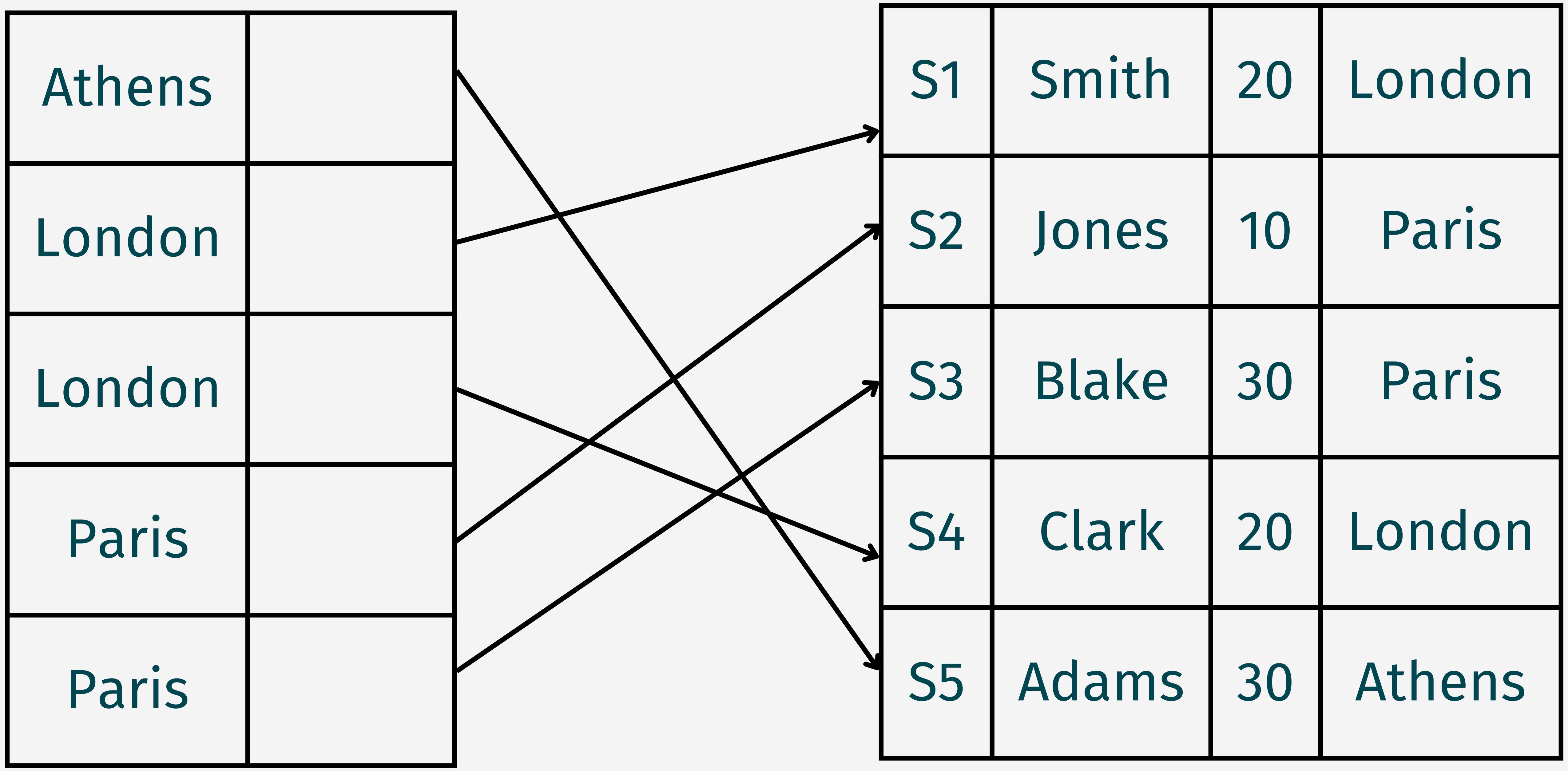
- **Example:**
  - DROP VIEW vw_work_hrs



Indexes:

- They are used to speed up the retrieval of records in response to certain search conditions.
- May be defined on multiple columns
- Can be created by the user or by the DBMS
- Are used and maintained by the DBMS

Indexes (Cont'd):



Athens	
London	
London	
Paris	
Paris	

S1	Smith	20	London
S2	Jones	10	Paris
S3	Blake	30	Paris
S4	Clark	20	London
S5	Adams	30	Athens

File (index)

Supplier file (data)

Indexes (Cont’d):

When to create an index:	
✓	A column contains a wide range of values
✓	A column contains a large number of null values
✓	One or more columns are frequently used together in a WHERE clause or a join condition
✓	The table is large and most queries are expected to retrieve less than 2% to 4% of the rows in the table

Indexes (Cont’d):

When to don’t create an index	
✗	The columns are not often used as a condition in the query
✗	The table is small or most queries are expected to retrieve more than 2% to 4% of the rows in the table
✗	The table is updated frequently
✗	The indexed columns are referenced as part of an expression -- [Upper(Lname)]

Indexes (Cont'd):

- **Creation**
  - CREATE INDEX index_name ON Table_name (column_name);
  - CREATE INDEX emp_inx ON Employee (Salary);
- **Removing**
  - DROP INDEX index_name;
  - DROP INDEX emp_inx;

SQL Tutorials

- **Oracle SQL URL:**
    - beginner-sql-tutorial.com
- **Sybase SQL URL:**
    - http://infocenter.sybase.com/help/index.jsp
- **ANSI SQL URL:**
    - http://www.w3schools.com/SQL/sql_join.asp
- **MS SQL URL:**
    - http://msdn.microsoft.com/en-us/library/bb264565.aspx
- **IBM Informix SQL:**
    - http://publib.boulder.ibm.com/infocenter/idshelp/v10/index.jsp?topic=/com.ibm.sqlt.doc/sqltmst104.htm

Questions?