KOÇ UNIVERSITY

Database Management Systems

Advanced SQL

M. Emre Gürsoy

Assistant Professor
Department of Computer Engineering
www.memregursoy.com

Outline

- In this lecture, we will cover some more advanced issues related to SQL.
  - NULLs and three-valued logic (3VL)
  - Nested SQL queries
  - Different types of JOINs
  - Aggregate operators: COUNT, SUM, MIN, MAX, ...
  - Grouping (GROUP BY – HAVING)
  - Triggers and views

NULLs and 3VL

NULL

- **NULL:** A reserved word in SQL to mean "value does not exist" or "unknown" or "could be anything".
  - Not every field is allowed to be NULL
    - E.g., primary key
    - Subject to table definition and integrity constraints

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Shero	shero@cs	18	NULL
53650	Shero	NULL	19	3.8

```
CREATE TABLE EMPLOYEE
        ( Fname                 VARCHAR(15)         NOT NULL,
          Minit                 CHAR,
          Lname                 VARCHAR(15)         NOT NULL,
          Ssn                   CHAR(9)             NOT NULL,
          Bdate                 DATE,
          Address               VARCHAR(30),
          Sex                   CHAR,
          Salary                DECIMAL(10,2),
          Super_ssn             CHAR(9),
          Dno                   INT                 NOT NULL,
        PRIMARY KEY (Ssn),
CREATE TABLE DEPARTMENT
        ( Dname                 VARCHAR(15)         NOT NULL,
          Dnumber               INT                 NOT NULL,
          Mgr_ssn               CHAR(9)             NOT NULL,
          Mgr_start_date        DATE,
        PRIMARY KEY (Dnumber),
        UNIQUE (Dname),
        FOREIGN KEY (Mgr_ssn) REFERENCES EMPLOYEE(Ssn) );
```

Checking for NULLs

- You should **NOT** use comparison operators or arithmetic operators with NULL.
  - Avoid: x = NULL or x < NULL or x <> NULL
  - These evaluate to "Unknown" (neither True nor False)
  - Not even NULL = NULL
- Instead, use: IS NULL and IS NOT NULL

Query 18. Retrieve the names of all employees who do not have supervisors.

```
Q18:  SELECT  Fname, Lname
      FROM    EMPLOYEE
      WHERE   Super_ssn IS NULL;
```

3VL

- SQL uses three-valued logic (3VL)
  - True, False, Unknown

- Truth tables for 3VL:

p	*q*	*p* OR *q*	*p* AND *q*	*p* = *q*
True	True	True	True	True
True	False	True	False	False
True	Unknown	True	Unknown	Unknown
False	True	True	False	False
False	False	False	False	True
False	Unknown	Unknown	False	Unknown
Unknown	True	True	Unknown	Unknown
Unknown	False	Unknown	False	Unknown
Unknown	Unknown	Unknown	Unknown	Unknown

p	NOT *p*
True	False
False	True
Unknown	Unknown

3VL

```
SELECT *
FROM t
WHERE i = NULL;
```

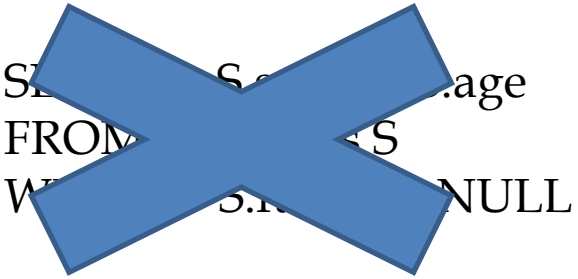
- A common mistake!
- **WHERE** clause always evaluates to Unknown, which is not equal to True. So, this SQL query returns nothing.
  - That's why you should use IS NULL instead

- (ps: In some practical SQL implementations, the above may behave the same as "IS NULL", but that's an engineering fix.)

Example

- Sailors (sid, sname, rating, age)
- Boats (bid, bname, color)
- Reserves (sid, bid, day)

- Find the names and ages of sailors who do not yet have a rating.



```
SELECT   S.sname, S.age
FROM     Sailors S
WHERE    S.rating IS NULL
```

Nested SQL Queries

Nested Queries

- Nested SQL query: SQL query inside an SQL query
  - Nested queries often appear in the WHERE clause
- Useful keywords when building nested queries: IN, NOT IN, ALL, ANY, EXISTS, NOT EXISTS, UNIQUE

```
Q4A:  SELECT   DISTINCT Pnumber
      FROM     PROJECT
      WHERE    Pnumber IN
               ( SELECT     Pnumber
                 FROM       PROJECT, DEPARTMENT, EMPLOYEE
                 WHERE      Dnum=Dnumber AND
                            Mgr_ssn=Ssn AND Lname='Smith' )
               OR
               Pnumber IN
               ( SELECT     Pno
                 FROM       WORKS_ON, EMPLOYEE
                 WHERE      Essn=Ssn AND Lname='Smith' );
```

"IN", "NOT IN"

- IN and NOT IN are comparison operators
  - Compares a value v with a set (or multiset) V
  - IN returns TRUE if $v \in V$, FALSE otherwise
  - The opposite holds for NOT IN

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     Pno IN (1, 2, 3);
```

- You can also do "tuple comparisons" (use parantheses):

```
SELECT    DISTINCT Essn
FROM      WORKS_ON
WHERE     (Pno, Hours) IN ( SELECT    Pno, Hours
                            FROM      WORKS_ON
                            WHERE     Essn='123456789' );
```

"ALL", "ANY"

- ALL: value must exceed all values from nested query

```
SELECT   Lname, Fname
FROM     EMPLOYEE
WHERE    Salary > ALL  ( SELECT   Salary
                         FROM     EMPLOYEE
                         WHERE    Dno=5 );
```

- ANY: value should exceed any value from nested query
  - "Find sailors whose rating is greater than some sailor called Charlie"

```
SELECT  *
FROM  Sailors S
WHERE  S.rating > ANY  ( SELECT  S2.rating
                         FROM  Sailors S2
                         WHERE S2.sname='Charlie' )
```

"EXISTS", "NOT EXISTS"

- EXISTS: check whether the result of the nested query is empty or not.
- NOT EXISTS: logically opposite of EXISTS
- EXISTS and NOT EXISTS are Boolean operators, i.e., they return TRUE or FALSE

Find the first and last names of managers who have at least one dependent.

```
SELECT Fname, Lname
FROM Employee
WHERE EXISTS (SELECT *
              FROM Dependent
              WHERE Ssn = Essn)

AND EXISTS (SELECT   *
            FROM Department
            WHERE Ssn = Mgr_Ssn)
```

Referring to Employee.Ssn

"EXISTS", "NOT EXISTS"

- EXISTS: check whether the result of the nested query is empty or not.
- NOT EXISTS: logically opposite of EXISTS
- EXISTS and NOT EXISTS are Boolean operators, i.e., they return TRUE or FALSE

Retrieve the names of employees who have no dependents.

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( SELECT    *
                       FROM      DEPENDENT
                       WHERE     Ssn = Essn )
```

"UNIQUE"

- UNIQUE: check for duplicate tuples.
  - If the set contains duplicates, return FALSE
  - Otherwise, return TRUE

Find the names of sailors who reserved a boat at most once.

```
SELECT  S.sname
FROM  Sailors S
WHERE   UNIQUE (     SELECT  R.bid
                     FROM  Reserves R
                     WHERE   S.sid=R.sid)
```

Division in SQL

Find the first and last names of employees who work on ALL projects controlled by department number 5.

- Logically equivalent to: "Find the first and last names of employees for whom there does NOT EXIST a project controlled by dept no 5 that this employee does not work on.

```
SELECT    Fname, Lname
FROM      EMPLOYEE
WHERE     NOT EXISTS ( ( SELECT    Pnumber
                         FROM      PROJECT
                         WHERE     Dnum = 5)
                       EXCEPT    ( SELECT    Pno
                                   FROM      WORKS_ON
                                   WHERE     Ssn = Essn) )
```

Division in SQL

Find the names of sailors who have reserved all boats.

```
SELECT  S.sname
FROM    Sailors S
WHERE  NOT EXISTS ((SELECT  B.bid
                    FROM  Boats B)
                    EXCEPT ( SELECT  R.bid
                             FROM    Reserves R
                             WHERE  R.sid =  S.sid))
```

Types of JOINs

INNER JOIN

- **INNER JOIN**s are the "default" kind of join

```
SELECT   *
FROM     Employee, Department
WHERE    Employee.DepartmentID = Department.DepartmentID
```

```
SELECT   *
FROM     Employee INNER JOIN Department
         ON Employee.DepartmentID = Department.DepartmentID
```

Employee table

LastName	DepartmentID
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34
Jasper	36

DepartmentID	DepartmentName
31	Sales
33	Engineering
34	Clerical
35	Marketing

Department table

NATURAL JOIN

- Result of INNER JOIN:

Employee.LastName	Employee.DepartmentID	Department.DepartmentName	Department.DepartmentID
Smith	34	Clerical	34
Jones	33	Engineering	33
Robinson	34	Clerical	34
Steinberg	33	Engineering	33
Rafferty	31	Sales	31

- You can eliminate duplicate DepartmentID column by doing a **NATURAL JOIN**. (column name **must** be same)

```
SELECT *
FROM Employee NATURAL JOIN Department
```

Employee.LastName	DepartmentID	Department.DepartmentName
Smith	34	Clerical
Jones	33	Engineering
Robinson	34	Clerical
Steinberg	33	Engineering
Rafferty	31	Sales

NATURAL JOIN

- If you want to do a NATURAL JOIN but column names are not the same, you can use renaming.

- Employee table has Dno, Department table has Dnumber
- Temporarily "rename" Dnumber in Department to Dno so that Employee.Dno = Department.Dno can be executed by the NATURAL JOIN

```
SELECT   Fname, Lname, Address
FROM     (EMPLOYEE NATURAL JOIN
          (DEPARTMENT AS DEPT (Dname, Dno, Mssn, Msdate)))
WHERE    Dname = 'Research';
```

OUTER JOIN

- **INNER JOIN:** Tuple is included in the result only if a matching tuple exists in the other relation.
- **LEFT OUTER JOIN:** Every tuple in the left table **must** appear in the result. If no matching tuple, pad with NULLs.

```
SELECT *
FROM Employee LEFT OUTER JOIN Department
        ON Department.DepartmentID = Employee.DepartmentID
```

Employee. LastName	Employee. DepartmentID	Department. DepartmentName	Department. DepartmentID
Jones	33	Engineering	33
Rafferty	31	Sales	31
Robinson	34	Clerical	34
Smith	34	Clerical	34
Jasper	36	NULL	NULL
Steinberg	33	Engineering	33

JOINs

- **INNER JOIN:** Tuple is included in the result only if a matching tuple exists in the other relation.
- **NATURAL JOIN:** Inner join with an implicit column equality condition (column names must be the same!).
- **LEFT OUTER JOIN:** Every tuple in the left table **must** appear in the result. If no matching tuple, pad with NULLs.
- **RIGHT OUTER JOIN:** Every tuple in the right table **must** appear in the result. If no matching tuple, pad with NULLs.
- **FULL OUTER JOIN:** "Combine" the result of LEFT OUTER JOIN and RIGHT OUTER JOIN.
- **CROSS JOIN:** Equivalent of Cartesian product from relational algebra.

SELECT *
FROM Employee XYZ JOIN Department

Aggregate Operators

Aggregate Operators

- Used to summarize information from multiple tuples into a short summary.
- Commonly used for reporting purposes:
  - Total sales in year 2004
  - Average salary of employees
  - Number of employees hired/fired in 2004
  - ...

```
COUNT (*)
COUNT ( [DISTINCT] A)
SUM ( [DISTINCT] A)
AVG ( [DISTINCT] A)
MAX (A)
MIN (A)
```

Aggregate Operators

- Total number of sailors in the club?

```
SELECT  COUNT (*)
FROM    Sailors S
```

COUNT (*)
COUNT ([DISTINCT] A)
SUM ([DISTINCT] A)
AVG ([DISTINCT] A)
MAX (A)
MIN (A)

- Average [distinct] age of sailors in the club whose rating is 10?

```
SELECT   AVG (S.age)
FROM     Sailors S
WHERE    S.rating=10
```

```
SELECT  AVG (DISTINCT S.age)
FROM  Sailors S
WHERE  S.rating=10
```

- Names of sailors whose rating is equal to the max rating in the club?

How about the # of sailors with max rating?

```
SELECT  S.sname
FROM    Sailors S
WHERE  S.rating  = (SELECT  MAX(S2.rating)
                    FROM  Sailors S2)
```

Aggregate Operators

- Summarize the min, max, sum, avg salary of EMPLOYEEs.

```
SELECT    SUM (Salary) AS Total_Sal, MAX (Salary) AS
          Highest_Sal, MIN (Salary) AS Lowest_Sal, AVG
                (Salary) AS Average_Sal
FROM      EMPLOYEE;
```

Query 20. Find the sum of the salaries of all employees of the 'Research' department, as well as the maximum salary, the minimum salary, and the average salary in this department.

```
Q20:  SELECT   SUM (Salary), MAX (Salary), MIN (Salary), AVG (Salary)
      FROM     (EMPLOYEE JOIN DEPARTMENT ON Dno=Dnumber)
      WHERE    Dname='Research';
```

Queries 21 and 22. Retrieve the total number of employees in the company (Q21) and the number of employees in the 'Research' department (Q22).

```
Q21:  SELECT   COUNT (*)
      FROM     EMPLOYEE;

Q22:  SELECT   COUNT (*)
      FROM     EMPLOYEE, DEPARTMENT
      WHERE    DNO=DNUMBER AND DNAME='Research';
```

GROUP BY - HAVING

GROUP BY – HAVING

- Sometimes we want to group tuples in our relations.
- ***Example:*** Find the age of the youngest sailor for each rating level.
  - We don't know how many rating levels exist
  - Even if we do, we must write a new query to find the age of the youngest sailor for each level
- Instead, we use GROUP BY – HAVING clauses.

For $i = 1, 2, \ldots, 10$:

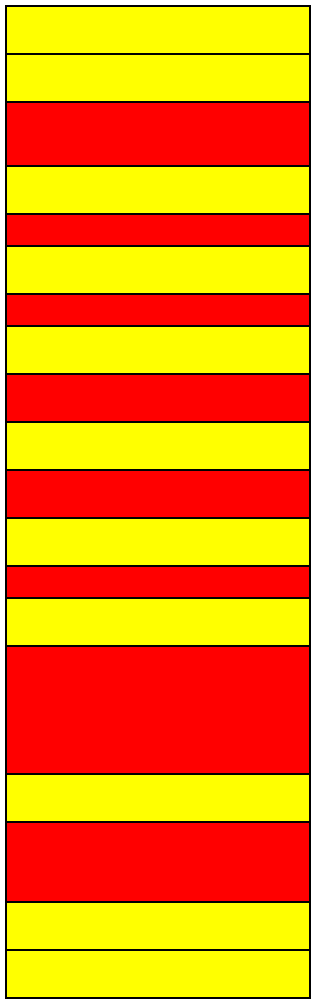
```
SELECT  MIN (S.age)
FROM  Sailors S
WHERE  S.rating = i
```

```
SELECT      [DISTINCT]  target-list
FROM        relation-list
WHERE       qualification
GROUP BY    grouping-list
HAVING      group-qualification
```

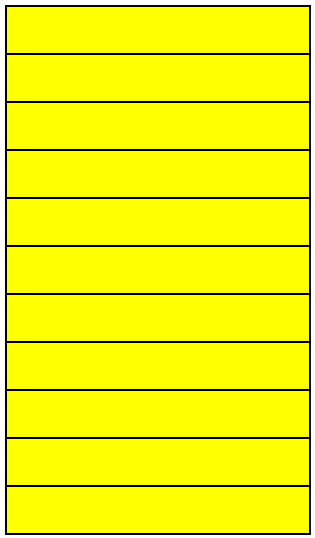
Conceptual Evaluation

SELECT *target-list*
FROM *relation-list*
WHERE *qualification*

SELECT [DISTINCT] *target-list*
FROM *relation-list*
WHERE *qualification*
GROUP BY *grouping-list*
HAVING *group-qualification*



GROUP BY *grouping-list*

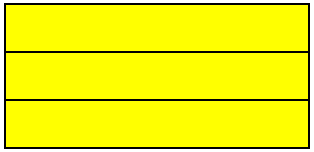


HAVING *group-qualification*

gr1

gr2

gr3



gr4

gr5

RESULT

Conceptual Evaluation

```
SELECT  S.rating, MIN(S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

```
FROM  Sailors S
WHERE  S.age >= 18
```

Age = 20
Age = 25
Age = 19
Age = 70
Age = 60
Age = 40
Age = 33
Age = 32
Age = 18
Age = 22
Age = 39

GROUP BY *S.rating*

Rating=4
Rating=2
Rating=3
Rating=2
Rating=3
Rating=5
Rating=1
Rating=4
Rating=3
Rating=4
Rating=1

HAVING COUNT (*) > 1

gr1: Rating=1, Rating=1

gr2: Rating=2, Rating=2

gr3: Rating=3, Rating=3, Rating=3

gr4: Rating=4, Rating=4, Rating=4

gr5: Rating=5

RESULT

Rating = 1	**33**
Rating = 2	**25**
Rating = 3	**18**
Rating = 4	**20**

Exercises

- For sailors with age >= 18 and for each rating level with at least two sailors satisfying this age criteria, find the age of the youngest sailor.

```
SELECT  S.rating, MIN(S.age)
FROM  Sailors S
WHERE  S.age >= 18
GROUP BY  S.rating
HAVING  COUNT (*) > 1
```

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
71	zorba	10	16.0
64	horatio	7	35.0
29	brutus	1	33.0
58	rusty	10	35.0

rating	age
1	33.0
7	45.0
7	35.0
8	55.5
10	35.0

rating	
7	35.0

Exercises

- Sailors (sid, sname, rating, age)
- Boats (bid, bname, color)
- Reserves (sid, bid, day)

- For each red boat, find the number of reservations for that boat.

```
SELECT      B.bid,  COUNT (*) AS rescount
FROM        Boats B, Reserves R
WHERE       R.bid = B.bid AND B.color = 'red'
GROUP BY    B.bid
```

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

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

```
SELECT      Dno, COUNT (*), AVG (Salary)
FROM        EMPLOYEE
GROUP BY    Dno
```

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

- For each project, retrieve the project number, the project name, and the number of employees who work on it.

```
SELECT    Pnumber, Pname, COUNT (*)
FROM      PROJECT, WORKS_ON
WHERE     Pnumber = Pno
GROUP BY  Pnumber, Pname;
```

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

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

```
SELECT    Pnumber, Pname, COUNT (*)
FROM      PROJECT, WORKS_ON
WHERE     Pnumber = Pno
GROUP BY  Pnumber, Pname
HAVING    COUNT (*) > 2;
```

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

- For each project, retrieve the project number, project name, and the number of employees from department 5 who work on that project.

```
SELECT      Pnumber, Pname, COUNT (*)
FROM        PROJECT, WORKS_ON, EMPLOYEE
WHERE       Pnumber = Pno AND Ssn = Essn AND Dno = 5
GROUP BY    Pnumber, Pname;
```

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

- For each department that has more than 5 employees, retrieve the department number and number of employees who are earning more than $40,000.

```
SELECT      Dno, COUNT (*)
FROM        EMPLOYEE
WHERE       Salary>40000
GROUP BY    Dno
HAVING      COUNT (*) > 5
```

THIS QUERY IS INCORRECT!

WHY?

Exercises

- Employee(Ssn, ..., Salary, Super_Ssn, Dno)
- Department(Dnumber, Dname, Mgr_ssn, Mgr_start_date)
- Project(Pnumber, Pname, Plocation, Dnum)
- Works_On(Essn, Pno, Hours)

- For each department that has more than 5 employees, retrieve the department number and number of employees who are earning more than $40,000.

```
SELECT    Dno, COUNT (*)
FROM      EMPLOYEE
WHERE     Salary>40000 AND Dno IN
          ( SELECT     Dno
            FROM       EMPLOYEE
            GROUP BY   Dno
            HAVING     COUNT (*) > 5)
GROUP BY  Dno
```

Triggers and Views

Triggers

- Sometimes there exist **semantic constraints** that are beyond the scope of the relational model.
  - Salary shouldn't exceed $100,000.
  - A department cannot have > 1000 employees.
- **`CREATE TRIGGER:`** Specify automatic actions that the DBMS will perform when certain events and conditions occur. → **"Active databases"**
- A trigger typically consists of 3 parts:
  - Event(s)
  - Condition
  - Action

Triggers

- **Example:** When the salary of an employee exceeds the salary of his/her supervisor, inform the supervisor.
- INFORM_SUPERVISOR can be a sequence of SQL statements, a stored procedure, a transaction, or an external program that will be automatically executed.

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPERVISOR_SSN
    ON EMPLOYEE
FOR EACH ROW
    WHEN ( NEW.SALARY > ( SELECT SALARY FROM EMPLOYEE
                          WHERE SSN = NEW.SUPERVISOR_SSN ) )
    INFORM_SUPERVISOR(NEW.Supervisor_ssn, NEW.Ssn );
```

Views

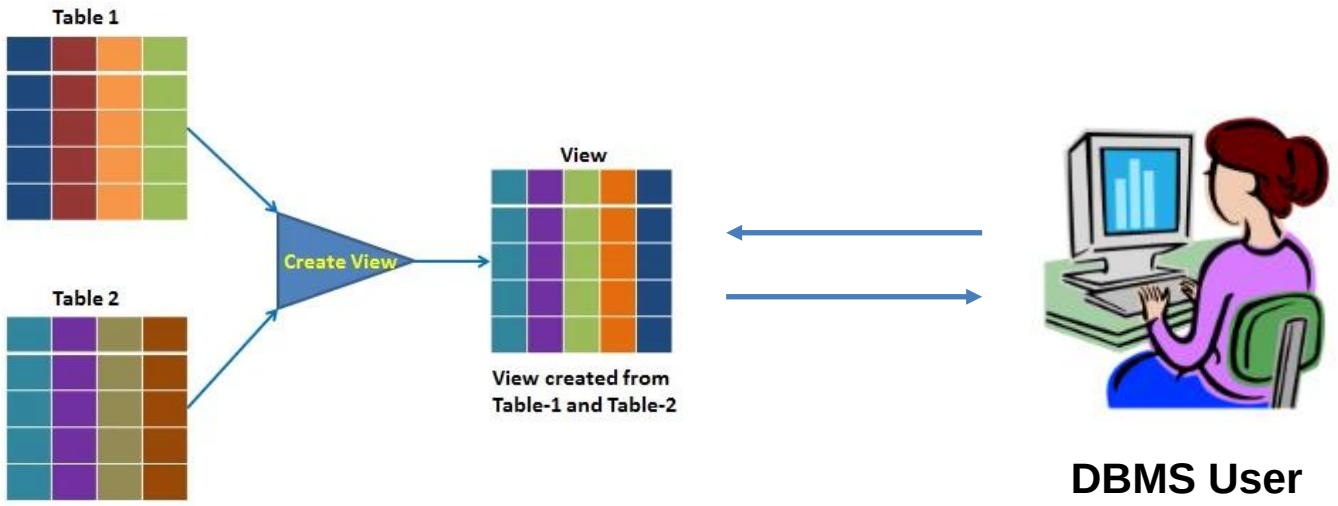
- Views are like "virtual tables" – they are tables derived from other tables ("real tables").
- Once a view is defined, SQL queries can use the view in their FROM clause.
- Two commands: **CREATE VIEW** – **DROP VIEW**

```
V1:   CREATE VIEW   WORKS_ON1
      AS SELECT     Fname, Lname, Pname, Hours
          FROM      EMPLOYEE, PROJECT, WORKS_ON
          WHERE     Ssn=Essn AND Pno=Pnumber;

V2:   CREATE VIEW   DEPT_INFO(Dept_name, No_of_emps, Total_sal)
      AS SELECT     Dname, COUNT (*), SUM (Salary)
          FROM      DEPARTMENT, EMPLOYEE
          WHERE     Dnumber=Dno
          GROUP BY  Dname;
```

Why Use Views?

- SQL retrieval queries can use views (-> simpler queries)
- Avoid repeating joins that are used very often
- Applications can use views; meanwhile, you can change the underlying tables without affecting the apps
- Privacy & security: Some users/apps can only read/query a view, which contains a limited portion of the whole DB



Updating Views

- As data in the underlying tables change, views must also be maintained and updated.
- Three main update strategies:
  - Immediate update – update the view as soon as the base tables have changed
  - Periodic update – update the view periodically, e.g., at the end of every day
  - Lazy update – update the view when it is necessary, e.g., right before a retrieval query uses a view
- These strategies have **pros and cons**
  - Do all of them guarantee correctness of query results?
  - Which one is fastest?

Exercises

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- List the names of managers with at least one dependent.

```
SELECT    EMPLOYEE.NAME
FROM      EMPLOYEE, DEPARTMENT
WHERE     DEPARTMENT.MGRSSN = EMPLOYEE.SSN   AND
              EMPLOYEE.SSN IN
                  (SELECT DISTINCT ESSN
                   FROM DEPENDENT)
```

Check "EXISTS", "NOT EXISTS" slide for a different solution

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- List the names of employees who do not work on any of the projects controlled by department number 5.

```
SELECT NAME
FROM EMPLOYEE
WHERE SSN NOT IN (SELECT WORKSON.ESSN
                  FROM WORKSON, PROJECT
                  WHERE WORKSON.PNO = PROJECT.PNUMBER
                        AND PROJECT.DNUM = 5)
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- List the names of employees who do not work on all of the projects controlled by department number 5.

```
SELECT E.NAME
FROM WORKSON W, EMPLOYEE E
WHERE E.SSN = W.ESSN
        AND EXISTS ((SELECT P.PNUMBER
                     FROM PROJECT P
                     WHERE P.DNUM = 5)
                EXCEPT (SELECT W1.PNO
                        FROM WORKSON W1
                        WHERE W1.ESSN = W.ESSN))
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- List the names of employees who do not have supervisors.

```
SELECT NAME
FROM EMPLOYEE
WHERE SUPERSSN IS NULL
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- Find the total, maximum, minimum, and average salary of employees working in the Research department.

```
SELECT SUM(SALARY) AS SALARY_SUM, MAX(SALARY) AS MAX_SALARY,
       MIN(SALARY) AS MIN_SALARY, AVG(SALARY) AS AVERAGE_SALARY
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DNO = DEPARTMENT.DNUMBER
            AND DEPARTMENT.DNAME = 'RESEARCH'
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- Find the number of employees in the Research department and print it to the screen with title "Employee_Count".

```
SELECT COUNT(*) AS EMPLOYEE_COUNT
FROM EMPLOYEE, DEPARTMENT
WHERE EMPLOYEE.DNO = DEPARTMENT.DNUMBER
        AND DEPARTMENT.DNAME = 'RESEARCH'
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- How many different employee salary values exist?

```
SELECT COUNT(DISTINCT SALARY)
FROM EMPLOYEE
```

Exercises

EMPLOYEE(NAME, SSN, BDATE, ADDRESS, SALARY, SUPERSSN, *DNO*)
DEPARTMENT(DNAME, DNUMBER, *MGRSSN*, MGRSTARTDATE)
DEPT_LOCATIONS(*DNUMBER*, DLOCATION)
PROJECT(PNAME, PNUMBER, PLOCATION, *DNUM*)
WORKSON(*ESSN*, *PNO*, HOURS)
DEPENDENT(*ESSN*, DEPENDENT_NAME, SEX, BDATE, RELATIONSHIP)

- Find the names of employees who have more than 5 children.

```
SELECT NAME
FROM EMPLOYEE
WHERE SSN IN (SELECT ESSN
                FROM DEPENDENT
                WHERE RELATIONSHIP = 'Child'
                GROUP BY ESSN
                HAVING COUNT(*) > 5)
```