# SQL: Advanced Queries

## CS 377: Database Systems

# Recap: SQL Queries

```
SELECT      [DISTINCT] <attribute list>
FROM         <table list>
[WHERE       <condition on the tables>]
[GROUP BY <grouping attributes>]
[HAVING      <group condition>]
[ORDER BY  <attribute list>   ASC | DESC]
[LIMIT       <number of tuples>]
```

# Today and Next Lecture

1. Temporal Relations
2. Explicit Join Operations
3. SQL Set Operations
4. Query Formulation Techniques
5. SQL View

# SQL Query: Temporal Relation

- Result of a **SELECT** clause that exists temporally, which assists you in formulating a query

- Syntax:

  **SELECT <attributes>**
  **FROM R1, R2, (SELECT … ) <alias>, …, RN**
  **WHERE <condition>;**

- Must always use an alias to denote the result relation of the **SELECT** command

# SQL Example: Temporal Relation

Find fname, lname of male employees with salary > 50K

```
SELECT *
FROM    (SELECT fname, lname, salary
         FROM    employee
         WHERE sex = ‘M’) r1
WHERE   r1.salary > 50000
```

# SQL Query: Temporal Relation Notes

- You can use multiple temporal relations
- You cannot use a temporal relation to create another temporal relation

Example of incorrect usage:

```
SELECT …
FROM    …,
            (SELECT …)  r1,
            (SELECT … FROM r1  …) r2,
WHERE …;
```

# SQL Query: WITH

- SQL-99 standard introduced **WITH** clause to help refine the result of a query (another way to achieve temporal relation)

- Syntax:
  **WITH <alias> AS (SELECT …)[,**
  **<alias2> AS (SELECT …)]**
  **SELECT <query>;**

- Can be used to perform “refinement” on a query

  - Subsequent queries in the **WITH** clause can use the results of the previous query

# SQL Example: WITH

Find all information on dependents of John Smith

```
WITH r1 as (SELECT *
            FROM employee
            WHERE fname = 'John'
                AND lname = 'Smith')
SELECT *
FROM   dependent
WHERE  essn IN (SELECT ssn from r1);
```

# SQL Query: WITH Notes

- Some vendors do not support **WITH** (e.g., MySQL)
- Options for dealing
  - TEMPORAL relations
  - TEMPORARY tables
  - VIEW (more on this later)

# SQL Query: JOIN Operations

- SQL-99 standard added several join operations:
  - **INNER JOIN** (normal join)
  - **LEFT JOIN** (left outer join)
  - **RIGHT JOIN** (right outer join)
  - **FULL JOIN** (outer join)

# SQL Query: JOIN Operations

- Each operation results in a relation
- Operation can only appear in:
  - **FROM** clause of **SELECT** command
  - **WHERE** clause of **SELECT** command with an operator that uses a sub-query

# SQL Query: [INNER] JOIN

- Compute the (inner) join between tables r1 and r2 with a given join condition

- Syntax:
  **r1 JOIN r2 ON <join-condition>;**
  or
  **r1 INNER JOIN r2 ON <join-condition>;**

- **JOIN** operator makes the SQL query look a lot like RA query

- Can join more than 2 relations

# SQL Example: INNER JOIN

Find fname, lname of employees in the ‘Research’ department

RA Query:

$$\pi_{\text{fname,lname}}(\sigma_{\text{dname='Research'}}(\text{EMPLOYEE} \bowtie_{\text{dno=dnumber}} \text{DEPARTMENT}))$$

SQL Query:

```
SELECT fname, lname
FROM   (employee JOIN department
          ON dno = dnumber)
WHERE  dname = ‘Research’;
```

# SQL Query: OUTER JOIN

- Compute the outer join between tables r1 and r2 with a given join condition - see RA slides for details on difference between left, right, and full outer joins

- Syntax:
  **r1 LEFT | RIGHT| FULL [OUTER] JOIN r2 on <join condition>;**

- Results in NULL values for the attributes where non-matching tuples occur

# SQL Query: NATURAL JOIN

- Compute the natural join on attributes with the same names from two or more tables with the common attribute appearing only once in the result

- Syntax:
  **r1 NATURAL JOIN r2;**

- Example:
  **SELECT ***
  **FROM works_on NATURAL JOIN dependent;**

# SQL Query: CROSS JOIN

- Cross join is the same as a Cartesian Product
- Syntax:
  **r1 CROSS JOIN r2;**
- Example:
  **SELECT ssn, fname, lname, dno, dnumber, dname**
  **FROM employee CROSS JOIN dependent;**

# SQL: Implicit vs Explicit JOIN

- Why the difference?

- SELECT *
  FROM table1 a
  INNER JOIN table2 b
  ON a.id = b.id;

  vs

  SELECT *
  FROM table1 a, table2 b
  WHERE a.id = b.id;

- Explicit (inner join) vs implicit join

- In some systems, explicit queries are better optimized for large records

# SQL: Set Operations

Not all set operations have been incorporated into SQL

- **UNION**: in most implementations because it's very easy to merge 2 result sets (O(n) running time)
- **INTERSECT**: in few implementations because it's hard to intersect 2 sets (O(N log N) running time)
- **MINUS**: almost no implement provides this (just as expensive as **INTERSECT**)
- **CARTESIAN PRODUCT**: built into SELECT command

# SQL: Set Operations (2)

- Resulting relations of set operations are sets of tuples
  —> duplicate tuples are eliminated from the result

- Set operations apply only to union compatible relations:
  two relations must have the same attributes and
  attributes must be in the same order

- Set division is not part of the SQL standard

- MySQL only implements the UNION operator

# Example Query: UNION

Find the name of projects that are worked on by ‘Smith’ or ‘Borg’

```
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno
       AND essn = ssn
       AND lname = ‘Smith’)
UNION
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno
       AND essn = ssn
       AND lname = ‘Borg’)
```

# Example Query: UNION (2)

List all project names that involve an employee whose last name is ‘Smith” either as a worker or manager of the department that controls the project

```
(SELECT pname
 FROM    project, department, employee
 WHERE dnum = dnumber AND mgrssn = ssn
      AND lname = ‘Smith’)
UNION
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno AND essn = ssn
    AND lname = ‘Smith’)
```

# SQL: Beyond Union

- Can I make queries that use intersection, difference, or division?

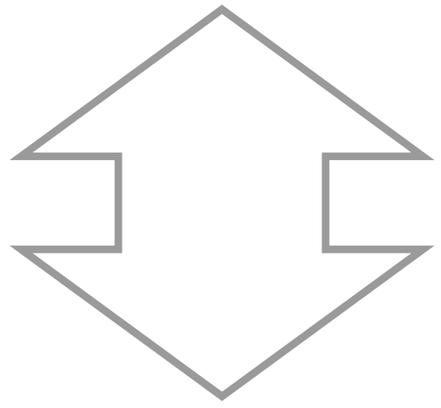
- What are techniques for answering such queries?

Ans: There are query formulation techniques (QFT) that can be followed to address these deficiencies!

# QFT: INTERSECT

How to compute the intersection of two sets when the system does not support INTERSECT (e.g., MySQL)?

**x IN (set1 INTERSECT set2)**



**(x IN set1) AND (x IN set2)**

# SQL Example: INTERSECT

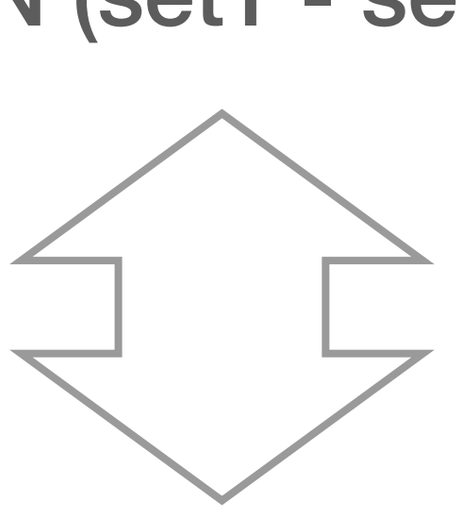
Find fname and lname of employees who work on some project controlled by the ‘Research’ department and also on some project controlled by the ‘Administration’ department

```
SELECT  fname, lname
FROM    employee
WHERE  ssn IN ( SELECT essn
                FROM    works_on, project, department
                WHERE  pno = pnumber
                    AND dnum = dnumber AND dname = ‘Research’)
   AND   ssn IN  ( SELECT essn
                   FROM    works_on, project, department
                   WHERE  pno = pnumber AND dnum = dnumber
                     AND dname = ‘Administration’);
```

# QFT: DIFFERENCE

How to compute the difference of two sets when SQL doesn't support set difference?

**x IN (set1 - set2)**



**(x IN set1) AND (x NOT IN set2)**

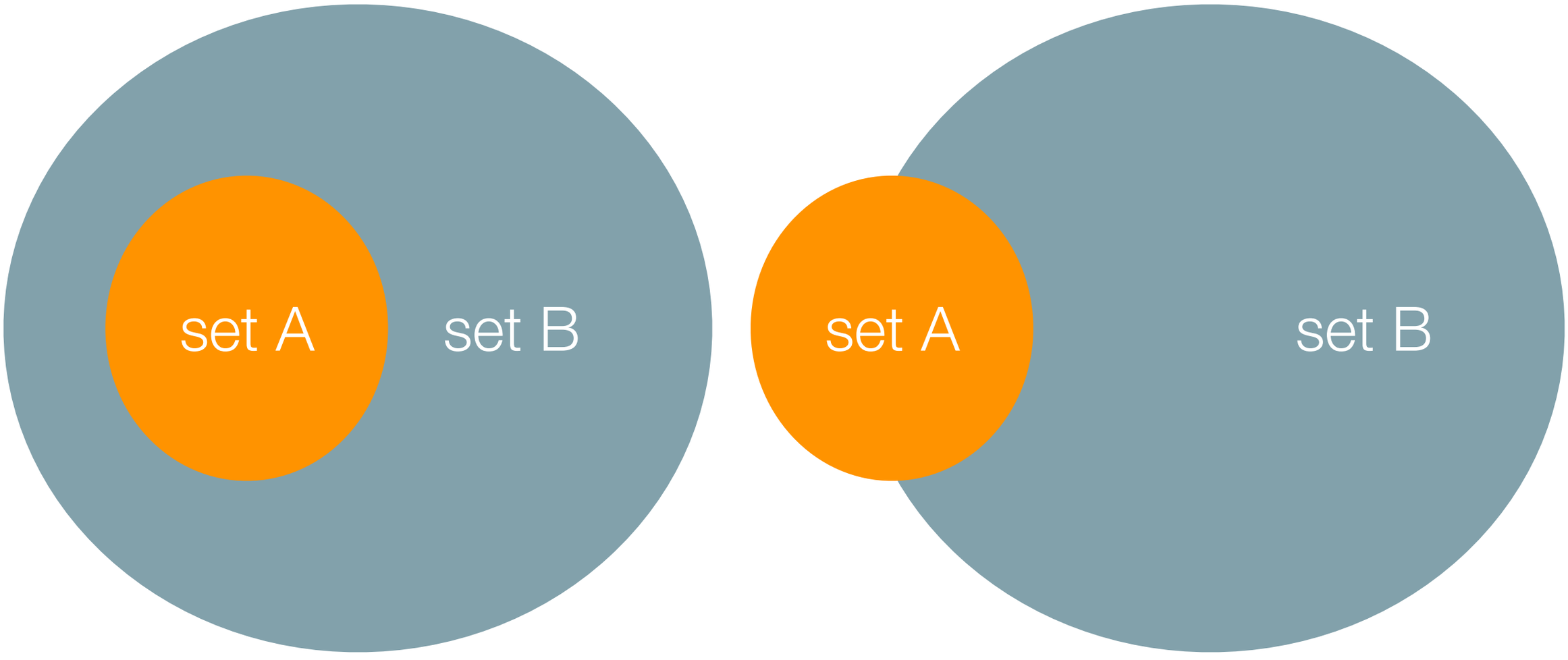
# SQL Example: DIFFERENCE

Find SSN of employees in the ‘Research’ department who has no dependents

```
SELECT  ssn
FROM    employee
WHERE   ssn IN ( SELECT ssn
                 FROM   employee, department
                 WHERE dno = dnumber
                   AND dname = ‘Research’)
   AND   ssn NOT IN ( SELECT essn
                      FROM dependent)
```

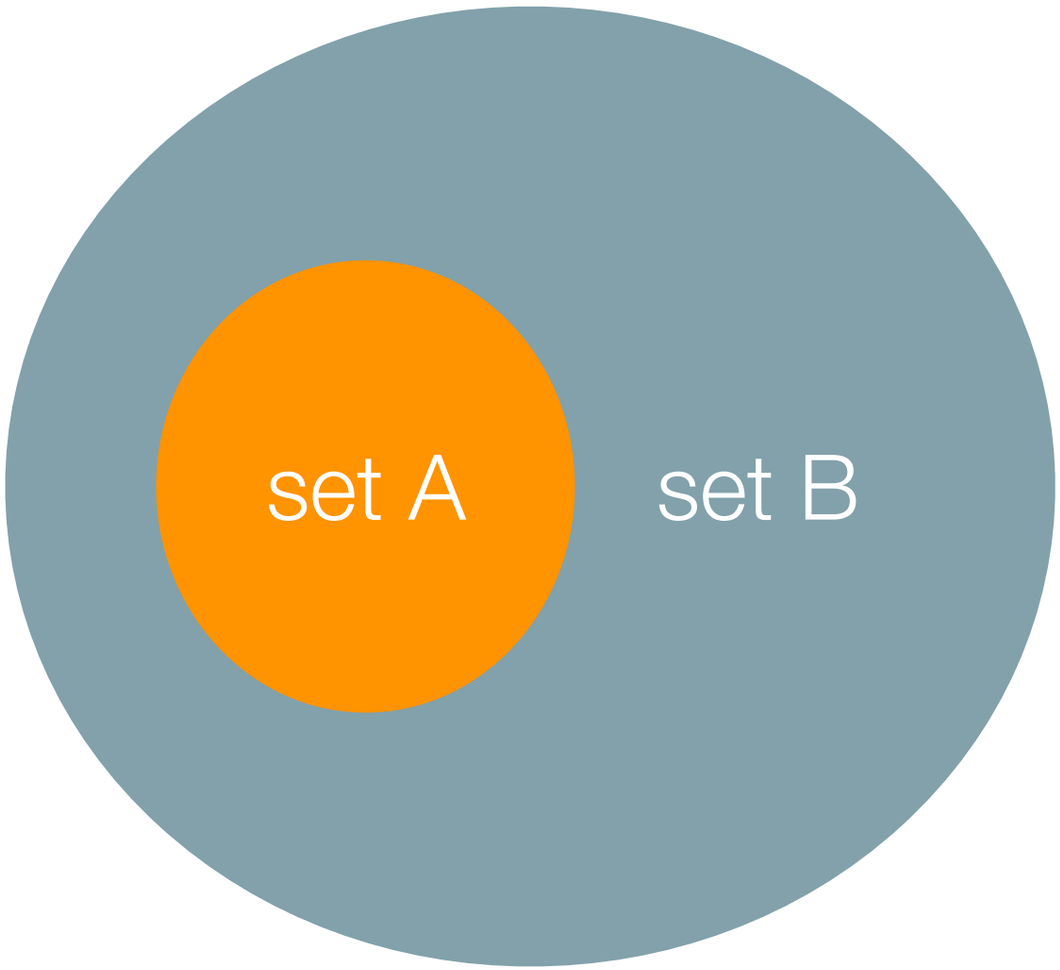
# QFT: Superset

How to formulate set B is a super set of set A?



# QFT: Superset

How to formulate set B is a super set of set A?




If superset, set B - set A is empty set! Otherwise, set B does not contain all of set A.

# QFT: Superset

How to formulate set1 is a superset (contains) of another set, set2?

**set1 CONTAINS set2 <=> set2 - set1 = EMPTY**

```
SELECT …
FROM …
WHERE NOT EXISTS (SELECT *
                  FROM    <table>
                  WHERE x IN set2
                    AND  x NOT IN set1)
```

# QFT: Subset

How to formulate set1 is a subset (part of) of another set, set2?

**set1 SUBSET set2 <=> set1 - set2 = EMPTY**

```
SELECT …
FROM …
WHERE NOT EXISTS (SELECT *
                  FROM    <table>
                  WHERE x IN set1
                     AND  x NOT IN set2)
```

# Subset vs Superset

- Syntax is almost the same, only nested query is different
- Relations specified for IN and NOT IN are the differentiators for subset vs superset query — easy to get them mixed up

```
(SELECT *
FROM    <table>
WHERE x IN set2
AND  x NOT IN set1)
```

set1 superset of set2

```
(SELECT *
FROM    <table>
WHERE x IN set1
AND  x NOT IN set2)
```

set1 subset of set2

# QFT: Division

- How to compute the division between two relations?
- Example: Find lname of all employees who work on all projects controlled by department number 4
  - RA:

$$\text{H1} = \pi_{\text{pnumber}}(\text{PROJECT} \bowtie_{\text{dnum=dnumber}} \sigma_{\text{dname='Research'}}(\text{DEPARTMENT}))$$
$$\text{H2} = \pi_{\text{essn,pno}}(\text{WORKS\_ON})$$
$$\text{H3} = \text{H2} \div \text{H1}$$
$$\text{Answer} = \pi_{\text{fname,lname}}(\text{EMPLOYEE} \bowtie_{\text{ssn = ssn}} \text{H3})$$

# QFT: Division

- How to compute the division between two relations?
- Example: Find lname of all employees who work on all projects controlled by department number 4
- SQL: Use **NOT EXISTS** and set difference
- Use superset idea — set of projects worked on by an employee contains set of projects controlled by department 4

# SQL Example: DIVISION

Find lname of all employees who work on all projects controlled by department number 4

```
SELECT  fname, lname
FROM    employee
WHERE   NOT EXISTS
( SELECT pnumber
  FROM   project
  WHERE  pnumber IN (SELECT pnumber
                     FROM   project
                     WHERE  dnum = 4)
    AND  pnumber NOT IN (SELECT pno
                         FROM   works_on
                         WHERE essn = ssn));
```

project controlled by Research

projects worked on by employee

# QFT: Only

- How to compute queries that ask for only?
- Example: Find the names of projects that are worked on by only employees in the ‘Research’ department?
- Formulate the solution using a subset condition:
  - Employees working on project p are a subset of employees in the Research department

# SQL Example: Only

Find the names of projects that are worked on by only employees in the 'Research' department?

```
SELECT  pname
FROM    project
WHERE   NOT EXISTS
    ( SELECT ssn
      FROM    employee
      WHERE  ssn IN (SELECT essn
                     FROM    works_on
                     WHERE  pno = pnumber)
        AND    ssn NOT IN (SELECT ssn
                           FROM   employee, department
                           WHERE dno = dnumber
                             AND dname = 'Research'));
```

employees working on project p

employees from research department

# QFT: Most Number of

- How to compute queries that ask for the most number of some attribute?

- Example: Find the name of the departments with most number of employees?

Ans: Use nested query with the max function

```
SELECT      dname
FROM        department, employee
WHERE       dno = dnumber
GROUP BY dname
HAVING      COUNT(ssn) = (SELECT MAX(COUNT(ssn))
                          FROM    employee GROUP BY dno);
```

# SQL Practice (1)

- Find the name of the departments with 2 or more male employees

- Find the name of the employees with the most number of dependents

- Find fname and lname of employees who works on all projects that are worked on by John Smith

# SQL Practice (2)

- Find the department name, and the number of employees in that department that earns more than 40K for departments with at least 2 employees

- Find fname, lname of employees who work on 2 or more projects together with John Smith

- Find departments who have 2 or more employees working on all projects controlled by ‘Research’ department

# SQL Practice (3)

- Find the project name and the number of employees working on that project; for projects that has 3 or more employees working on the project

- Find the fname and lname of the employees with more than 2 dependents and work on more than 2 projects

- Find the fname and lname of the employees with more than 2 dependents and work on all projects controlled by department #1

# SQL: View

# SQL: VIEW

- A view is a virtual table, a relation that is defined in terms of the contents of other tables and views

- A view d*oes not* exist in the physical form

- In contrast, a relation whose value is really in the database is called a base table

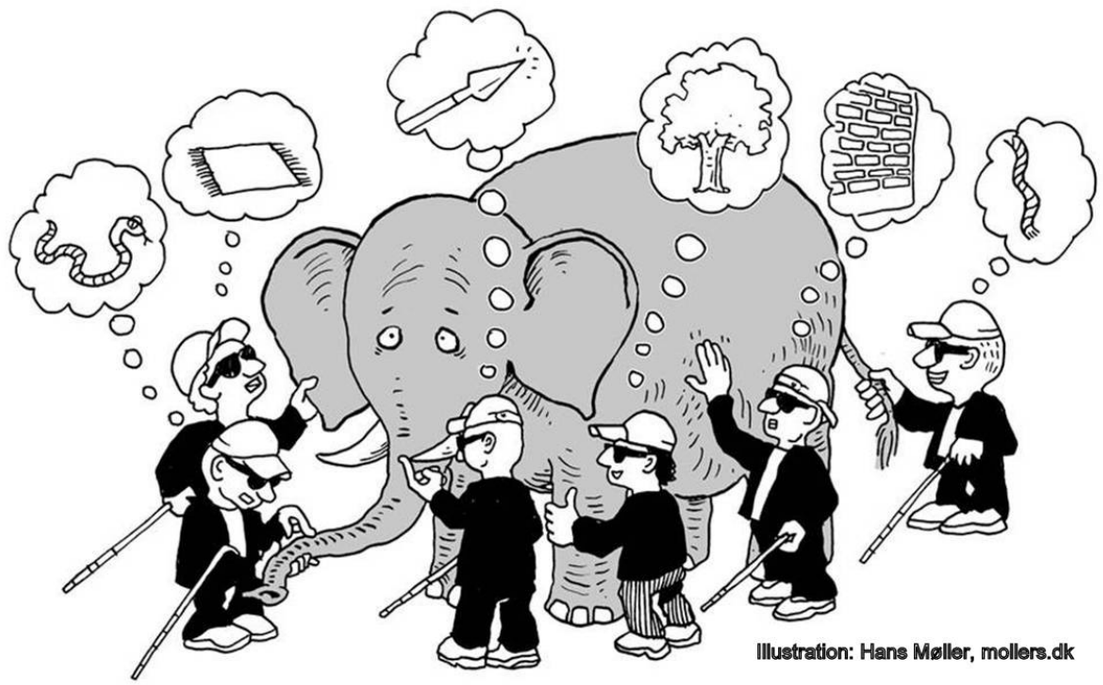
- Syntax:
  **CREATE VIEW <name> AS <query>;**

# SQL: View & Logical Data Independence

- Recall Logical Data Independence (class on Database Concepts)
    - Ability to present the stored information in a different way to different users

- View can be adapted to the need of the user

- If conceptual schema changes, only the **SELECT** query needed to construct view needs to change



# SQL Example: VIEW

- Suppose an administrator maintains a list of activities of all employees which contains the following information:
  fname, lname, project_name, hours_worked

- Regular SELECT query:

```
SELECT fname, lname, pname, hours
FROM   employee, works_on, project
WHERE  ssn = essn AND pno = pnumber;
```

- Create VIEW for the admin:

```
CREATE VIEW emp_activity
AS (SELECT fname, lname, pname, hours
      FROM   employee, works_on, project
      WHERE  ssn = essn AND pno = pnumber);
```

# SQL: VIEW Advantages

- View can be used in queries like an ordinary relation
  - When a view is used in a **SELECT** query, the virtual relation is computed first

- Simplify complex queries by hiding them from the end-user and applications

- Limit data access to specific users (expose only non-sensitive data) and provides extra security for read/write access

- Enables backward compatibility - changes to database won't affect changes to other applications

# SQL: VIEW Disadvantages

- Querying data from database view can be slow (since view is computed each time)

- Tables dependency - updates to the underlying tables will force changes to the view itself to make it work properly

- Most data manipulation statements (**INSERT**, **DELETE**, **UPDATE**) are not possible on the view

# SQL Advanced Queries: Recap

- Temporal relations
- Explicit join operations
- Formulating set operations
- SQL view operation