SQL – Data Manipulation Language
Part 2

Lecturer: Neena Thota

neena.thota@it.uu.se

Intended Learning Outcomes

- Write and understand:
  - Nested queries.
  - Queries based on views.
  - Outer and Inner Joins.

Review

select target_list

from table_list

[where tuple_predicates]

[group by attribute_list]

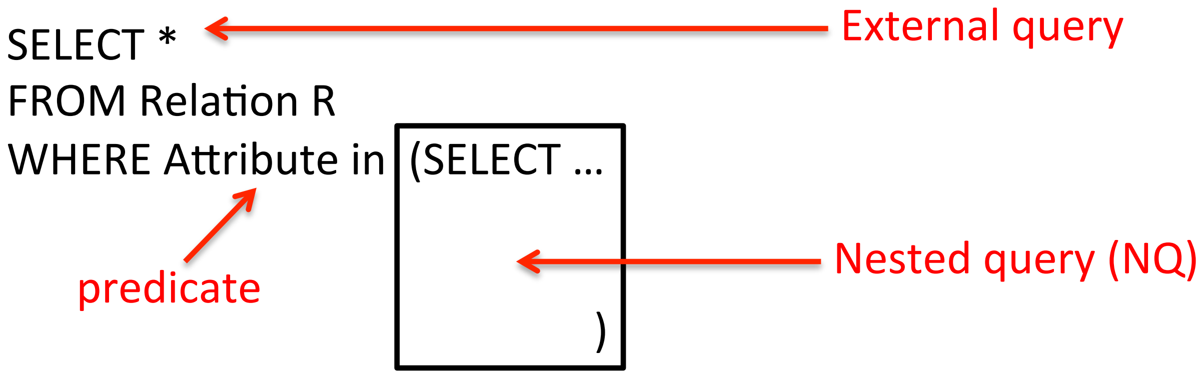
[having group_predicates]

[order by attribute_list + ASC/DESC]

UPPSALA
UNIVERSITET

NESTED QUERIES

Nested Queries



```
SELECT *
FROM Relation R
WHERE Attribute in (SELECT …

                               )
```

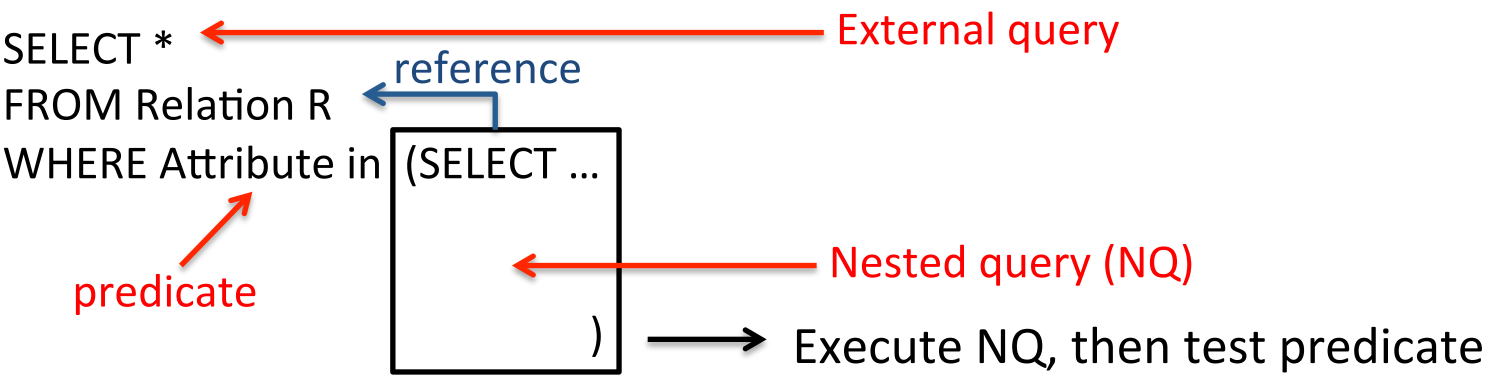
External query

predicate

Nested query (NQ)

- Require that existing values in database be fetched and then used in a comparison condition.
- **E.g**. Make a list of all project numbers for projects that involve an employee whose last name is ‘Smith’, either as a worker or as a manager of the department that controls the project.

General interpretation of Nested Queries



SELECT *
FROM Relation R
WHERE Attribute in (SELECT …

)

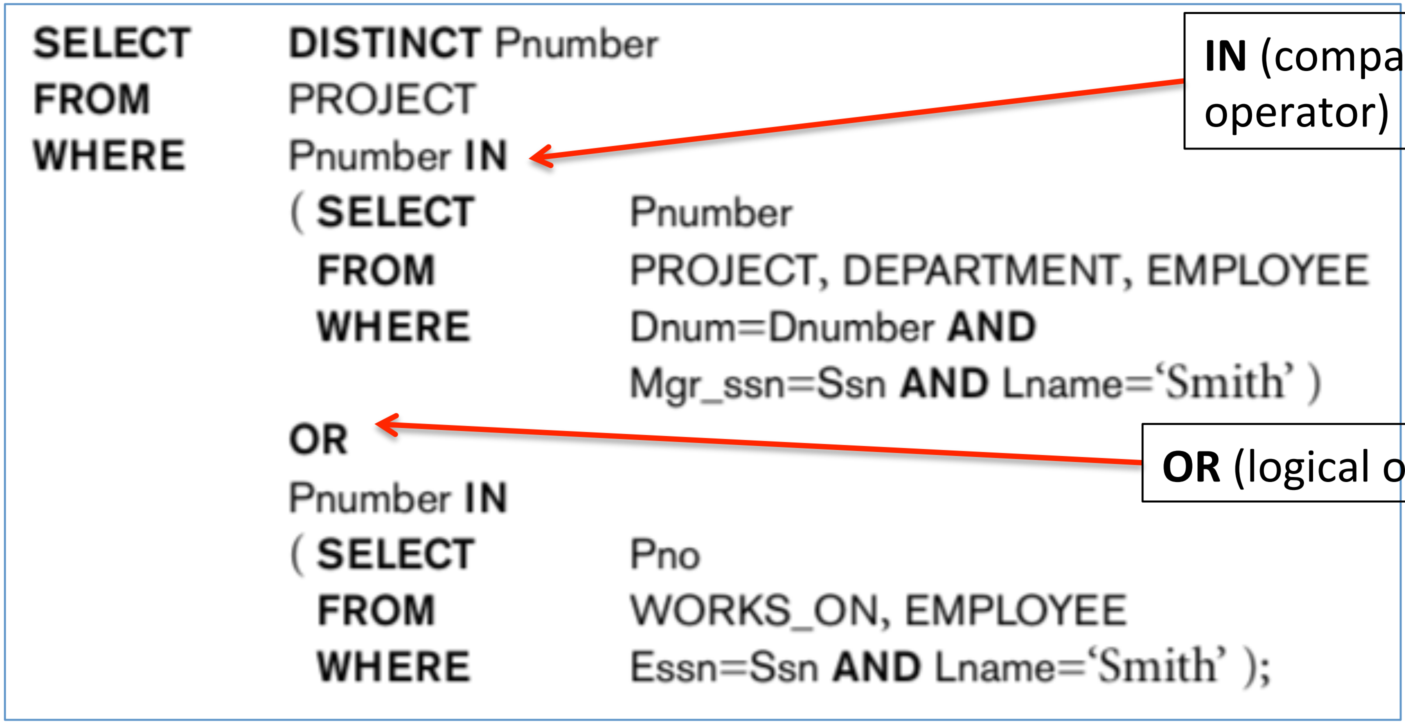
External query

reference

predicate

Nested query (NQ)

Execute NQ, then test predicate



```
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 (comparison operator)

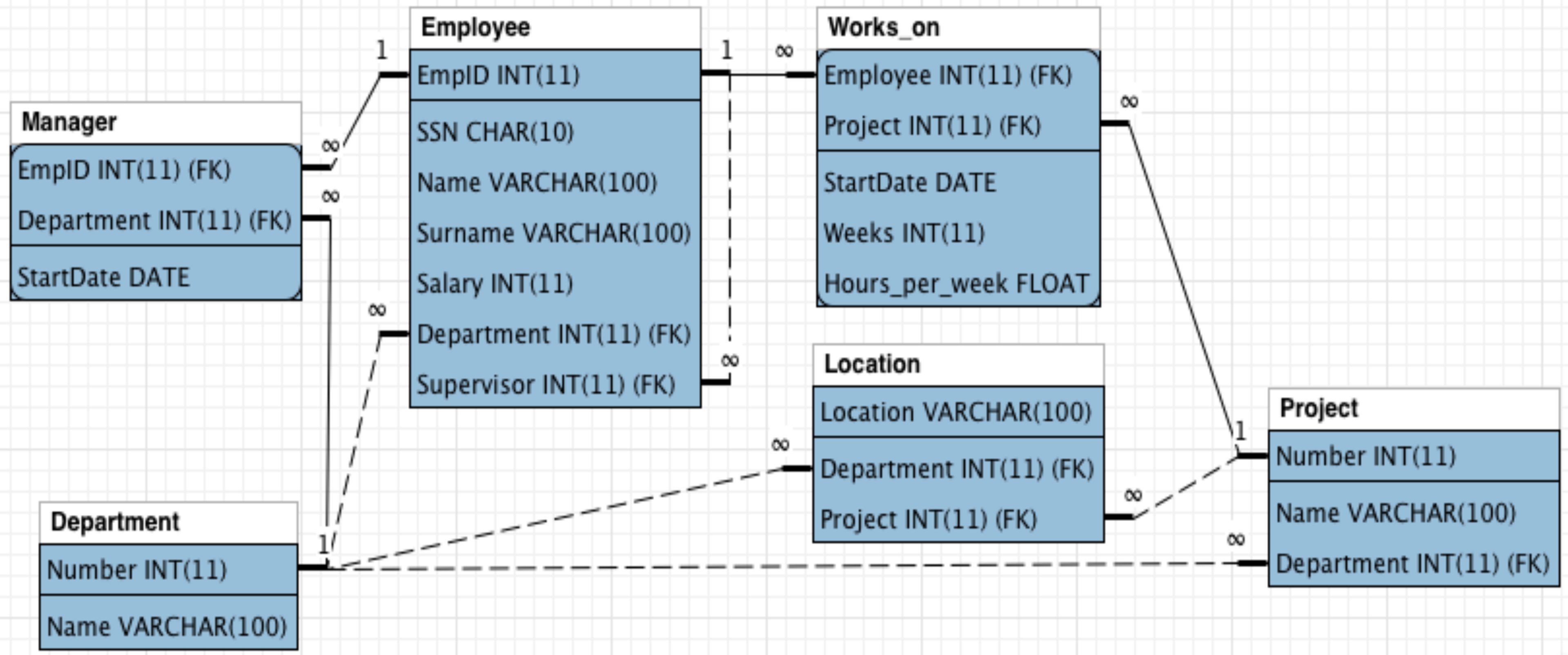
OR (logical operator)

Operators for Nested Queries

- When the internal (nested) query returns more than one tuple, we need to better specify the external predicate.
- We can use normal comparison operators like **<, >, >=, =**, etc. followed by the operators **ANY** and **ALL**. For example:
  - = ANY (can also be written: **IN**)
  - <> ALL (can also be written: **NOT IN**)
- ***EXISTS*** checks if the result of an internal query is empty.
  - The opposite is: ***NOT EXISTS***.

EXAMPLES

1. Select all the employees whose supervisor is a manager (using the *=ANY* or *IN* operators).
2. Select all the departments where there are no employees called "Doo" (using the *<>ALL* or *NOT IN* operators).
3. Select all the employees having the maximum salary with respect to their department (using the *NOT EXISTS* operator).



E.g.1. Select all the employees whose supervisor is a manager (using ANY or IN).

```
select *
from Employee
where Supervisor = ANY (Select EmpID
                        from Manager)
-- or
select *
from Employee
where Supervisor IN (Select EmpID
                     from Manager)
```

E.g.2. Select all the departments where there are no employees called “Doo” (using the <>*ALL* or *NOT IN)*.

```
select *
from Department
where Number <>ALL (select Department
                    from Employee
                    where Surname = ‘Doo’)
-- or
select *
from Department
where Number NOT IN (select Department
                     from Employee
                     where Surname = ‘Doo’)
```

E.g.3. Select all the employees having the maximum salary with respect to their department (using *NOT EXISTS*)

```
select *
from Employee E
where NOT EXISTS (select *
                  from Employee
                  where Department = E.Department
                  and Salary > E.Salary)
```

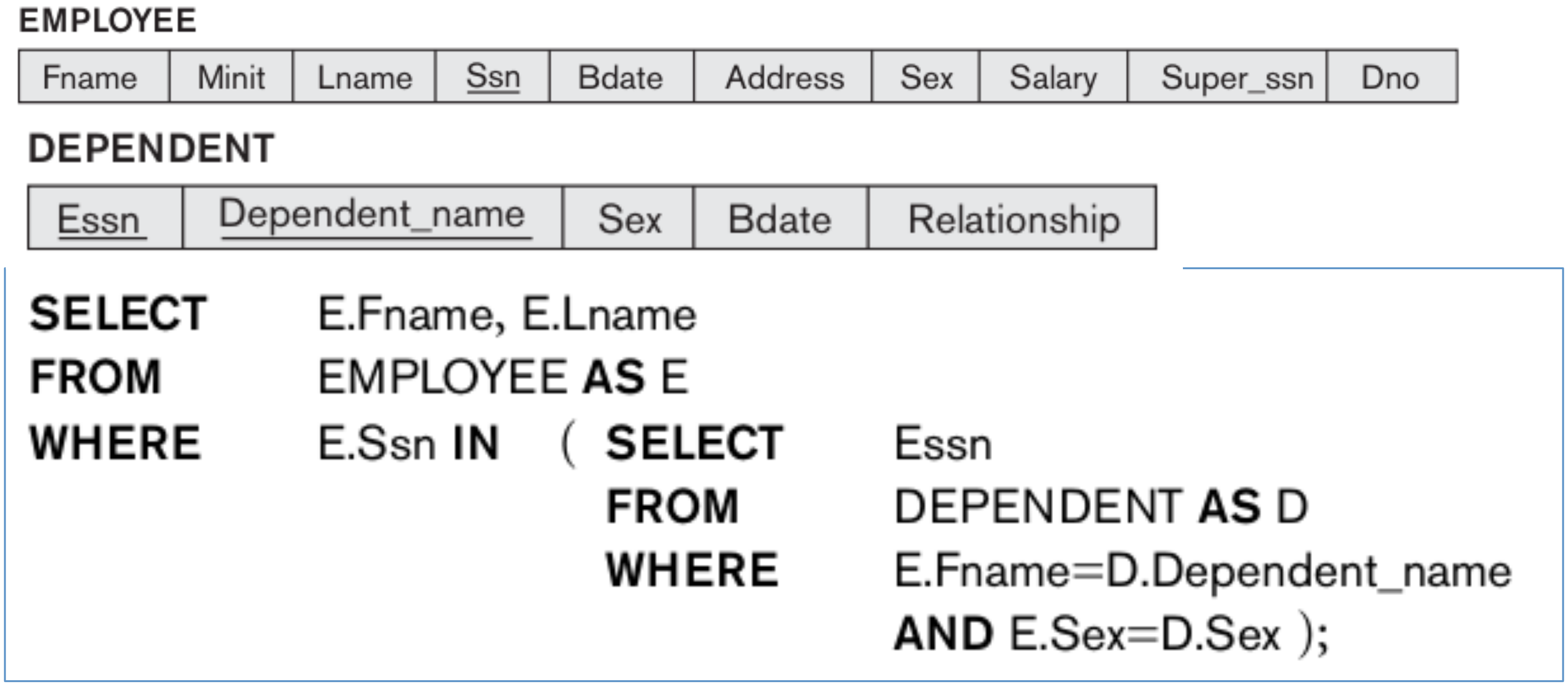
TO COMPARE SINGLE ATTRIBUTE

Summary

- = ANY (or = SOME) operator returns TRUE if value *v* is equal to *SOME value* in set *V* and is hence equivalent to IN.
- Comparison condition (*v* > ALL *V*) returns TRUE if value *v* is greater than *ALL* values in set (or multiset) *V*.
- Result of EXISTS is Boolean value TRUE if nested query result contains at least 1 tuple, or FALSE if nested query result contains no tuples.

Visibility

- Each relation/alias is visible only inside internal subqueries.
- Rule: reference to an *unqualified attribute* refers to the relation declared in the **innermost nested query**.
- Advisable to create tuple variables (aliases) for *all the tables referenced in an SQL query*



EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship

```
SELECT      E.Fname, E.Lname
FROM        EMPLOYEE AS E
WHERE       E.Ssn IN  ( SELECT      Essn
                        FROM        DEPENDENT AS D
                        WHERE       E.Fname=D.Dependent_name
                                    AND E.Sex=D.Sex );
```

VIEWS

Views (Virtual Tables)

- **View** – a single (virtual) table derived from other tables; can be from **base tables (physical form)** or other **previously defined views.**
- To create a view:

  CREATE VIEW *viewname* **AS** <query expression>

```
CREATE VIEW RichEmployees AS
select *
from Employee
where Salary > 60000
```

```
CREATE VIEW ProjectWork(Emp, Project) AS
select E.Surname, P.Name
from Employee E, Works_on W, Project P
where E.EmpID=W.Employee AND P.Number=W.Project
```

Specifies new attribute names

Update of a view

DBMS

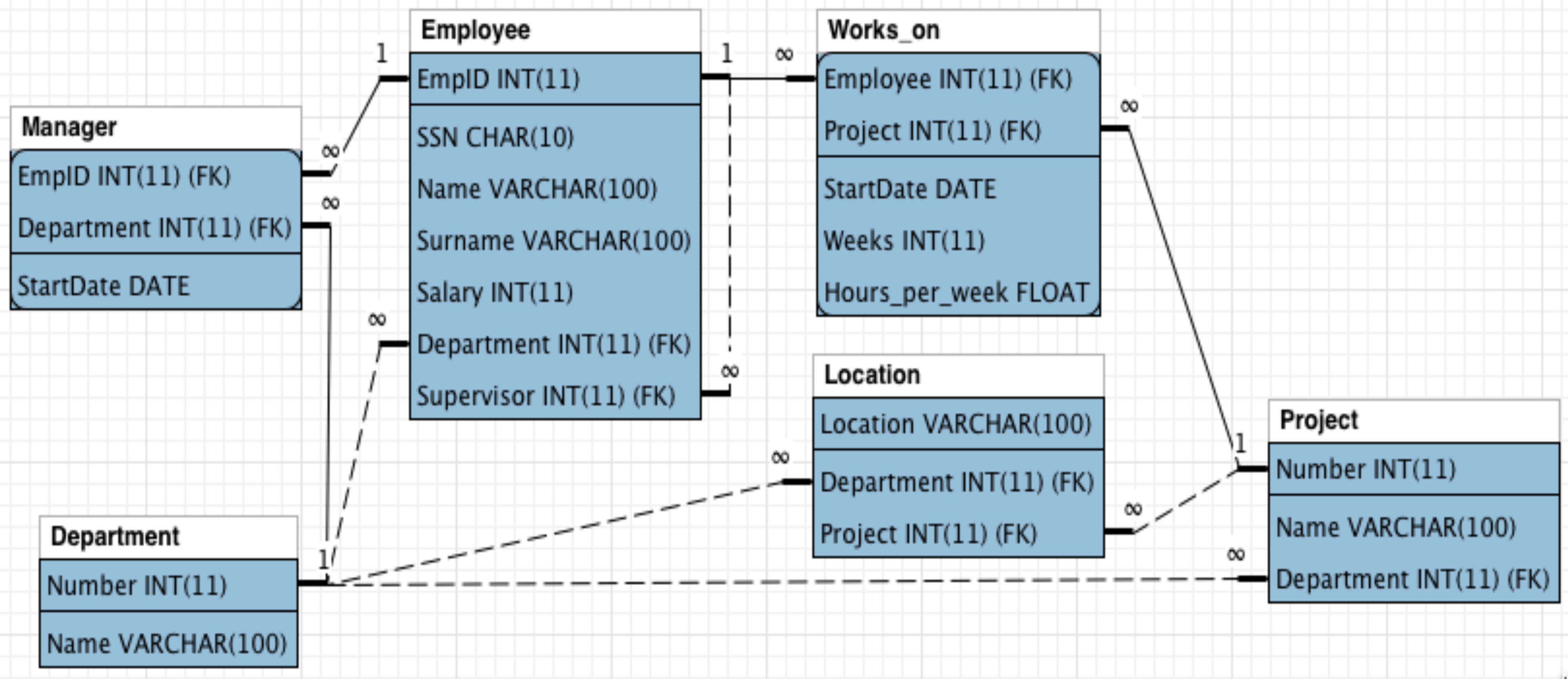
- – Can modify or transform view query (submitted by user) into query on underlying base tables – **Query Modification**.
- – Physically creates temporary view table when view is first queried and keeps table on assumption that other queries on view will follow – **View Materialization**.

- View/s
  - – with a single definition table is **updatable** if its attributes **contain the PK**.
  - – on multiple tables **using joins** are **not updatable** in general.
  - – using **GROUP BY and aggregations** are **not updatable**.
- Advanced options can be specified using SQL for security and authorization (not treated here).
- We can use the **DROP VIEW** command to dispose of a view.

Exercise

- Express the following query defining a view (or more than one, if you prefer), and writing a query that uses the view as a base relation.

Select the department(s) with the highest average salary.



Employee
EmpID INT(11)
SSN CHAR(10)
Name VARCHAR(100)
Surname VARCHAR(100)
Salary INT(11)
Department INT(11) (FK)
Supervisor INT(11) (FK)

Works_on
Employee INT(11) (FK)
Project INT(11) (FK)
StartDate DATE
Weeks INT(11)
Hours_per_week FLOAT

Manager
EmpID INT(11) (FK)
Department INT(11) (FK)
StartDate DATE

Location
Location VARCHAR(100)
Department INT(11) (FK)
Project INT(11) (FK)

Project
Number INT(11)
Name VARCHAR(100)
Department INT(11) (FK)

Department
Number INT(11)
Name VARCHAR(100)

Solution, previous slide

Select the department(s) with the highest average salary.

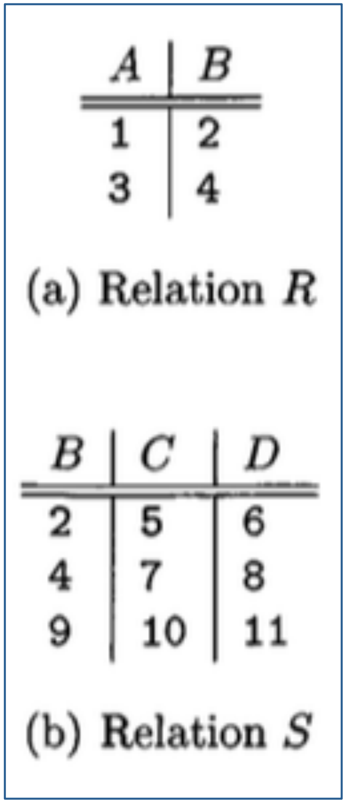
```
CREATE VIEW dept_avg_salary AS
SELECT Department, AVG(Salary) as avg_salary
FROM Employee
GROUP BY Department;

SELECT Department
FROM dept_avg_salary
WHERE avg_salary >= ALL (Select avg_salary
from dept_avg_salary);
```

JOINS

Some Types of Joins

- **NATURAL JOIN**
  - no join condition is specified; implicit *condition* for *each pair of attributes with same name* from *both tables* is created.
  - Each such pair of attributes included *only once* in resulting relation
- **INNER JOIN**
  - only pairs of tuples that **match join condition** are retrieved,
- **OUTER JOIN**
  - tuples from two tables **combined** by matching corresponding rows without losing any tuples for lack of matching values.
- **LEFT OUTER JOIN**
  - every tuple in the **left** table must appear in the result;
  - If no matching tuple, padded with NULL values for attributes of right table.
- **RIGHT OUTER JOIN**
  - Every tuple in the **right** table must appear in the result;
  - If no matching tuple, padded with NULL values for attributes of left table.
- **FULL OUTER JOIN**
  - Keeps all tuples in both **left and right relations** when no matching tuples are found, padding with NULL values as needed.

EXAMPLES

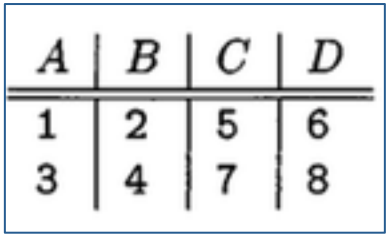


A	B
1	2
3	4

(a) Relation *R*

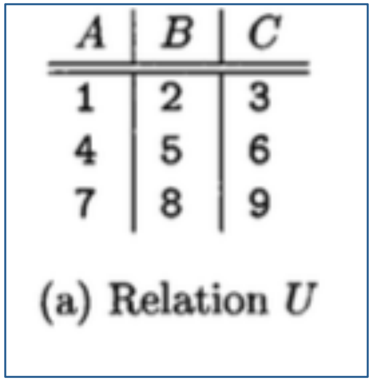
B	C	D
2	5	6
4	7	8
9	10	11

(b) Relation *S*



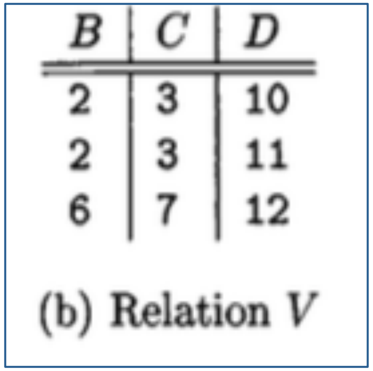
A	B	C	D
1	2	5	6
3	4	7	8

NATURAL JOIN



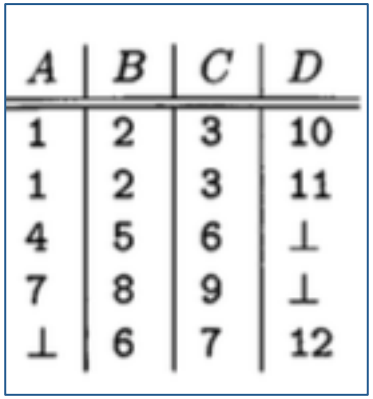
A	B	C
1	2	3
4	5	6
7	8	9

(a) Relation *U*



B	C	D
2	3	10
2	3	11
6	7	12

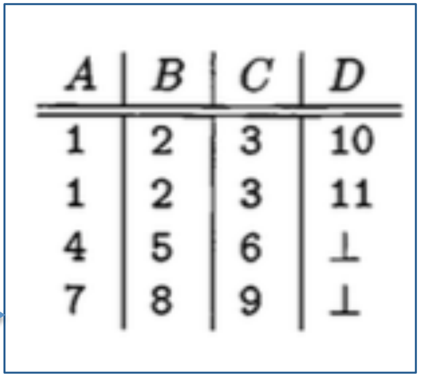
(b) Relation *V*



A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	⊥
7	8	9	⊥
⊥	6	7	12

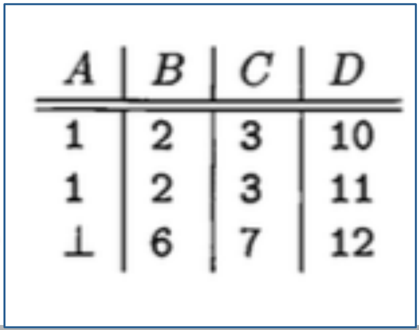
OUTER JOIN

LEFT JOIN



A	B	C	D
1	2	3	10
1	2	3	11
4	5	6	⊥
7	8	9	⊥

RIGHT JOIN



A	B	C	D
1	2	3	10
1	2	3	11
⊥	6	7	12

EXAMPLE

```
select EmpID, Name
from Manager right outer join Department
on Department = Number
```

- Using a simple join between managers and departments, a department without a manager is not selected.
- We can extend the result by including those departments that do not match any manager.
  - – (in general: with those tuples not matching any tuple in the other relation).
- The rest of the tuple in the result is filled with *nulls*.

Example of right outer join

```
select EmpID, Name
from Manager right outer join Department
on Department = Number
```

MANAGER

EmpID	***Department***
1	45
2	48

DEPARTMENT

Number	***Name***
45	IT
48	Phyl
51	Math

Result:

EmpID	***Name***
1	IT
2	Phyl
NULL	Math

Similarly, we can use a **left outer join** or a **full outer join**.

Wrap up: SQL as a DML

- Simple SQL queries.
- Extensions of SELECT-FROM-WHERE.
  - LIKE, BETWEEN, IS (NOT) NULL.
  - DISTINCT.
  - ORDER BY.
- Set operators
  - UNION, (EXCEPT, INTERSECT)
- Aggregation
  - GROUP BY / HAVING.
- Nested queries
  - ANY, ALL, (NOT) IN, (NOT) EXISTS.
- Views.
- Outer and Inner joins.