Lecture 11

Relational Algebra
(cont'd)

Week 5

Overview

- Join
  - join
  - equijoin and natural join
- Division
- Simplifying relational algebra expressions

Join

$$A \bowtie_C B$$

Not strictly necessary, but very useful

$$A \bowtie_C B \equiv \sigma_C (A \times B)$$

I.e. take the cartesian product (cross product) of A and B and select tuples which satisfy condition C

Equijoin

$$A \bowtie_{\text{equality conditions}} B$$

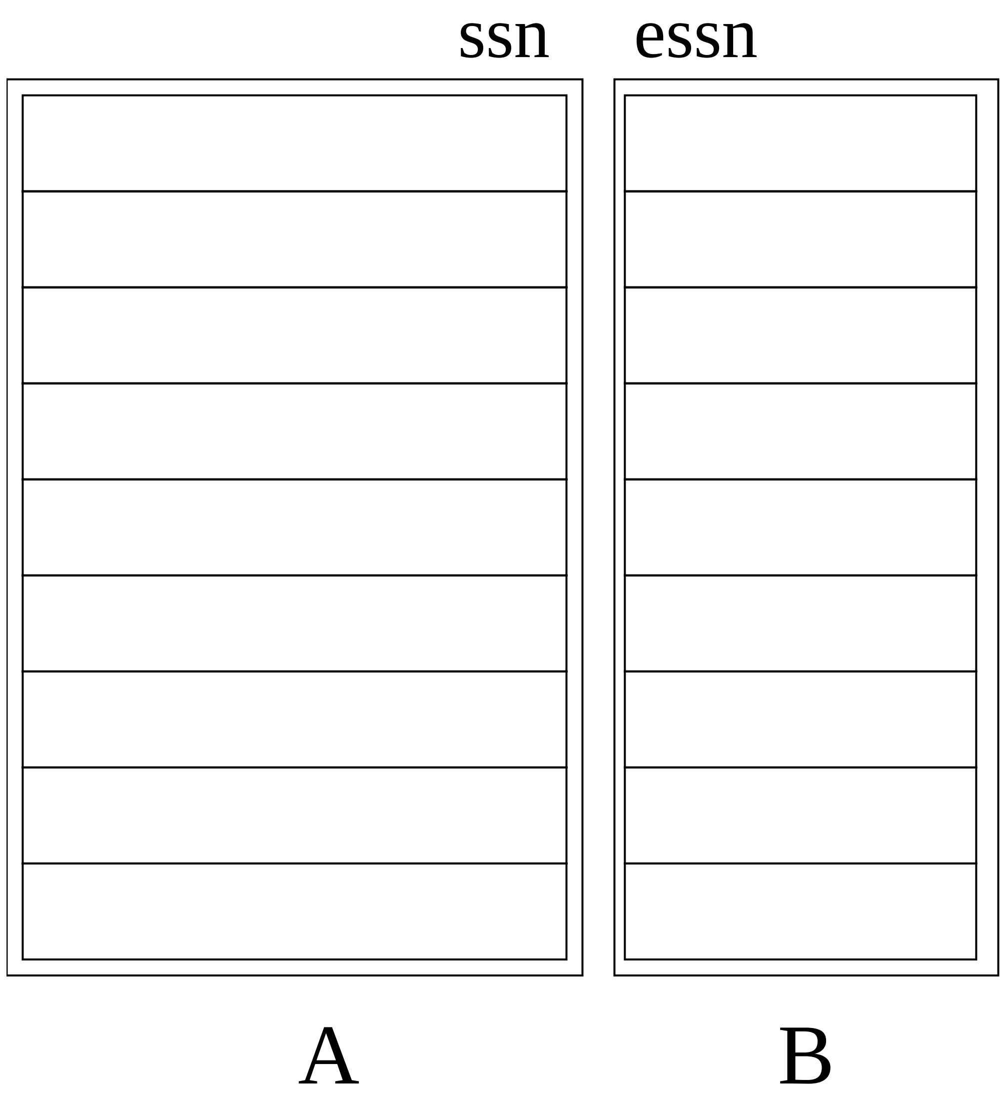
where the equality conditions prescribe that corresponding attribute values be equal

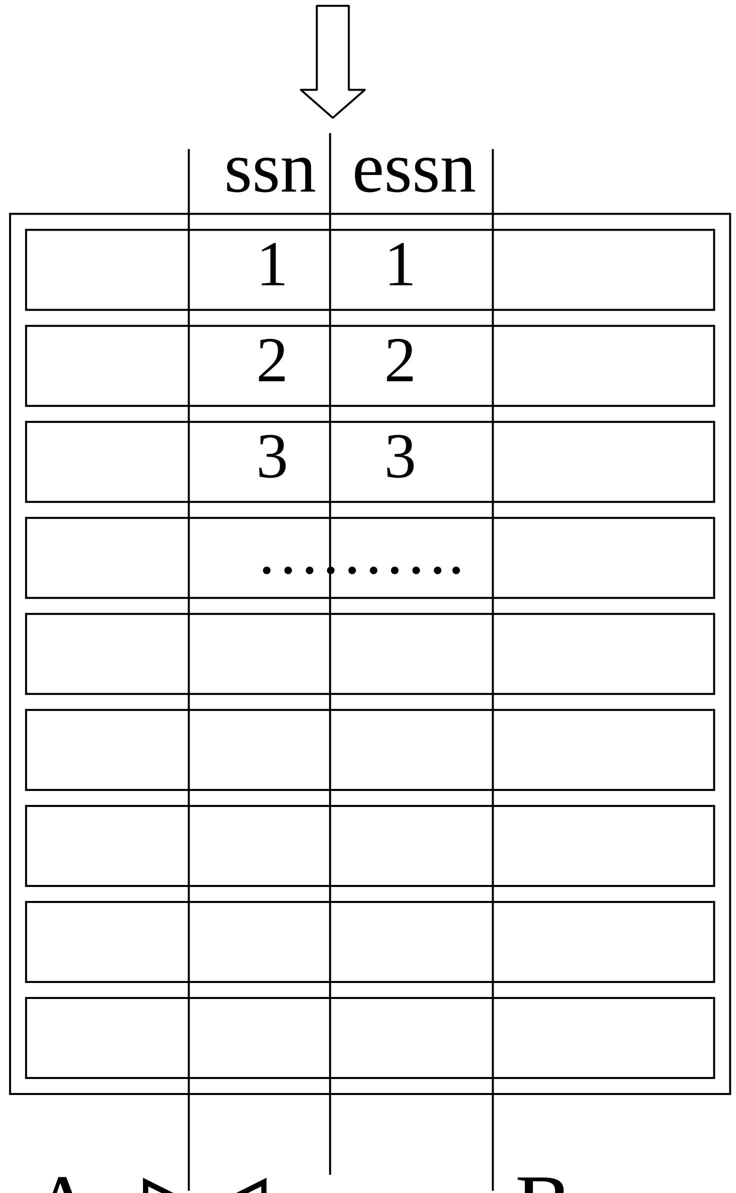
---

E.g. Employee $\bowtie_{SSN = ESSN}$ Dependent

will join the corresponding tuples of Employee and Dependent

Unnecessary duplicates





$A \bowtie_{ssn=essn} B$

Natural join

- To remove the duplicate columns a variation of equijoin is used

$$A \ * _{\text{equality conditions}} B$$

Example:

$$\text{Employee} * _{\text{ssn=essn}} \text{Dependent}$$

- If no condition is listed, ***all corresponding attributes*** (ones with the same name) are joined through the equality condition
  But, be careful:

  Employee * Dependent

  a) would not work, because there are no attributes with the same name
  b) sometimes attribute names inadvertently join. ***It is best to always list the join condition explicitly!***

Division

$$A \div B$$

Example:

"List the employees who work on every project"

Take a simplified company schema:

employee(ssn)

project(pno)

works_on(ssn,pno)

- If we knew that every employee works on every project, then the works_on relation would be unnecessary, it could be reconstructed by the cartesian product

$$W = \text{employee} \times \text{project}$$

- if we took away works_on from W:

$$W \setminus \text{works\_on}$$

the result would contain potential, but not actual works_on relationship instances

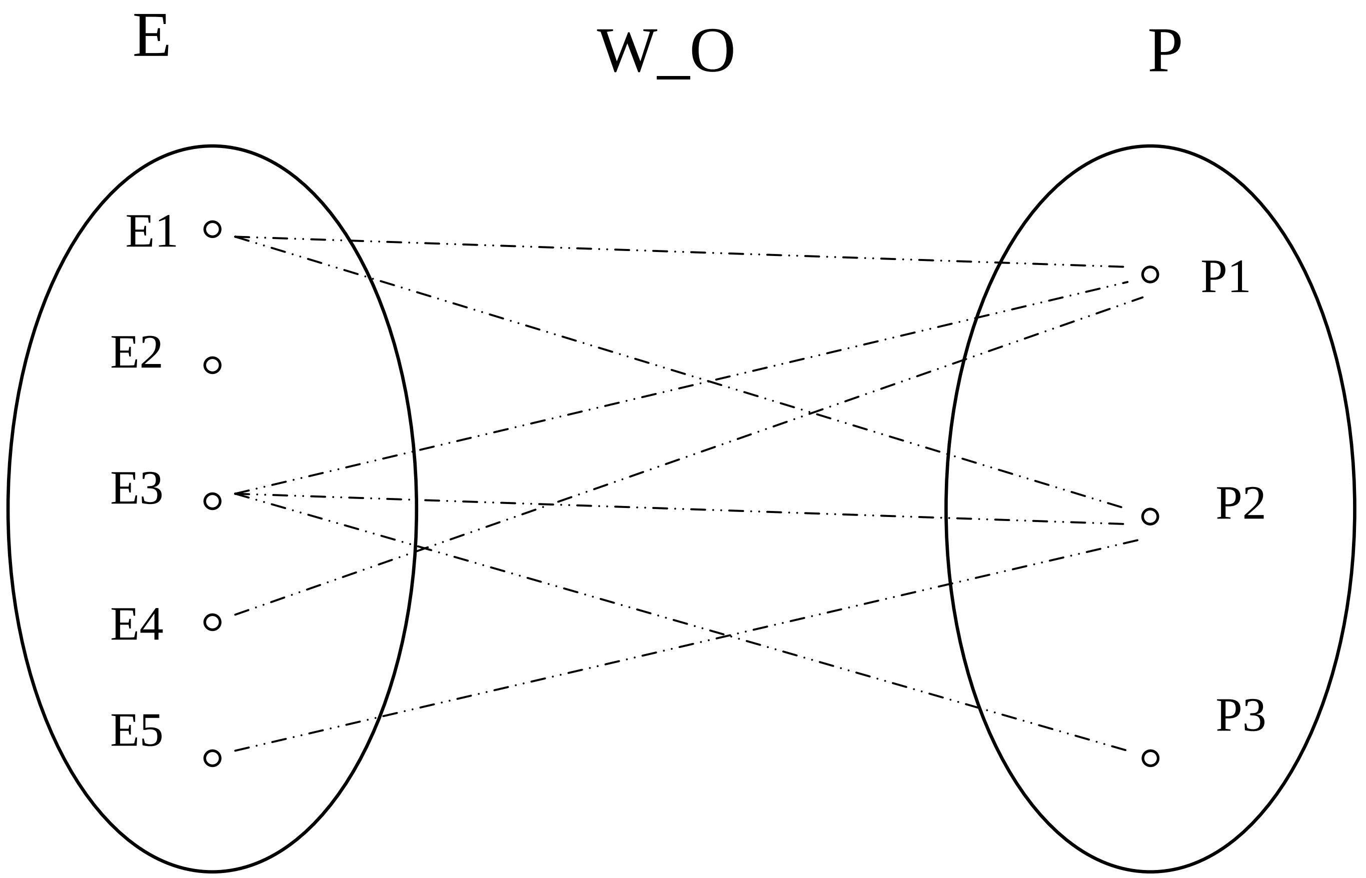
- Therefore

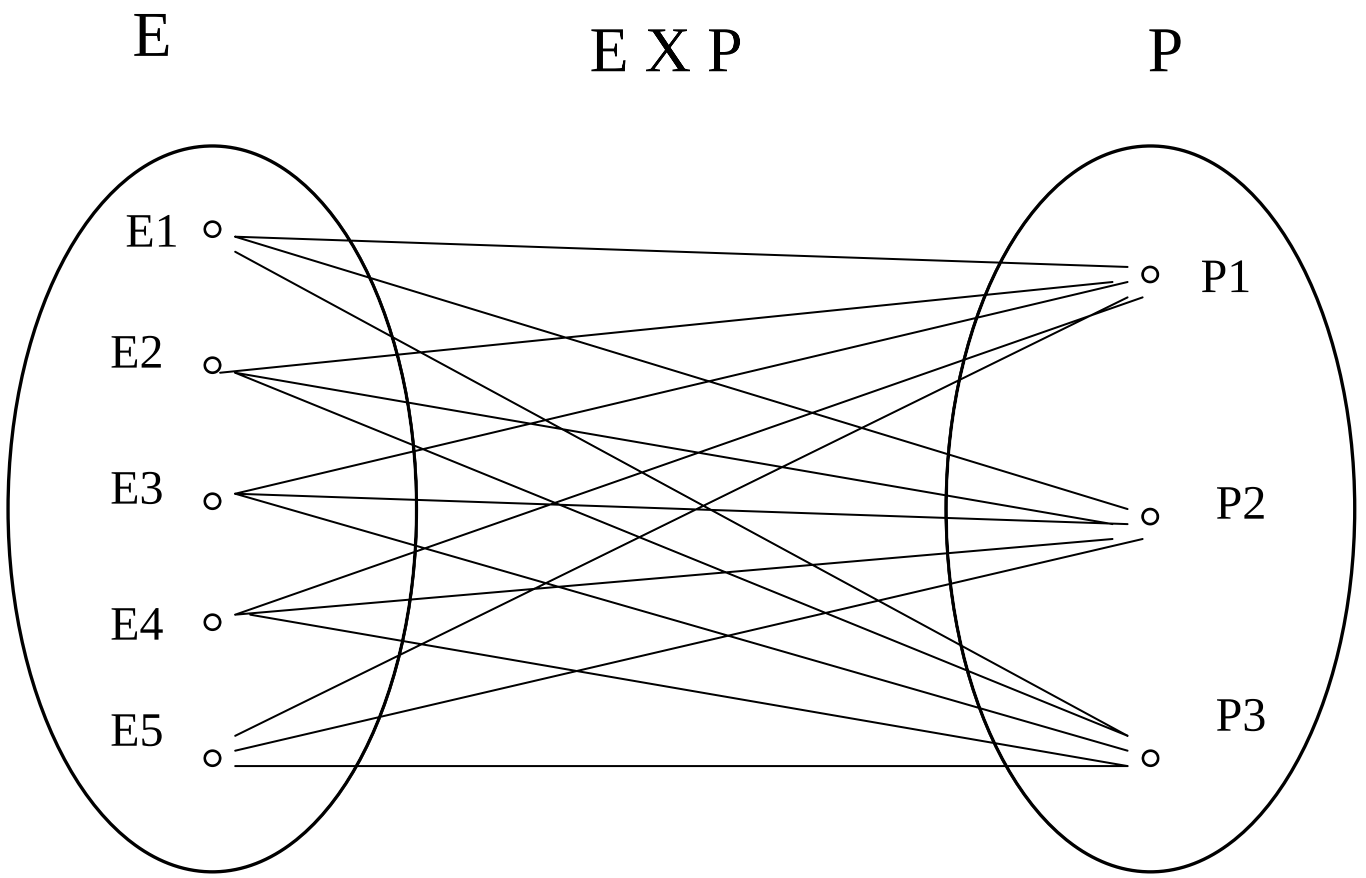
Π_{ssn} (Employee x Project \ Works_on)

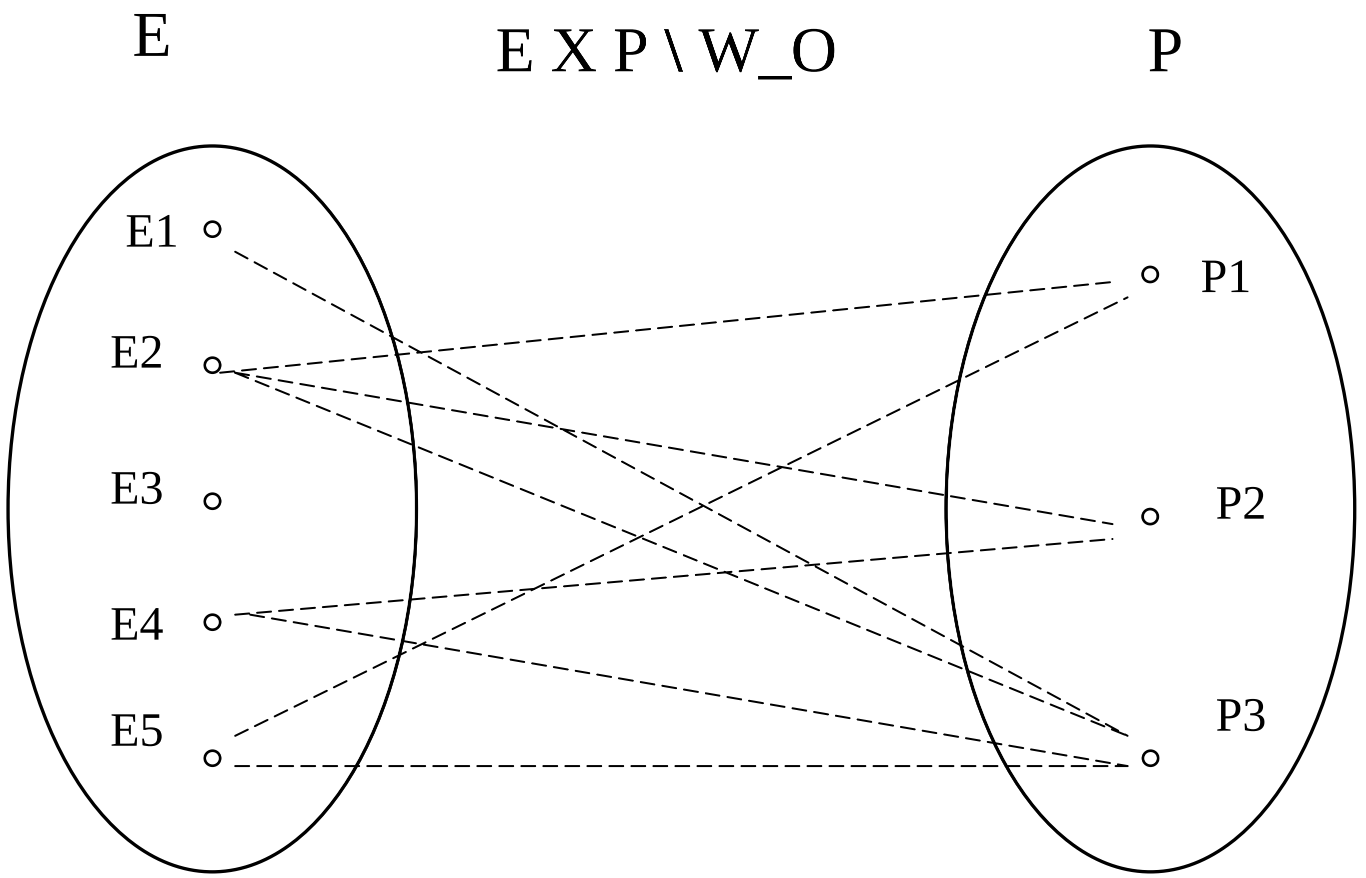
is the set of emplooyees who do *not* work on every project

- To calculate the set of employees who work on every project we take this away from employee:

$$\Pi_{ssn} \text{ Employee} \setminus \Pi_{ssn} (\text{ Employee x Project } \setminus \text{Works\_on})$$

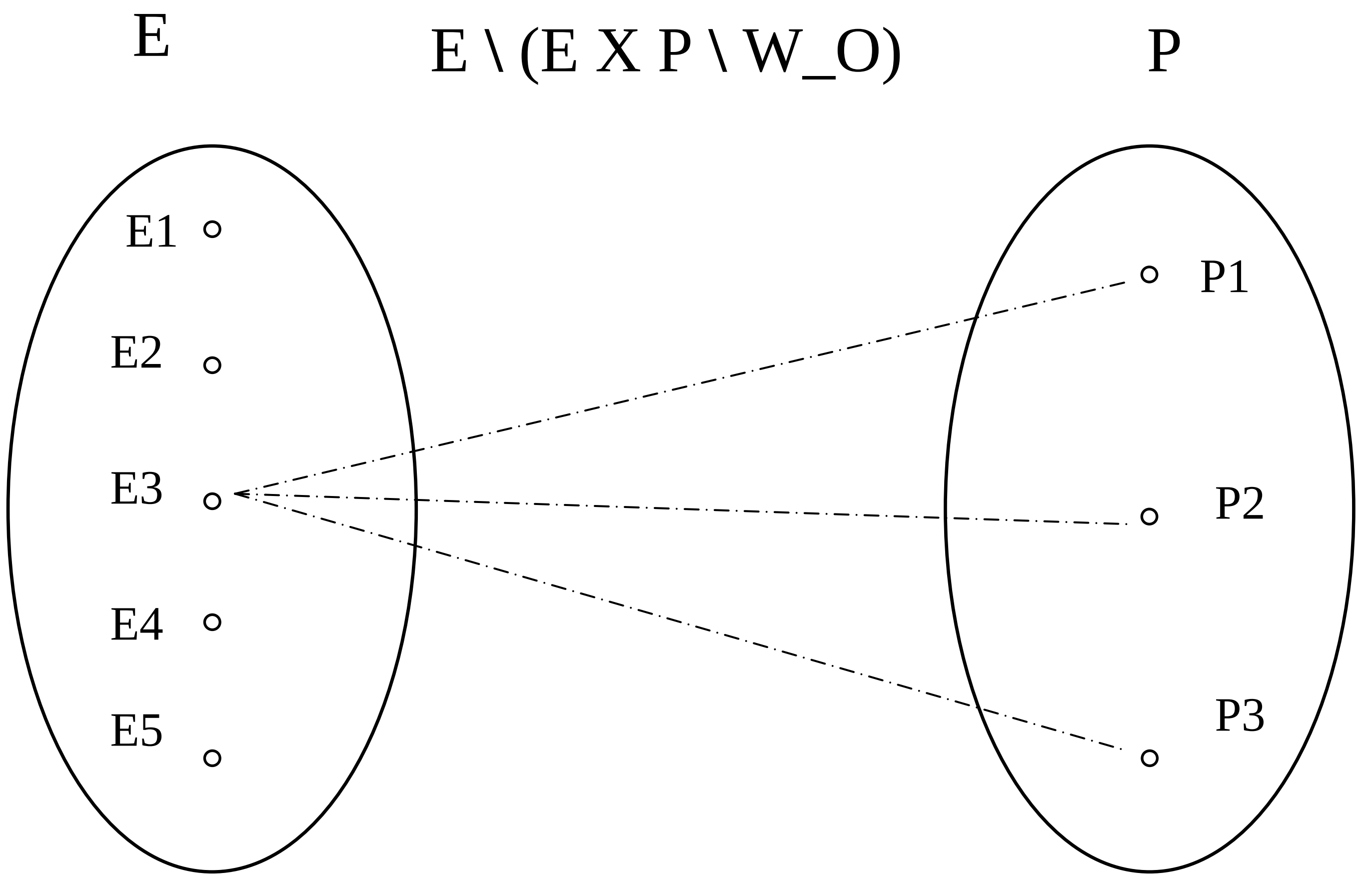

E
W_O
P
E1
E2
E3
E4
E5
P1
P2
P3


E
E X P
P
E1
E2
E3
E4
E5
P1
P2
P3



Employees who do *not* work on every Project

E \ (E X P \ W_O)



Employees who work on every Project

Introduce the division operator:

$$\text{Works\_on}(ssn,pno) \div \text{Project}(pno) \equiv$$
$$\Pi_{ssn}\,\text{Employee} \setminus (\Pi_{ssn}(\text{Employee} \times \text{Project} \setminus \text{Works\_on}))$$

We can do this in general as well:

Definition

$$A(a,b) \div B(b) \equiv \text{the largest set } C\ (a) \text{ such that } B \times C \subseteq A$$

Simplifying relational algebra expressions

- 10*5 + 10* 7 + 10*8 =
  10 * (5+7+8) = 10 * 20 = 200
- In the same way expressions in relational algebra can be simplified based on the discussed properties of the operators
- Query processors evaluate queries using simplification or to the contrary, elaboration so that the execution is cheaper (e.g cutting down on tuples which will later be discarded anyway)

Example

Simpler

$$\Pi_{\text{ssn,name}} (\sigma_{\text{name='mary'}} \text{ Employee } *_{\text{dno=dnumber}} \sigma_{\text{dname='research'}} \text{ Department}) =$$

$$\Pi_{\text{ssn,name}} (\Pi_{\text{ssn,name,dno}} \sigma_{\text{name='mary'}} \text{ Employee } *_{\text{dno=dnumber}} \Pi_{\text{dnumber}} \sigma_{\text{dname='research'}} \text{ Department})$$

Cheaper to evaluate

Query plan

- Relational algebra operators are implemented as parametric programming language functions, manipulating data structures that implement sets of tuples (relations)
- A query plan is a suitable concatenation (or parallel execution model) of such functions

Query plans can be evaluated ..

- …by the main processor of the computer on which the DBMS runs
- Simple operations, like select or project, can be performed by the disk drive itself - in order to avoid the transfer of massive amounts of data to main memory (later to be discarded anyway)

Conclusion

- Relational algebra has been introduced as an algebra of relations
- Each operator (monadic or binary) works on relations and returns relations
- Additional operations exist to carry out aggregation (this will be discussed only when studying SQL).
- Relational algebra can be used to express queries, as expressions to be evaluated on a database instance
- **Caution:** The result will depend on the actual database instance. The expression should be correct for ***every / any*** database instance.

The end