CS698F Advanced Data Management

Instructor: Medha Atre

Recap

Database Mgmt Sys

Schema generation & normalization

SQL query parsing, relational algebra

File Sys, Indexes, Query optimization

Distributed data mgmt, query processing

Transaction mgmt, crash recovery, concurrency control

Schema generation

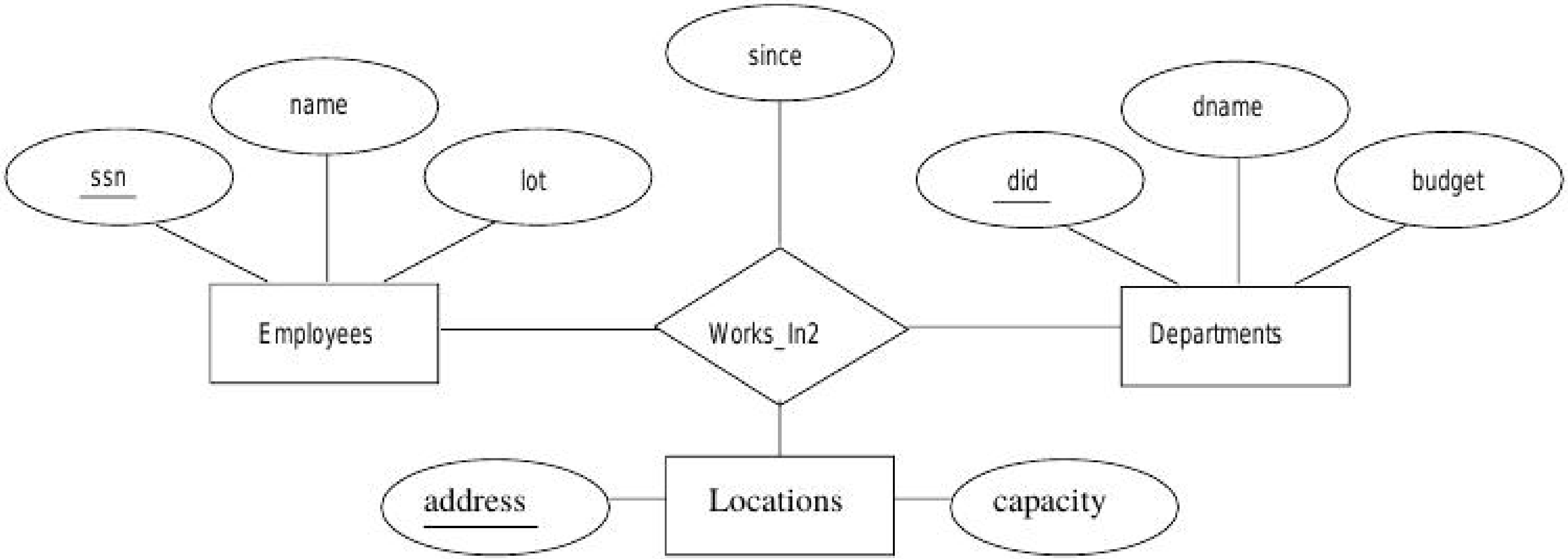


Fig taken from DBMS book.

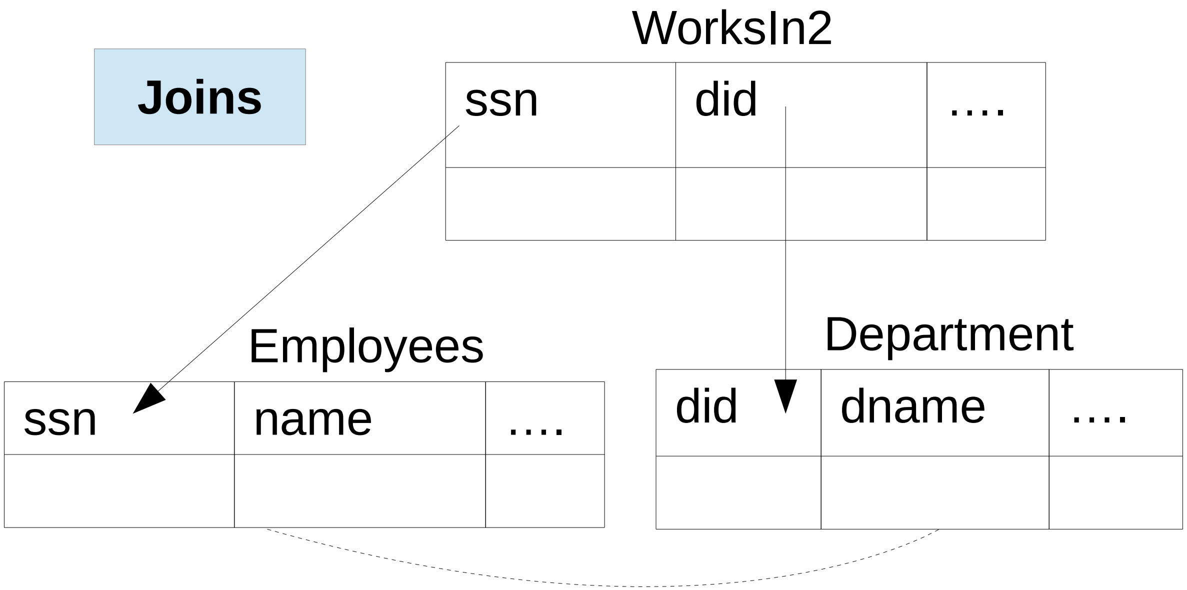
Table creation (for schema)

- *Employees(<u>ssn</u>, name, lot)*
- *Departments(<u>did</u>, dname, budget)*
- *Locations(<u>address</u>, capacity)*
- *WorksIn2(ssn, did, address, since,*
  <and write Foreign key refs>)

SQL queries

SELECT E.name, D.dname
FROM WorksIn2 as W,
Employees as E,
Department as D
WHERE
W.ssn=E.ssn AND
W.did="CSE" AND
W.did=D.did

Joins



Relational Algebra

- *Algebraic* representation of the SQL queries.
  - $\Pi_{(E.name,\ D.dname)}$ (W ⋈ E ⋈ D)
  - Π Projection symbol (what you are *SELECTing*)
  - ⋈ Join symbol (what tables are in *FROM* clause)
  - Join conditions are in the *WHERE* clause
    - Selection conditions, e.g., *D.did = "CSE"*
    - Join conditions, e.g., *W.ssn = E.ssn, W.did = D.did*

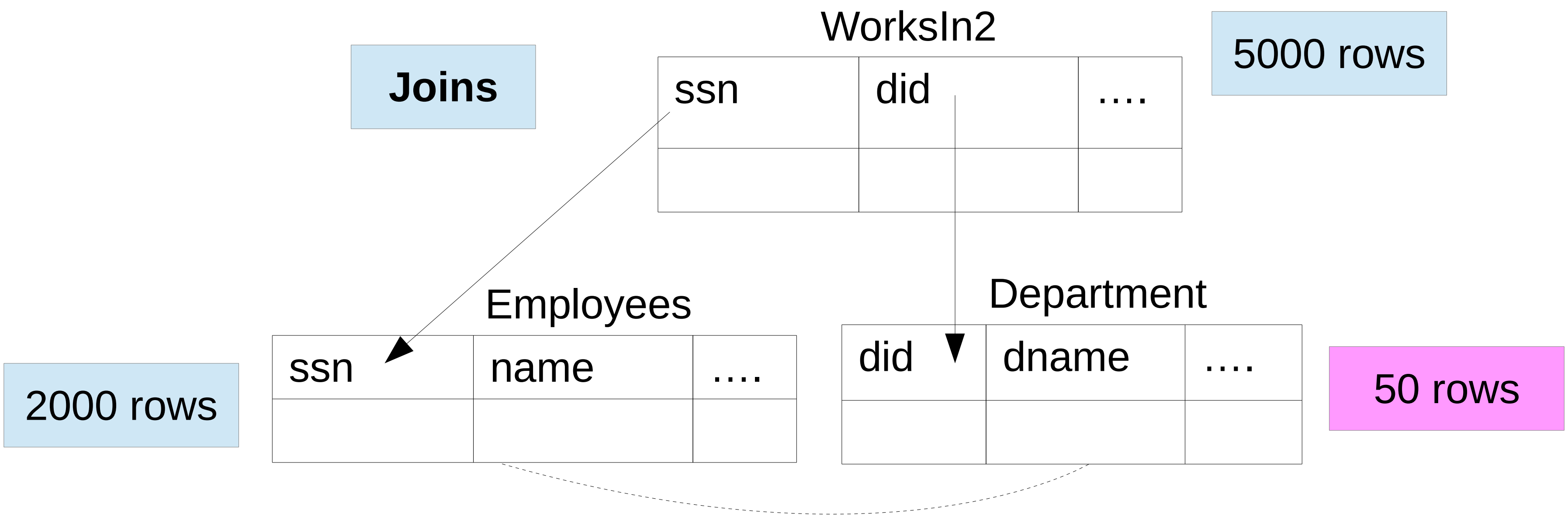
Relational Algebra

- Allows algebraic and set-theoretic operations
  - Helps in finding variety of *query plans* (we will visit this)
  - $(W \bowtie E \bowtie D) \equiv (E \bowtie W \bowtie D) \equiv ((W \bowtie E) \bowtie D)$
  - Joins are commutative and associative
- SQL has other set like operations as UNION
  - $(W1 \cup W2) \bowtie (E1 \cup E2) \equiv (W1 \bowtie E1) \cup (W1 \bowtie E2) \cup (W2 \bowtie E1) \cup (W2 \bowtie E2)$

Relational Algebra and SQL

- Equivalence of various relational algebraic terms means many equivalent SQL queries possible to output *exactly* same results.
  - This is what is called as various *query plans*.
  - This gives opportunities for *cost estimation* and *cost optimization*.
  - Let us see an example… continued...

Relational Algebra and SQL



Relational Algebra and SQL

- Joins are commutative and associative
  - Alternatives for performing (W ⋈ E ⋈ D)
  - First R1= (W ⋈ E) and then R2 = (R1 ⋈ D)
  - First R1= (W ⋈ D) and then R2 = (R1 ⋈ E)
  - First R1 = (E ⋈ D) and then R2 = (R1 ⋈ W)
- Which one of these do you think will be *low cost?*
  - Cost is w.r.t. number of total join operations needed

Relational Algebra and SQL

- Joins are commutative and associative
  - Alternatives for performing (W ⋈ E ⋈ D)
  - First R1= (W ⋈ E) (5000 x 2000) and then R2 = (R1 ⋈ D) (5000 x 2000 x 50) in reality this number will be much lower due to *join selectivity*.
  - First R1= (W ⋈ D) (5000 x 50) and then R2 = (R1 ⋈ E)
  - First R1 = (E ⋈ D) (2000 x 50) and then R2 = (R1 ⋈ W)
- So would you choose (E ⋈ D)?
  - No! Because E and D do not share any attribute
  - It is a Cartesian product, no reduction in result size due to *join selectivity*.

Join Selectivity and Why it matters

- Join selectivity is *how many results* will actually get generated?
  - Never equal to Cartesian product! Why?
- Usually not every column value appears on both the sides.
- Even if it does, simple math proves that join results will never be equal to Cartesian product!
  - $(W \bowtie D)$!= W x D (x => Cartesian product)

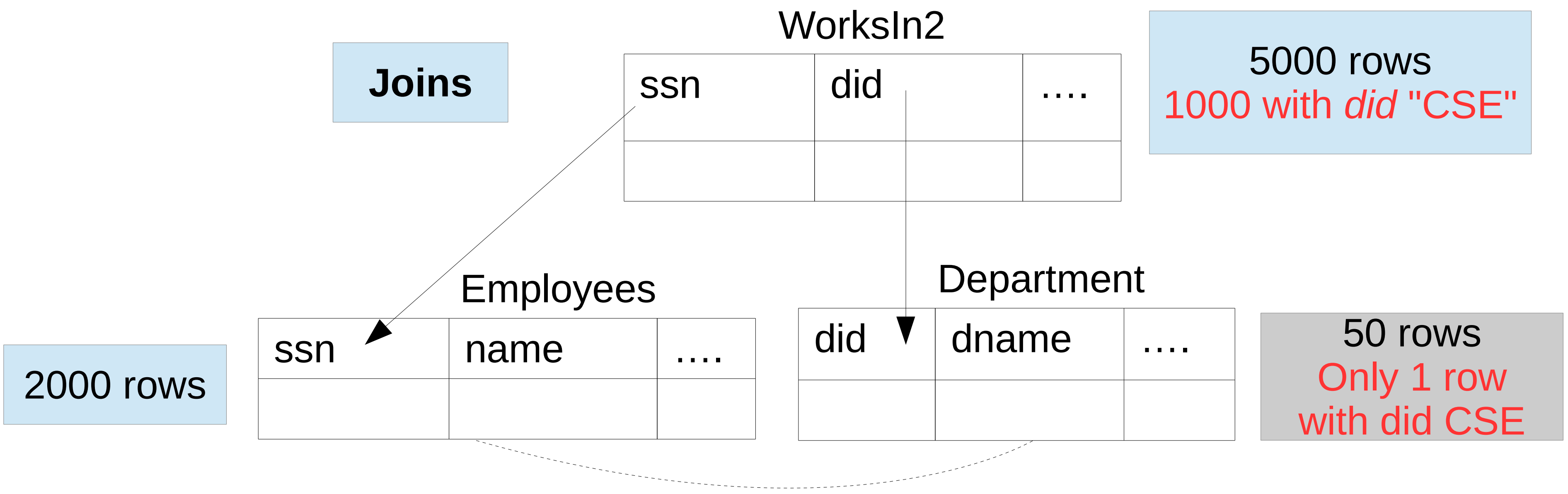
Some thumbrules

- Take into consideration "selection conditions"
  - W.did = "CSE"
  - They reduce join space
  - Now we will consider only rows with W.did="CSE", and not all 5000 rows!
- Two tables with least number of rows (before or after selection cond) to be joined first, and apply the rule inductively.

Some thumbrules

- Take into consideration *metadata* information from *System Catalogue*
  - Are there any indexes on the tables?
  - Are there any histograms on the table columns?
  - How do these help?
  - Let us see.

Some thumbrules



Some thumbrules

- Can you guess which join to do first now?
- Why?
- Can you formulate a stepwise procedure for any join query?
  - We will review this in the next lecture.
- How would you know that there are 1000 rows with *did* "CSE" in *WorksIn2* table?
  - Indexes!! => the heart and soul of all the data management on the web
  - Google, Facebook, Amazon all survive due to intelligent indexes
  - And we will review some of those.