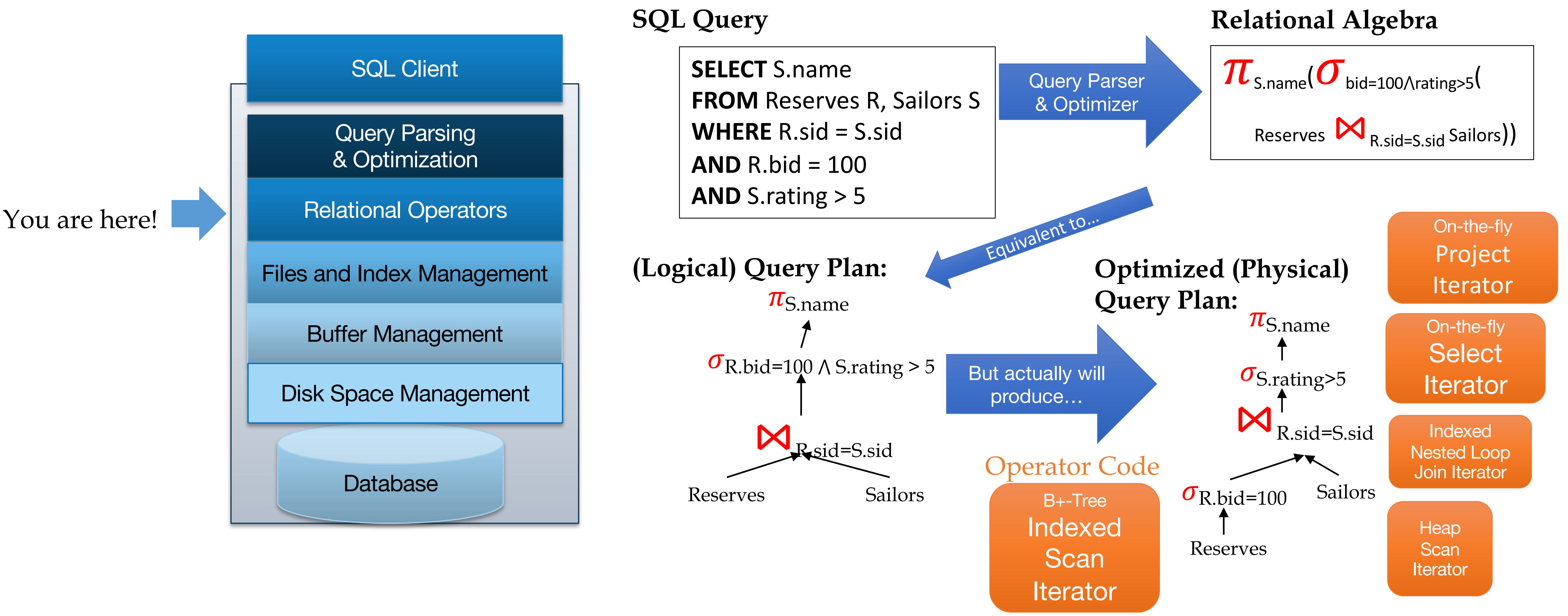
# Introduction to Database Systems

2023-Fall

# 2. Data Model

# Relational Algebra



# Relational Algebra Preliminaries

- Algebra of operators on relation instances
- $\pi_{\text{S.name}}(\sigma_{\text{R.bid=100} \wedge \text{S.rating>5}}(\text{R} \bowtie_{\text{R.sid=S.sid}} \text{S}))$
    - ***Closed***: result is also a relation instance
        - Enables rich composition!
    - ***Typed***: input schema determines output
        - Can statically check whether queries are legal.

# Relational Algebra and Sets

- Pure relational algebra has set semantics
  - ***No duplicate*** tuples in a relation instance
  - vs. SQL, which has ***multiset (bag) semantics***
  - We will switch to multiset in the system discussion

# Relational Algebra

- **Basic operations:**
  - *Selection* ($\sigma$) Selects a subset of rows from relation.
  - *Projection* ($\pi$) Deletes unwanted columns from relation.
  - *Cross-product or Cartesian product* ($\times$) Allows us to combine two relations.
  - *Set-difference* ($-$) Tuples in reln. 1, but not in reln. 2.
  - *Union* ($\cup$) Tuples in reln. 1 and in reln. 2.
  - *rename*: $\rho$
- $\{\sigma, \pi, \cup, -, \times\}$ is a ***complete operation set***. Any other relational algebra operations can be derived from them.
- Additional operations:
  - Intersection, ***join***, division, outer join, outer union: Not essential, but (very!) useful.

# 1.Selection

- Selecting rows that satisfy ***selection condition***.
- **No duplicates** in result! (Why?)
- ***Schema*** of result identical to that of input relation.
- ***Result*** relation can be the ***input*** for another relational algebra operation! (*Operator composition.*)

# Selection Example

- "Sailors", "Reserves" and "Boats" relations for our examples.

*S2*

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

$$\sigma_{rating>8}(S2)$$

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 58 | rusty | 10 | 35.0 |

# 2.Projection

- ***Deletes*** attributes that are not in *projection list*.
- Schema of result contains exactly the fields in the projection list, with the same names that they had in the input relation.
- Projection operator has to eliminate ***duplicates***!
  - ***Note***: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

# Projection Example

- "Sailors", "Reserves" and "Boats" relations for our examples.

*S2*

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

$$\pi_{sname,rating}(S2)$$

| sname | rating |
|---|---|
| yuppy | 9 |
| lubber | 8 |
| guppy | 5 |
| rusty | 10 |

$$\pi_{age}(S2)$$

| age |
|---|
| 35.0 |
| 55.5 |

# Operator composition

- ***Result*** relation can be the ***input*** for another relational algebra operation! (*Operator composition.*)

*S2*

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

$$\pi_{sname,rating}(\sigma_{rating>8}(S2))$$

| sname | rating |
|---|---|
| yuppy | 9 |
| rusty | 10 |

# 3.Cross-Product

***S1***

| <u>sid</u> | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

***R1***

| <u>sid</u> | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

- Each row of S1 is paired with each row of R1.
- ***Result schema*** has one attribute per attribute of S1 and R1, with attribute names *inherited* if possible.
    - *Conflict*: Both S1 and R1 have an attribute called *sid*. ***Renaming operator.***

$$\rho\,(C(1 \rightarrow sid1, 5 \rightarrow sid2),\, S1 \times R1)$$

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

# 6.Renaming ( $\rho$ = “rho” )

- *Renames relations and their attributes:*
- Note that relational algebra doesn’t require names.
  - We could just use positional arguments.

**sid1** **sid2**

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 22 | 101 | 10/10/96 |
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 22 | 101 | 10/10/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |
| 58 | rusty | 10 | 35.0 | 22 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 58 | 103 | 11/12/96 |

# Union, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - Corresponding attributes have the same type.

# 5. Union

*S1*

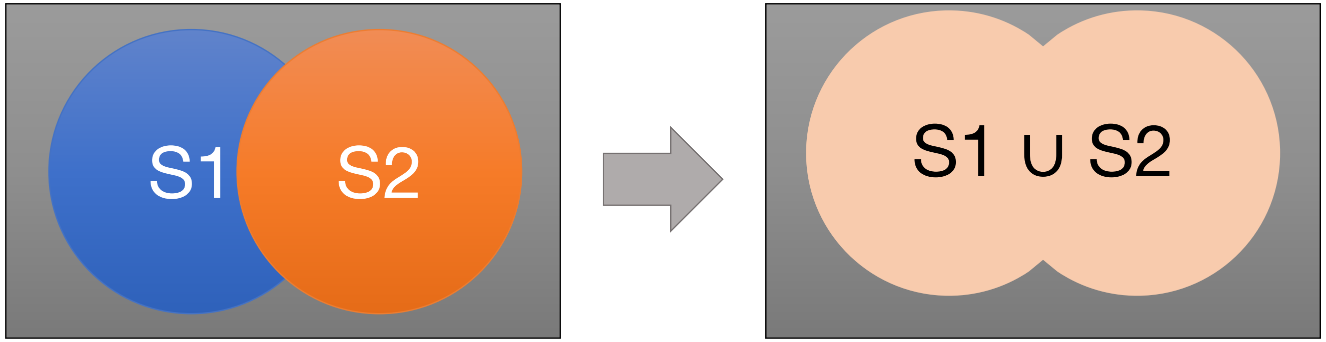
| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*S2*

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

- Two input relations, must be *compatible*:
  - Same number of fields
  - Fields in corresponding positions have same **type**
- SQL Expression: UNION vs. UNION ALL

**S1 U S2**




| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |
| 44 | guppy | 5 | 35.0 |
| 28 | yuppy | 9 | 35.0 |

$S1 \cup S2$

# 4. Set Difference ( − )

*S1*

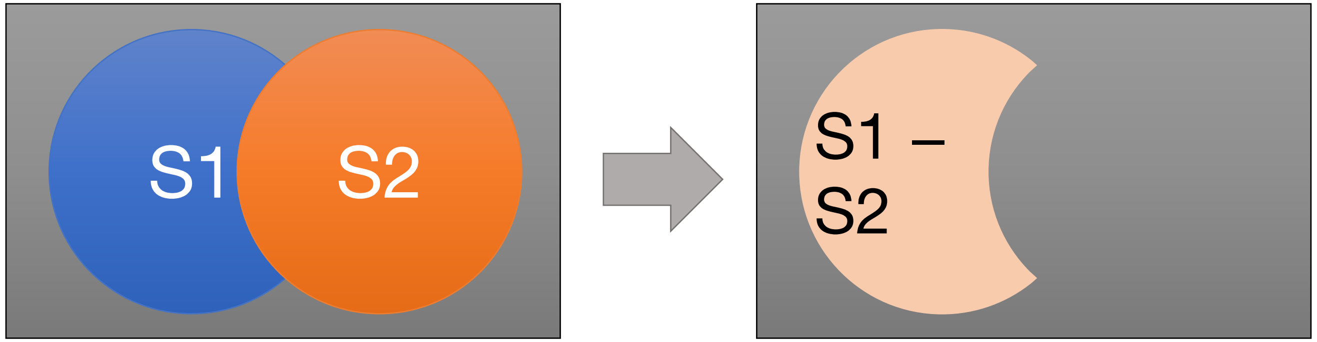
| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*S2*

| sid | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

- Same as with union, both input relations must be *compatible*.
- SQL Expression: EXCEPT

**S1 − S2**




| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |

$S1 - S2$

# Operator composition: Intersection

*S1*

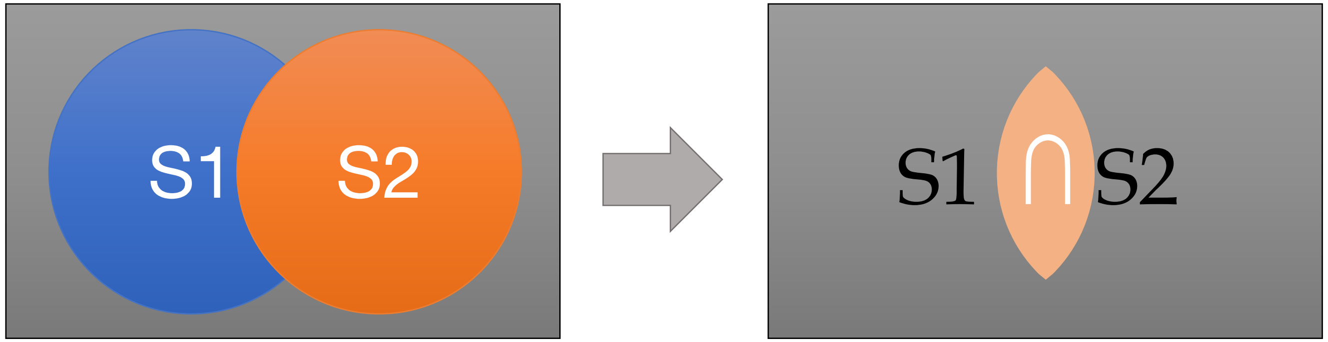
| <u>sid</u> | sname | rating | age |
|---|---|---|---|
| 22 | dusti n | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*S2*

| <u>sid</u> | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

- Same as with union, both input relations must be *compatible*.
- SQL Expression: INTERSECT

**S1 ⋂ S2** ***Equivalent to: S1 — (S1 — S2)***




| sid | sname | rating | age |
|---|---|---|---|
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

$S1 \cap S2$

## Operator composition: Joins

*S1*

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*R1*

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

- ***Condition Join*** : $R \bowtie_C S = \sigma_C (R \times S)$

| (sid) | sname | rating | age | (sid) | bid | day |
|---|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 58 | 103 | 11/12/96 |
| 31 | lubber | 8 | 55.5 | 58 | 103 | 11/12/96 |

$S1 \bowtie_{S1.sid<R1.sid} R1$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*. $\bowtie_\theta$

Note: we will need to learn a good join algorithm.
Avoid cross-product if we can!!

# Operator composition: Joins

*S1*

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

***R1***

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

- ***Equi-Join***: A special case of condition join where the condition *c* contains only ***equalities***.

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |

$\Leftarrow$ $S1 \bowtie_{sid} R1$

- *Result schema* similar to cross-product, but only one copy of attributes for which equality is specified.
- *Special special case*
  - ***Natural Join***: Equi-join on *all* common attributes.

# Natural Join ($\bowtie$)

- Special case of equi-join in which equalities are specified for all matching fields and duplicate fields are projected away

$$R \bowtie S = \pi_{\text{unique fld.}} \, \sigma_{\text{eq. matching fld.}} ( R \times S )$$

- Compute $R \times S$
- Select rows where fields appearing in both relations have equal values
- Project onto the set of all unique fields.

# Natural Join (⋈)

*S1*

| <u>sid</u> | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*R1*

| <u>sid</u> | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

- $\mathbf{R} \bowtie \mathbf{S} = \pi_{\text{unique fld.}}\, \sigma_{\text{eq. matching fld.}}(\mathbf{R} \times \mathbf{S})$

R1 ⋈ S1

| sid | bid | day | sid | sname | rating | age |
|---|---|---|---|---|---|---|
| 22 | 101 | 10/10/96 | 22 | dustin | 7 | 45.0 |
| ~~22~~ | ~~101~~ | ~~10/10/96~~ | ~~31~~ | ~~lubber~~ | ~~8~~ | ~~55.5~~ |
| ~~22~~ | ~~101~~ | ~~10/10/96~~ | ~~58~~ | ~~rusty~~ | ~~10~~ | ~~35.0~~ |
| ~~58~~ | ~~103~~ | ~~11/12/96~~ | ~~22~~ | ~~dustin~~ | ~~7~~ | ~~45.0~~ |
| ~~58~~ | ~~103~~ | ~~11/12/96~~ | ~~31~~ | ~~lubber~~ | ~~8~~ | ~~55.5~~ |
| 58 | 103 | 11/12/96 | 58 | rusty | 10 | 35.0 |

| sid | bid | day | sname | rating | age |
|---|---|---|---|---|---|
| 22 | 101 | 10/10/96 | dustin | 7 | 45.0 |
| 58 | 103 | 11/12/96 | rusty | 10 | 35.0 |

# Outer Joins

- The extension of join operation. In join operation, only matching tuples fulfilling join conditions are left in results. Outer joins will keep ***unmated tuples***, the vacant part is set ***Null*** :
  - ***Left outer join***($*\bowtie$)

    Keep all tuples of left relation in the result.
  - ***Right outer join*** ($\bowtie*$)

    Keep all tuples of right relation in the result.
  - **Full outer join** ($*\bowtie*$)

    Keep all tuples of left and right relations in the result.

# Examples of Outer Joins

*S1*

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*R1*

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |
| 31 | Lubber | 8 | 55.5 | null | null |

S1 *⋈ R1

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |

S1⋈* R1 =

S1⋈R1 **(Why?)**

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | 101 | 10/10/96 |
| 58 | rusty | 10 | 35.0 | 103 | 11/12/96 |
| 31 | Lubber | 8 | 55.5 | null | null |

S1 *⋈* R1

# Outer Unions

*S1*

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

*R1*

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

- The extension of union operation. It can union two relations which are not *union-compatible*.
- The attribute set in result is the union of attribute sets of two operands.
- The values of attributes which don't exist in original tuples are filled as ***NULL***

| sid | sname | rating | age | bid | day |
|---|---|---|---|---|---|
| 22 | dustin | 7 | 45.0 | null | null |
| 31 | Lubber | 8 | 55.5 | null | null |
| 58 | rusty | 10 | 35.0 | null | null |
| 22 | null | null | null | 101 | 10/10/96 |
| 58 | null | null | null | 103 | 11/12/96 |

$S1 \underline{\cup} R1$

# Division

- Not supported as a primitive operator, but useful for expressing queries like:
  - *Find sailors who have reserved **<u>all</u>** boats.*
- Let *A* have 2 fields, *x* and *y*; *B* have only field *y*:
  - *A/B* = $\left\{\langle x\rangle \mid \exists \langle x, y\rangle \in A \ \ \forall \langle y\rangle \in B\right\}$
  - i.e., *A/B* contains all *x* tuples (sailors) such that for *<u>every</u>* *y* tuple (boat) in *B*, there is an *xy* tuple in *A*.
  - *Or*: If the set of *y* values (boats) associated with an *x* value (sailor) in *A* contains all *y* values in *B*, the *x* value is in *A/B*.
- In general, *x* and *y* can be any lists of fields; *y* is the list of fields in *B*, and *x* ∪ *y* is the list of fields of *A*.

# Examples of Division A/B

| sno | pno |
|---|---|
| s1 | p1 |
| s1 | p2 |
| s1 | p3 |
| s1 | p4 |
| s2 | p1 |
| s2 | p2 |
| s3 | p2 |
| s4 | p2 |
| s4 | p4 |

*A*

| pno |
|---|
| p2 |

*B1*

| pno |
|---|
| p2 |
| p4 |

*B2*

| pno |
|---|
| p1 |
| p2 |
| p4 |

*B3*

| sno |
|---|
| s1 |
| s2 |
| s3 |
| s4 |

*A/B1*

| sno |
|---|
| s1 |
| s4 |

*A/B2*

| sno |
|---|
| s1 |

*A/B3*

# Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)
- ***Idea***: For *A/B*, compute all *x* values that are not *disqualified* by some *y* value in *B*.
  - *x* value is *disqualified* if by attaching *y* value from *B*, we obtain an *xy* tuple that is not in *A*.

**Disqualified *x* values:** $\pi_x((\pi_x(A) \times B) - A)$

***A/B:*** $\pi_x(A) -$ ***all disqualified tuples***

# Relational Algebra Operators: Unary

- ***Unary Operators***: on **single relation**
- **Projection** ($\pi$): Retains only desired columns (vertical)
- **Selection** ($\sigma$): Selects a subset of rows (horizontal)
- **Renaming** ( $\rho$ ): Rename attributes and relations.

# Relational Algebra Operators: Binary

- ***Binary Operators***: on **pairs of relations**
- **Union** ( $\cup$ ): Tuples in r1 or in r2.
- **Set-difference** ( — ): Tuples in r1, but not in r2.
- **Cross-product** ( $\times$ ): Allows us to combine two relations.

# Relational Algebra Operators: Compound

- ***Compound Operators***: common "*macros*" for the above
- **Intersection** ( $\cap$ ): Tuples in r1 and in r2.
- **Joins** ( $\bowtie_\theta$ , $\bowtie$ ): Combine relations that satisfy predicates

# Relational Calculus

- Relational Algebra needs to specify the order of operations; while relational calculus only needs to indicate the logic condition the result must be fulfilled.
- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops*, *logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (attribute values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

# Domain Relational Calculus

- *Query* has the form:

  $\{ <x_1,x_2, \ldots, x_n> \mid P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m}) \}$

- $x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m}$ are called *domain variables*. $x_1,x_2,\ldots,x_n$ appear in result.

✓ *Answer* includes all tuples $<x_1,x_2, \ldots, x_n>$ that make the ***formula*** $P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m})$ be *true*.

✓ *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

| <u>sid</u> | sname | rating | age |
|---|---|---|---|
| 28 | yuppy | 9 | 35.0 |
| 31 | lubber | 8 | 55.5 |
| 44 | guppy | 5 | 35.0 |
| 58 | rusty | 10 | 35.0 |

# Find all sailors with a rating above 7

- { <*I,N,T,A*> | <*I,N,T,A*> ∈ *Sailors* ∧ T>7 }
- The condition <*I,N,T,A*> ∈ *Sailors* ensures that the ***domain variables*** *I, N, T* and *A* are bound to fields of the same Sailors tuple.
- The term <*I,N,T,A*> to the left of '|' (which should be read as *such that*) says that every tuple <*I,N,T,A*> that satisfies *T*>7 is in the answer.
- Modify this query to answer:
  - Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

# Relational Algebra vs. Relational Calculus

- Projection $\pi_{AB} = \{t[AB] | t \in r\}$
- Selection $\sigma_F(R) = \{t | t \in r \ \wedge \ F\}$
- Union $r \cup s = \{t | t \in r \ \vee t \in s\}$
- Set difference $r - s = \{t | t \in r \ \wedge \neg(t \in s)\}$
- Join $r \bowtie s = \{t(ABCDE) | t[ABC] \in r \ \wedge (t[CDE] \in s)\}$

$R(ABC)$

$\mathrm{S}(CDE)$

# DRC Formulas

- *Atomic formula:*
  - $<x_1,x_2, \ldots, x_n> \in$ *Rname* , or X *op* Y, or X *op* constant
  - *op* is one of $<,>,=,\leq,\geq,\neq$
- *Formula:*
  - an atomic formula, or
  - $\neg p$, $p \wedge q$, $p \vee q$, where p and q are formulas, or
  - $\exists X(p(X))$, where variable X is ***free*** in p(X), or
  - $\forall X(p(X))$, where variable X is ***free*** in p(X)
- The use of quantifiers $\exists$ *X* and $\forall$ *X* is said to *bind* X.
  - A variable that is not bound is free.

# Free and Bound Variables

- The use of ***quantifiers*** $\exists X$ and $\forall X$ is said to ***bind*** X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

  $<x_1,x_2, \ldots, x_n> \mid P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m})$
- There is an important restriction: the variables $x_1,x_2, \ldots, x_n$ that appear to the left of '|' must be the ***only*** free variables in the formula p(...).

# Find sailors rated > 7 who've reserved boat # 103

- $\{<I,N,T,A>|\ <I,N,T,A> \in Sailors \wedge T>7 \wedge$
  $\exists\ Ir,Br,D\ [<Ir,Br,D> \in \mathrm{Reserves} \wedge Ir = I \wedge Br = 103]\}$
- We have used $\exists\ Ir, Br, D$ (...) as a shorthand for $\exists Ir(\exists Br(\exists D(...)))$
- Note the use of $\exists$ to find a tuple in Reserves that `*joins* with' the Sailors tuple under consideration.

**S1**

| sid | sname | rating | age |
|---|---|---|---|
| 22 | dustin | 7 | 45.0 |
| 31 | lubber | 8 | 55.5 |
| 58 | rusty | 10 | 35.0 |

**R1**

| sid | bid | day |
|---|---|---|
| 22 | 101 | 10/10/96 |
| 58 | 103 | 11/12/96 |

# Find sailors rated > 7 who've reserved a red boat

- $\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge T>7 \wedge$

  $\exists\ Ir,Br,D\ [\langle Ir,Br,D\rangle \in \mathrm{Reserves} \wedge Ir=I \wedge$

  $\exists\ B,BN,C\ [\langle B,BN,C\rangle \in Boats \wedge B=Br \wedge C='red'\ ]]\}$
- Observe how the parentheses control the scope of each quantifier's binding.
- This may look cumbersome, but with a good user interface, it is very intuitive. (MS Access, QBE)

# Find sailors who've reserved all boats

- $\{<I,N,T,A> \mid <I,N,T,A> \in Sailors \land$
  $\forall\, B,BN,C\ (\neg\, (<B,BN,C> \in Boats) \lor$

  $(\exists\, Ir,Br,D\ (<Ir,Br,D> \in \text{Reserves} \land I=Ir \land Br=B)))\}$
- Find all sailors *I* such that for each 3-tuple *B*,*BN*,*C* either it is not a tuple in Boats or there is a tuple in Reserves showing that sailor *I* has reserved it.

# Find sailors who've reserved all boats (again!)

- $\{\langle I,N,T,A\rangle \mid \langle I,N,T,A\rangle \in Sailors \wedge$

  $\forall\ B,BN,C \in Boats$

  $(\exists \langle Ir,Br,D\rangle \in \text{Reserves}\ (I=Ir \wedge Br=B))\}$
- Simpler notation, same query. (Much clearer!)
- To find sailors who've reserved all red boats:
- ..... $(C \neq \text{'red'} \vee \exists \langle Ir,Br,D\rangle \in \text{Reserves}\ (I=Ir \wedge Br=B\ ))\}$

$$\mathbf{R} = \pi_{r.ir,\ r.br}(\text{Reserves}) \qquad \mathbf{B} = \pi_{b.i}(\text{Boats})$$

**A/B?**

$$\mathbf{R1} = \mathbf{R/B} = \pi_{r.ir}(R) - \pi_{r.ir}(\pi_{r.ir}(R) \times \mathbf{B}) - R)$$

$$\mathbf{Result} = \pi_{r.ir,\ s.n.\ s,t,\ s.a}(R1 {*} \bowtie \text{Sailors})$$

# Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called ***unsafe***.
  - **e.g., $\{S \mid \neg(S \in \text{Sailors})\}$**
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- *Relational Completeness*: $\{\sigma, \pi, \cup, -, \times\}$ is a complete operation set. Relational calculus can express these five operations easily, so relational calculus is also *Relational Completeness*. SQL language is based on relational calculus, so it can express any query that is expressible in relational algebra /calculus.

# Summary

- Relational calculus is non-operational, and users define queries in terms of what they want, not in terms of how to compute it. (Declarativeness.)
- Algebra and safe calculus have same expressive power, leading to the notion of relational completeness.

# Remarks to Traditional Data Model

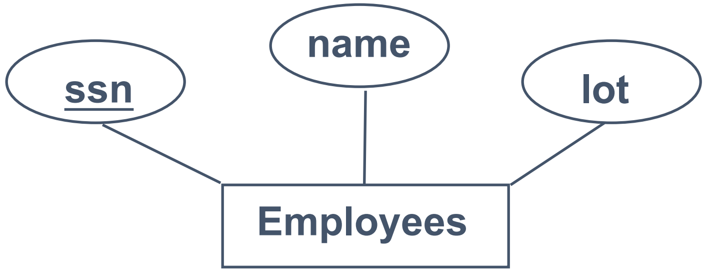
- Hierarchical, Network, and Relational Model
- Suitable for Online Transaction Processing (OLTP) applications
- Based on record, can’t orient to users or applications better
- Can’t express the relationships between entities in a natural mode.
- Lack of semantic information
- Few data type, hard to fulfill the requirements of applications

# 2.4 Entity-Relationship Model

- a graph-based model
  - can be viewed as a graph, or a veneer over relations
    - “feels” more flexible, less structured
  - corresponds well to “Object-Relational Mapping”
    - (ORM) SW packages
    - Ruby-on-Rails, Django, Hibernate, Sequelize, etc.

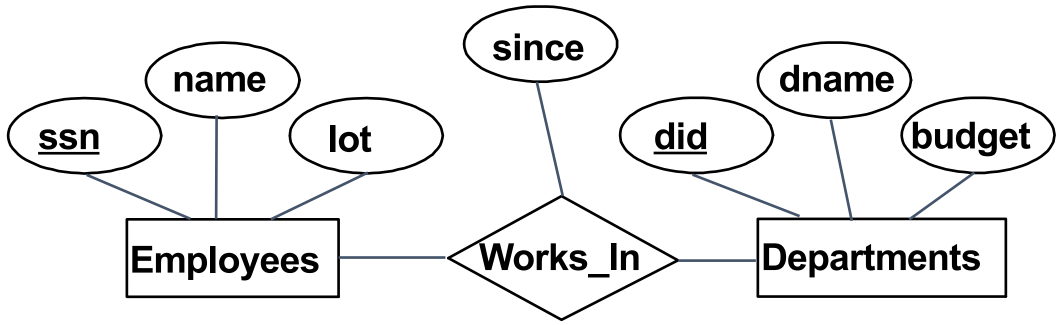
# ER Data Model

- *Entity:* Real-world object distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.
- *Entity Set*: A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a *key*.
  - Each attribute has a *domain*.
  - Permit combined or multi-valued attribute



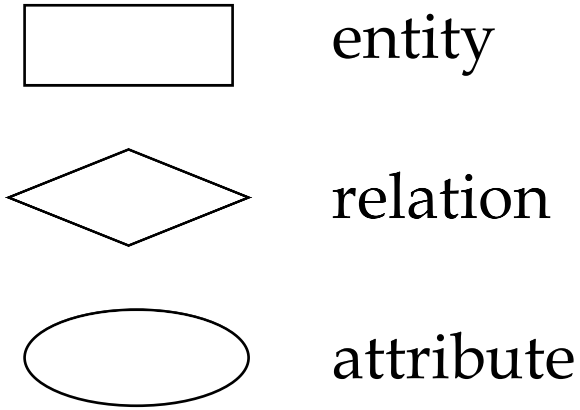
# Relationship

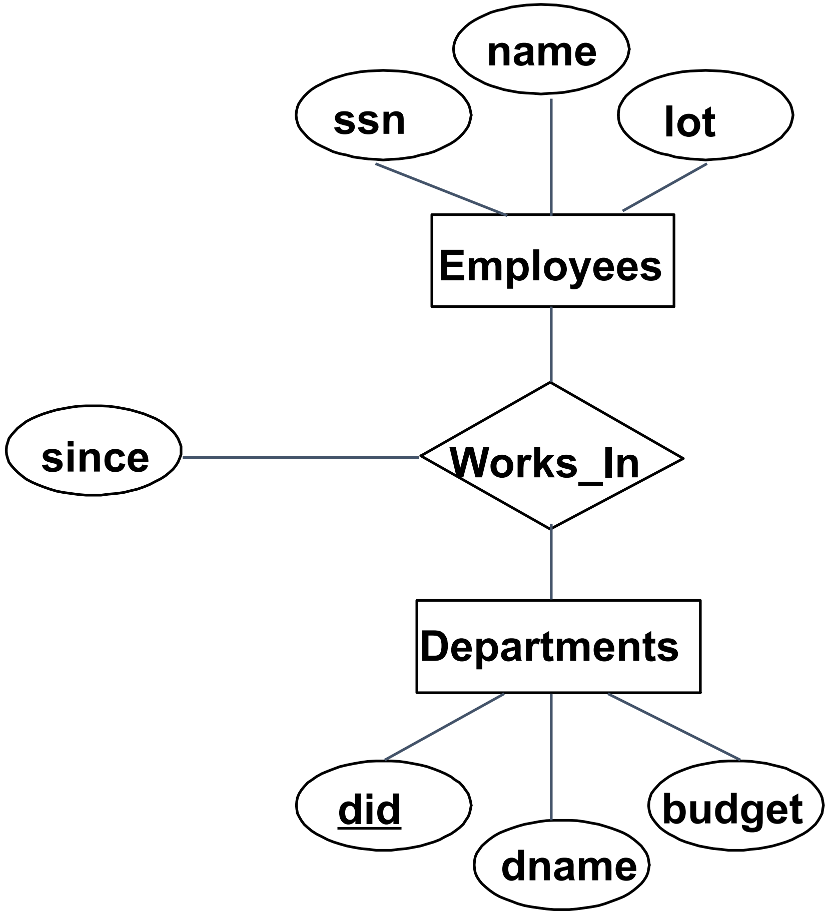
- *Relationship*: Association among two or more entities. E.g., Attishoo *works in* Pharmacy department.
  - Relationship can have attributes
- *Relationship Set*: Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets $E_1$ ... $E_n$; each relationship in R involves entities $e_1$, ..., $e_n$
  - Same entity set could participate in different relationship sets, or in different "roles" in same set.



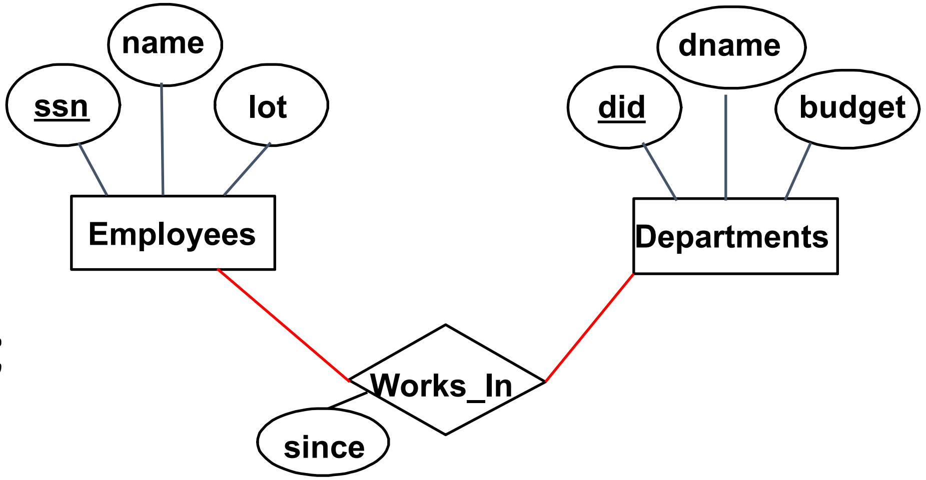
# ER Diagram

- Concept model: entity—relationship, be independent of practical DBMS.
- Legend:

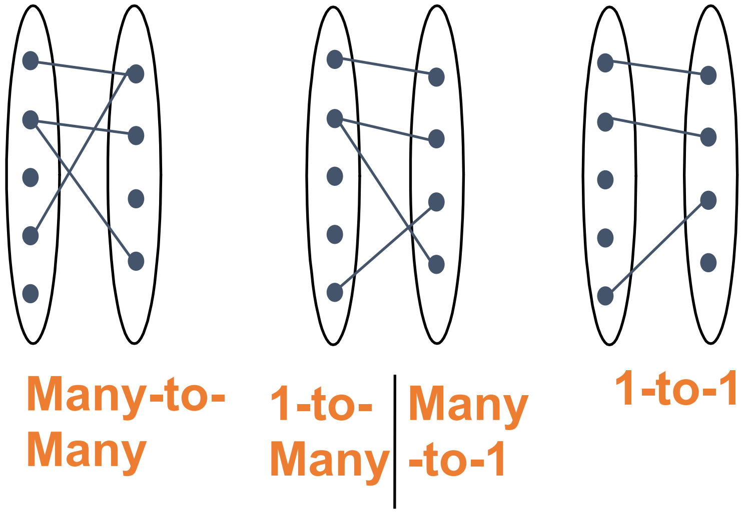


# Key Constraints

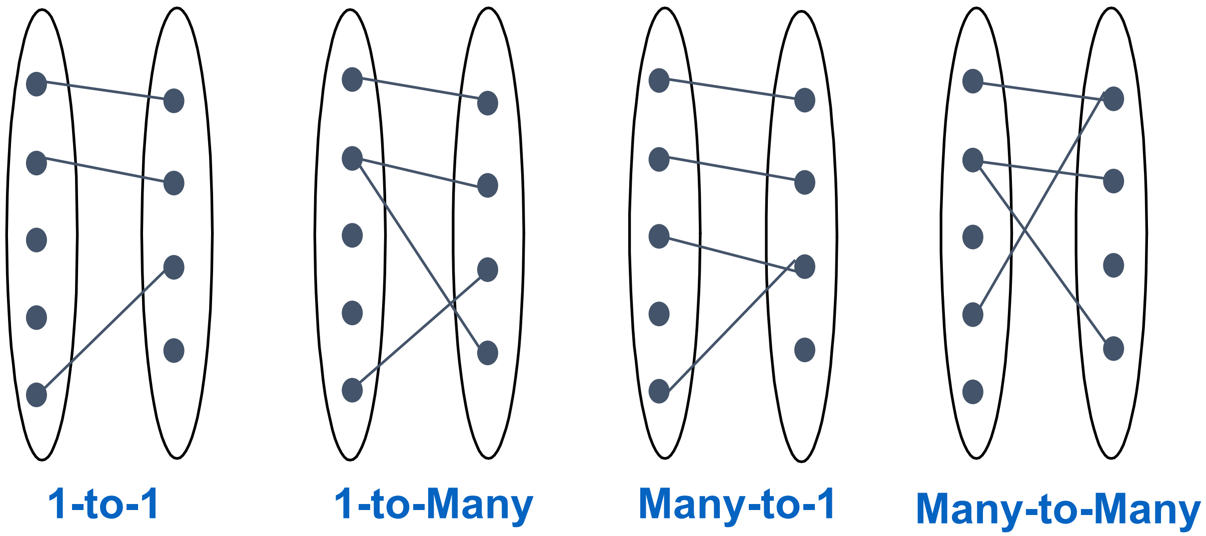



- An employee can work in **many** departments; a dept can have **many** employees.
- In contrast, each dept has **at most one** manager, according to the *key constraint* on **Department** in the **Manages** relationship set.
- A **key constraint** gives a 1-to-many relationship.



# Cardinality Ratio Constraints

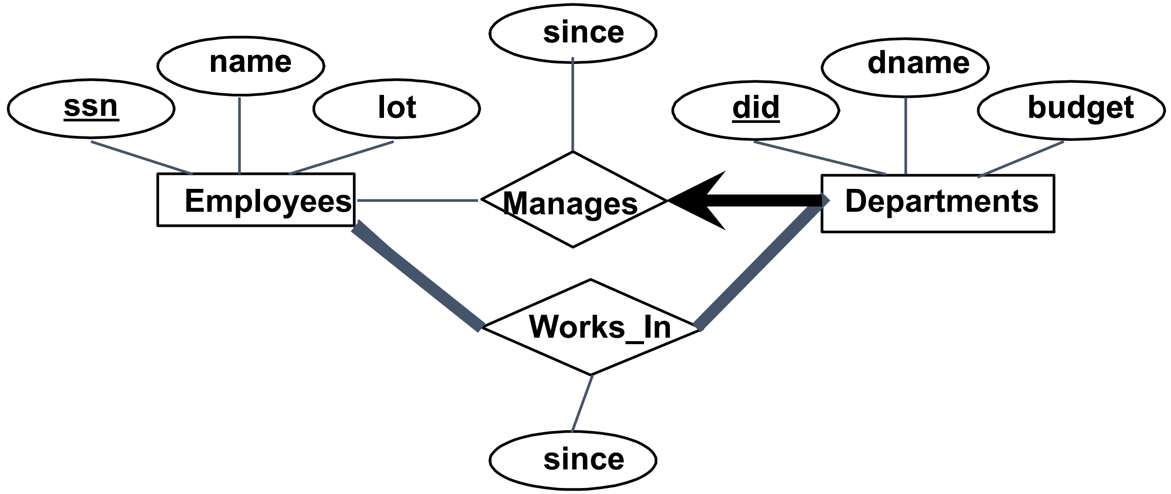
- Relationships can be distinguished as 1:1, 1:N, and M:N. This is called cardinality ratio constraints.




- For example: an employee can work in many departments; a dept can have many employees. This M:N. In contrast, each dept has at most one manager and one employee can only be manager of one dept, then this is 1:1.

# Participation Constraints

- Does every employee work in a department?
- If so: a **participation constraint**
  - participation of Employees in Works_In is total (vs. partial)
  - What if every department has an employee working in it?
- Basically means **at least one.**



# Participation Constraints

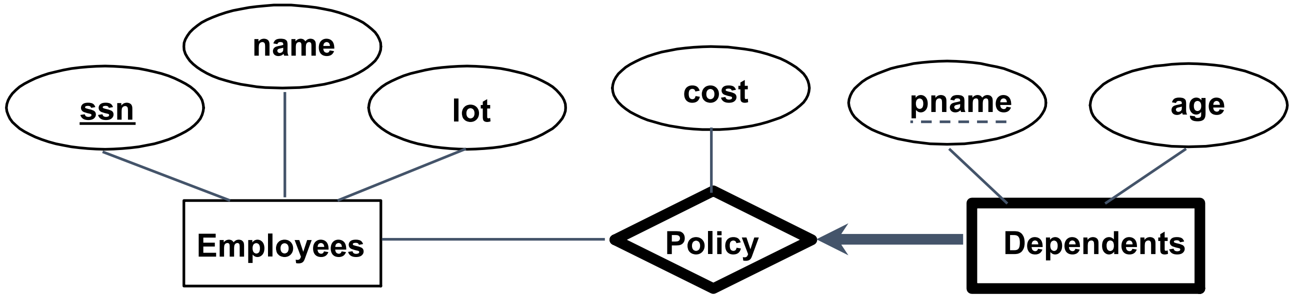
- We can further specify the minimal and max number an entity participates a relationship. This is called participation constraints.
- If a department must have a manager, then we say the Departments is *total participation* in Manages relationship (*vs. partial*). The minimal participating degree of Departments is 1.
- Another example: in the selected course relationship between Students and Courses, if we specify every student must select at least 3 courses and at most 6 courses, the participating degree of Students is said to be (3,6).

# Advanced Topics of ER Model

- Weak entity
- Specialization and Generalization
  - Similar as inheriting in Object-Oriented data model
- Aggregation
  - Allows us to treat a relationship set as an entity set   for purposes of participation in (other) relationships
- Category
  - Allow us to express an entity set consists of different types of entities. That is, a hybrid entity set.

# Weak Entities

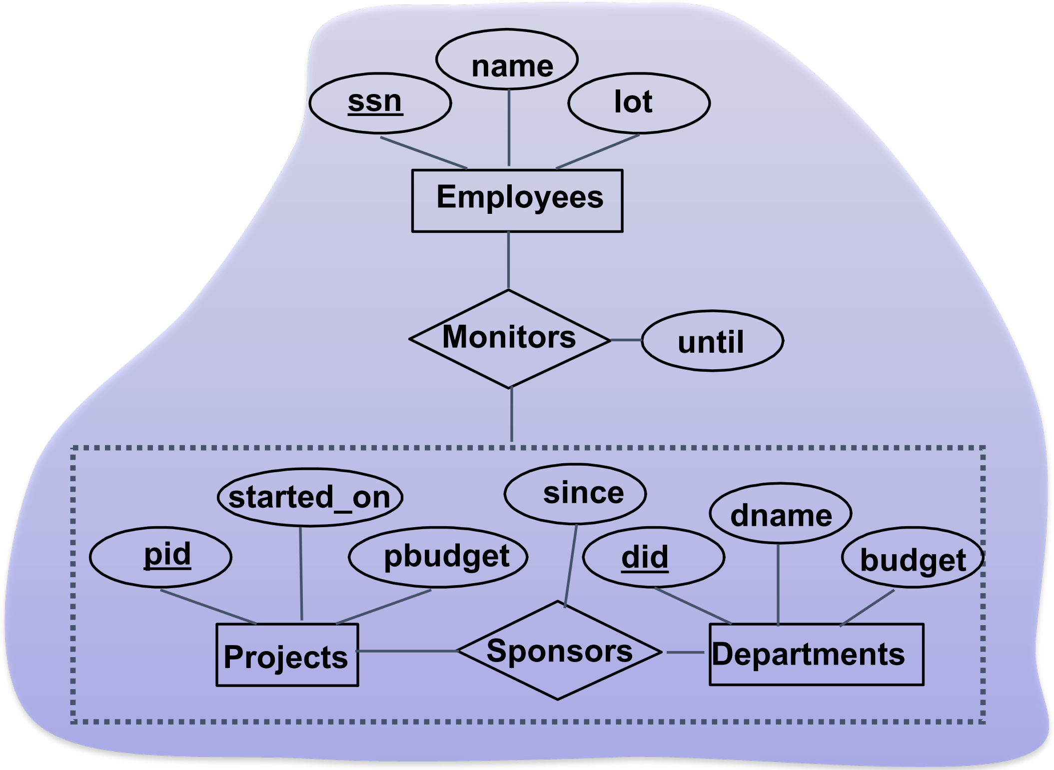
- A **weak entity** can be identified uniquely only by considering the primary key of another (owner) entity.
  - Owner entity set and weak entity set must participate in a one-to-many relationship set (one owner, many weak entities).
  - Weak entity set must have total participation in this identifying relationship set.




- Weak entities have only a “partial key” (dashed underline)

# Aggregation

- Allows relationships to have relationships.



# 2.5 Object-Oriented Data Model

- The shortage of relational data model
- Break through 1NF
- Object-Oriented analysis and programming
- Requirement of objects’ permanent store
- Object-Relation DBMS
- Native (pure) Object-Oriented DBMS

# 2.6 Other Data Models

- Logic-based data model (Deductive DBMS)
  - Extend the query function of DBMS (especially recursive query function)
  - Promote the deductive ability of DBMS
- Temporal data model
- Spatial data model
- XML data model
  - Store data on internet
  - Common data exchange standard
  - Information systems integration
  - Expression of semi-structured data
  - … …
- Others

# 2.7 Summary

- Data model is the core of a DBMS
- A data model is a methodology to simulate real world in database
- In fact, every kind of DBMS has implemented a data model

- If there will be a data model which can substitute relational model and become popular data model, just as relational model substituted hierarchical and network model 50 years ago **???**