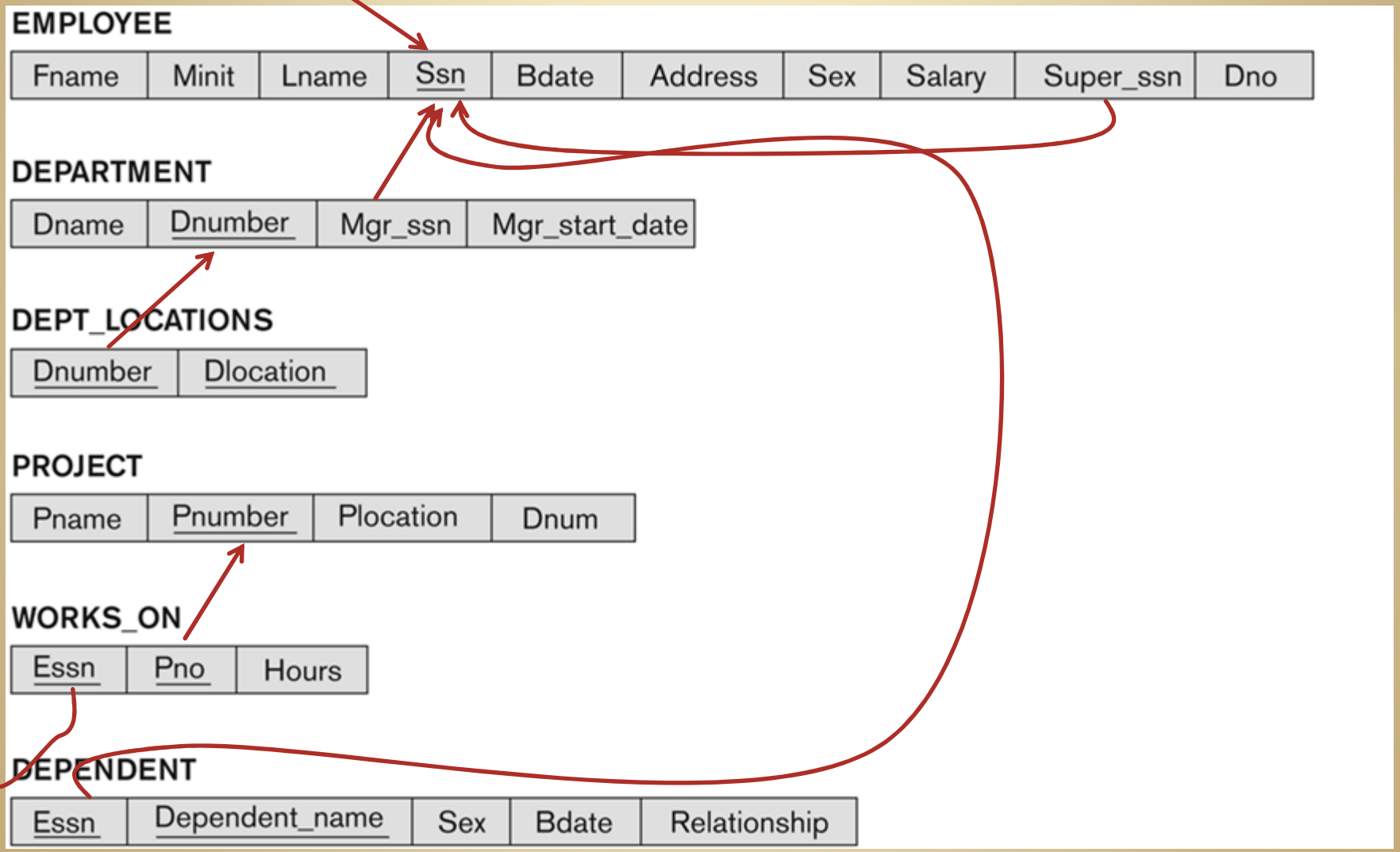
COMP163

Database Management Systems

Lecture 6 – Sections 6.1 and 6.3
The Relational Algebra

Example Schema



REVIEW

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

Example State

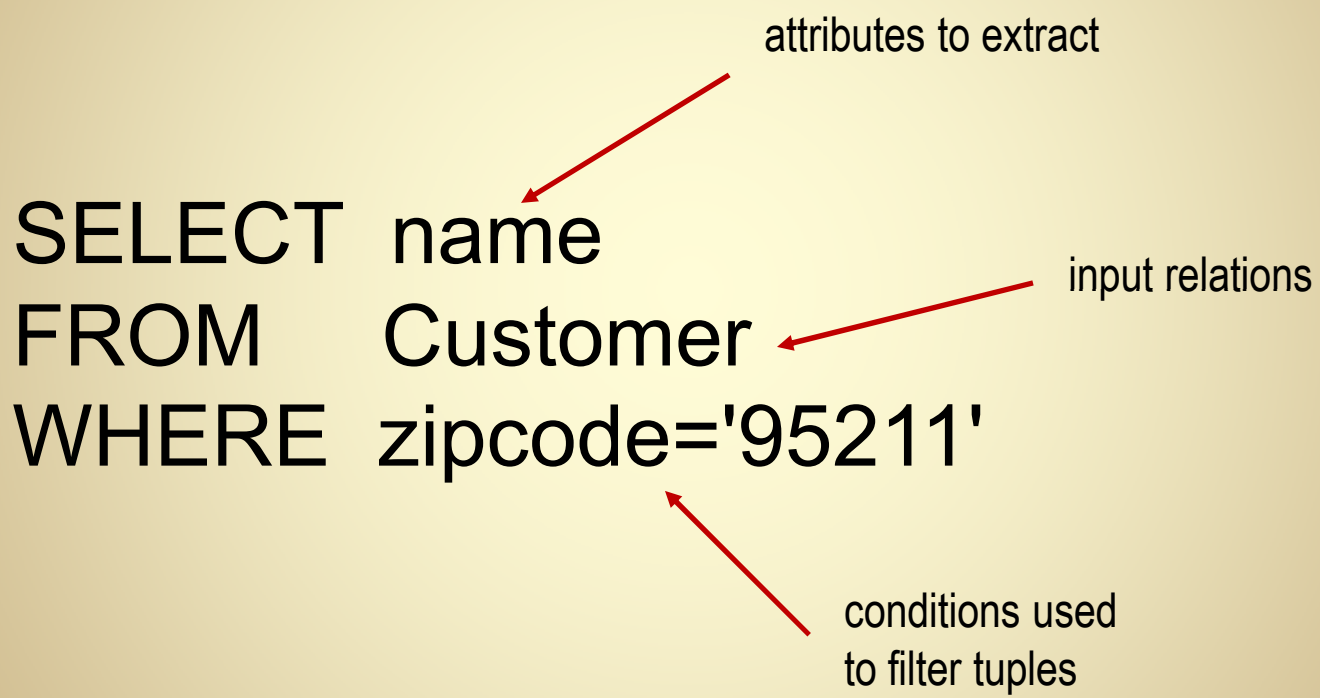
State for in-class exercise on constraint violations

REVIEW

Queries

- Information is extracted from a database by writing *queries*
  - query inputs: relations
  - query result: relation
- queries are *read-only* operations
- The SQL command for queries `select`

Example SQL Query



The Relational Algebra

- The ***relational algebra*** defines mathematical operations on relations
  - provides a solid theory for dealing with queries
- These operations define the meaning of SQL queries
  - *think* in terms of the algebra operators
  - *translate* the algebra to SQL syntax
- Query optimization:
  - SQL queries are translated to expression trees containing algebraic operators
  - expression trees are reorganized to optimize the order of operations

Relational Algebra Operations

- set theoretic:
  - **union, intersection, difference, cross-product**
  - usual mathematical meaning
- relational database unique:
  - **select, project**: extraction from a single relation
  - **join**: combining two relations

The SELECT Operation

- SELECT extracts tuples from a relation
  - result has same relation schema as operand
- SELECT requires a *selection condition*
  - selection condition is a boolean expression to filter tuple values
- Syntax:

$$\sigma_{\text{<selection condition>}}(R)$$

- Selection condition may contain

AND, OR, NOT, **=**, **<**, ≤, **>**, ≥, ≠

SELECT Examples

r_2(STORESTOCK) = { < "S002", "I065", 120 >, < "S333", "I954", 198 >, < "S047", "I099", 267 >, < "S047", "I954", 300 > }

$\sigma_{\text{StoreId = "S047"}}$ (STORESTOCK)

{ < "S047", "I099", 267 >, < "S047", "I954", 300 > }

$\sigma_{\text{quantity < 200}}$ (STORESTOCK)

{ < "S002", "I065", 120 >, < "S333", "I954", 198 > }

The PROJECT Operation

- PROJECT extracts attributes from a relation
  - result schema attributes are a subset of the operand schema
- PROJECT requires a *attribute list*
- Syntax:

$$\pi_{\text{<attribute list>}}(R)$$

- Duplicates are not kept in result
  - result is a relation, which is a set

PROJECT Examples

r_2(STORESTOCK) = { < "S002", "I065", 120 >, < "S333", "I954", 198 >, < "S047", "I099", 267 >, < "S047", "I954", 300 > }

$\pi_{\text{StoreId, Item}}$ (STORESTOCK)

{ < "S002", "I065" >, < "S333", "I954" >, < "S047", "I099" >, < "S047", "I954" > }

PROJECT Examples

r(STOCKITEM) =

{
< "I075", "Ice Cream", $1.49, false >,
< "I345", "Cupcakes", $1.99, false >,
< "I333", "Twinkies", $1.98, false >
}

$\pi_{\text{Description, Price}}$ (STOREITEM)

{
< "Ice Cream", $1.49 >,
< "Cupcakes", $1.99 >,
< "Twinkies", $1.98 >
}

Composing Operations

r(STOCKITEM) =

{ < "I075", "Ice Cream", $1.49, false >,
< "I345", "Cupcakes", $1.99, false >,
< "I333", "Twinkies", $1.98, false > }

$\pi_{\text{Price}} (\sigma_{\text{Description="Ice Cream"}} (\text{STOREITEM}))$

SELECT result: { < "I075", "Ice Cream", $1.49, false > }

PROJECT result: { < $1.49 > }

The JOIN Operation

- JOIN combines tuples from two tables based on values of related attributes (usually a FK)
- JOIN requires a join condition
  - boolean expression comparing attributes from each operand
- Syntax:

$$R \bowtie_{\text{<join condition>}} S$$

- The join condition may contain

AND, **=**, **<**, ≤, **>**, ≥, ≠

JOIN Examples

STORESTOCK(StoreId, Item, Quantity)
STOCKITEM(ItemId, Description, Price, Taxable)

r(STOCKITEM) = { < "I075", "Ice Cream", $1.49, false >, < "I345", "Cupcakes", $1.99, false >, < "I333", "Twinkies", $1.98, false > }

r(STORESTOCK) = { < "S002", "I075", 120 >, < "S047", "I333", 267 > }

$$\text{STORESTOCK} \bowtie_{\text{<Item = ItemId>}} \text{STOCKITEM}$$

JOIN Examples

RESULT(StoreId, Item, Quantity, ItemId, Description, Price, Taxable)

r(RESULT) =

{ < "S002", "I075", 120, "I075", "Ice Cream", $1.49, false >,
< "S047", "I333", 267, "I333", "Twinkies", $1.98, false > }

$$\text{STORESTOCK} \bowtie_{\text{<Item = ItemId>}} \text{STOCKITEM}$$

Writing Queries

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	1200

Query:
Get the Description and Price
for all Items stocked
by Store S002

STOCKITEM

ItemId	Description	Price	Taxable
I075	Ice Cream	$1.49	FALSE
I345	Cup Cakes	$1.99	FALSE
I333	Twinkies	$1.98	FALSE

result only
includes tuples
with
certain ItemIds

result has these attributes

Writing Queries

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	1200

STOCKITEM

ItemId	Description	Price	Taxable
I075	Ice Cream	$1.49	FALSE
I345	Cup Cakes	$1.99	FALSE
I333	Twinkies	$1.98	FALSE

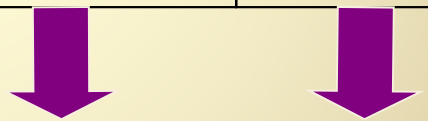
Query:
Get the Description and Price
for all Items stocked
by Store S002

We need a join
to merge data
across relations

Writing Queries

STORESTOCK $\bowtie_{\text{<Item = ItemId>}}$ STOCKITEM

StoreId	Item	Quantity	ItemId	Description	Price	Taxable
S002	I075	120	I075	Ice Cream	$1.49	FALSE
S047	I333	267	I333	Twinkies	$1.98	FALSE
S002	I333	1200	I333	Twinkies	$1.98	FALSE



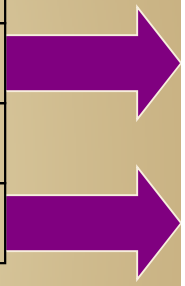
Query:
Get the Description and Price
for all Items stocked
by Store S002

Now we can select and project
to extract the information we want

Writing Queries

$R1 = STORESTOCK \bowtie_{<Item = ItemId>} STOCKITEM$

StoreId	Item	Quantity	ItemId	Description	Price	Taxable
S002	I075	120	I075	Ice Cream	$1.49	FALSE
S047	I333	267	I333	Twinkies	$1.98	FALSE
S002	I333	1200	I333	Twinkies	$1.98	FALSE



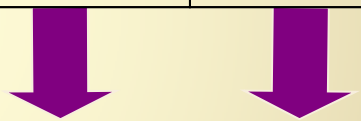
$R2 = \sigma_{StoreId = "S002"} (R1)$

StoreId	Item	Quantity	ItemId	Description	Price	Taxable
S002	I075	120	I075	Ice Cream	$1.49	FALSE
S002	I333	1200	I333	Twinkies	$1.98	FALSE

Writing Queries

$R2 = \sigma_{StoreId = "S002"} (R1)$

StoreId	Item	Quantity	ItemId	Description	Price	Taxable
S002	I075	120	I075	Ice Cream	$1.49	FALSE
S002	I333	1200	I333	Twinkies	$1.98	FALSE



$R3 = \pi_{<Description, Price>} (R2)$

Description	Price
Ice Cream	$1.49
Twinkies	$1.98

Composing Queries

Since the operands and results of every query are relations, we can compose or chain queries.

$$R1 = STORESTOCK \bowtie_{<Item = ItemId>} STOCKITEM$$

$$R2 = \sigma_{StoreId = "S002"} (R1)$$

$$R3 = \pi_{<Description, Price>} (R2)$$

$$\pi_{<Description, Price>}(\sigma_{StoreId = "S002"} (STORESTOCK \bowtie_{<Item = ItemId>} STOCKITEM))$$

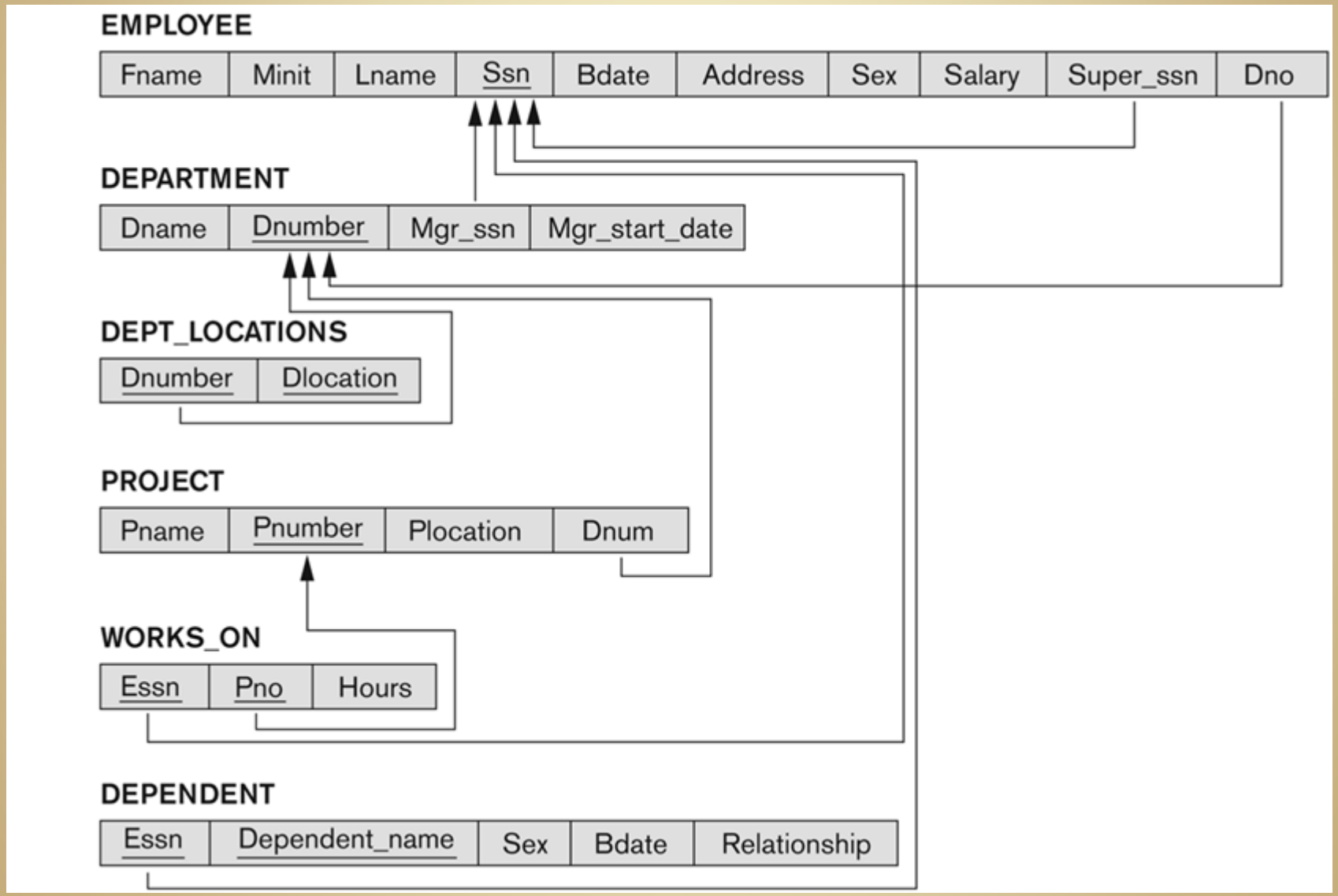
SQL Queries

Relational algebra queries are easily translated to SQL queries (more on this later)

$$\pi_{<\text{Description, Price}>}(\sigma_{\text{StoreId = "S002"}} (\text{STORESTOCK} \bowtie_{<\text{Item = ItemId}>} \text{STOCKITEM}))$$

```
SELECT Description, Price
FROM    STORESTOCK, STOCKITEM
WHERE StoreId='S002'
     AND STORESTOCK.Item = STOCKITEM.ItemId
```

EXERCISE 1: Schema



EXERCISE 1: Queries

Express the following queries in the algebra:

1. First and last name of employees who have no supervisor.
2. First and last name of employees supervised by Franklin Wong.
3. Last name of employees who have dependents.
4. Last name of employees who have daughters.
5. Last name of employees in department 5 who work more than 10 hours/week on ProductX.
6. Last name of supervisors of employees in department 5 who work more than 10 hours/week on ProductX.

Section 6.2

Operations from Set Theory

Set Operations

- UNION: R ∪ S
  - all tuples in either R or S
- INTERSECTION: R ∩ S
  - all tuples in both R and S
- DIFFERENCE: R - S
  - all tuples in R but not in S
- CROSS-PRODUCT or CARTESIAN PRODUCT: R × S
  - pair all tuples in R with all tuples in S

∪ ∩ −

operands must be *union compatable* (same attribute types)

UNION: example

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	267

WAREHOUSESTOCK

StoreId	Item	Quantity
W998	I075	1200
W087	I001	5000
W222	I188	11500
W023	I075	300

STORESTOCK ∪ WAREHOUSESTOCK

Id	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	267
W998	I075	1200
W087	I001	5000
W222	I188	11500
W023	I075	300

INTERSECTION: example

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	267

WAREHOUSESTOCK

StoreId	Item	Quantity
W998	I075	1200
W087	I001	5000
W222	I188	11500
W023	I075	300

$\pi_{ItemID}(STORESTOCK) \cap \pi_{Item}(WAREHOUSESTOCK)$

Item
I075
I333

$\cap$

Item
I075
I001
I188

$=$

Item
I075

DIFFERENCE: example

STOCKITEM

ItemId	Description	Price	Taxable
I075	Ice Cream	$1.49	FALSE
I345	Cup Cakes	$1.99	FALSE
I333	Twinkies	$1.98	FALSE

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267
S002	I333	267

$$\pi_{ItemID}(STOCKITEM) - \pi_{Item}(STORESTOCK)$$

ItemId
I075
I345
I333

-

Item
I075
I333

=

Item
I345

CROSS PRODUCT: example

STOCKITEM

ItemId	Description	Price	Taxable
I075	Ice Cream	$1.49	FALSE
I345	Cup Cakes	$1.99	FALSE
I333	Twinkies	$1.98	FALSE

STORESTOCK

StoreId	Item	Quantity
S002	I075	120
S047	I333	267

STOCKITEM × STORESTOCK

ItemId	Description	Price	Taxable	StoreId	Item	Quantity
I075	Ice Cream	$1.49	FALSE	S002	I075	120
I345	Cup Cakes	$1.99	FALSE	S002	I075	120
I333	Twinkies	$1.98	FALSE	S002	I075	120
I075	Ice Cream	$1.49	FALSE	S047	I333	267
I345	Cup Cakes	$1.99	FALSE	S047	I333	267
I333	Twinkies	$1.98	FALSE	S047	I333	267

CROSS PRODUCT

- CROSS PRODUCT is also called CROSS JOIN
  - a JOIN with no join condition
- XPROD brings together all possible information from two relations
- XPROD can result in very large relations
  - if R has i tuples and S has j tuples, then R × S has i*j tuples
- In real applications, avoid XPROD whenever possible
  - Database equivalent of a brute-force nested loop
  - massive unnecessary memory usage

JOIN = XPROD and SELECT

STORESTOCK $\bowtie_{\text{<Item = ItemId>}}$ STOCKITEM

ItemId	Description	Price	Taxable	StoreId	Item	Quantity
I075	Ice Cream	$1.49	FALSE	S002	I075	120
I333	Twinkies	$1.98	FALSE	S047	I333	267

$\sigma_{\text{ItemId=Item}}$ (STOCKITEM × STORESTOCK)

ItemId	Description	Price	Taxable	StoreId	Item	Quantity
I075	Ice Cream	$1.49	FALSE	S002	I075	120
I345	Cup Cakes	$1.99	FALSE	S002	I075	120
I333	Twinkies	$1.98	FALSE	S002	I075	120
I075	Ice Cream	$1.49	FALSE	S047	I333	267
I345	Cup Cakes	$1.99	FALSE	S047	I333	267
I333	Twinkies	$1.98	FALSE	S047	I333	267

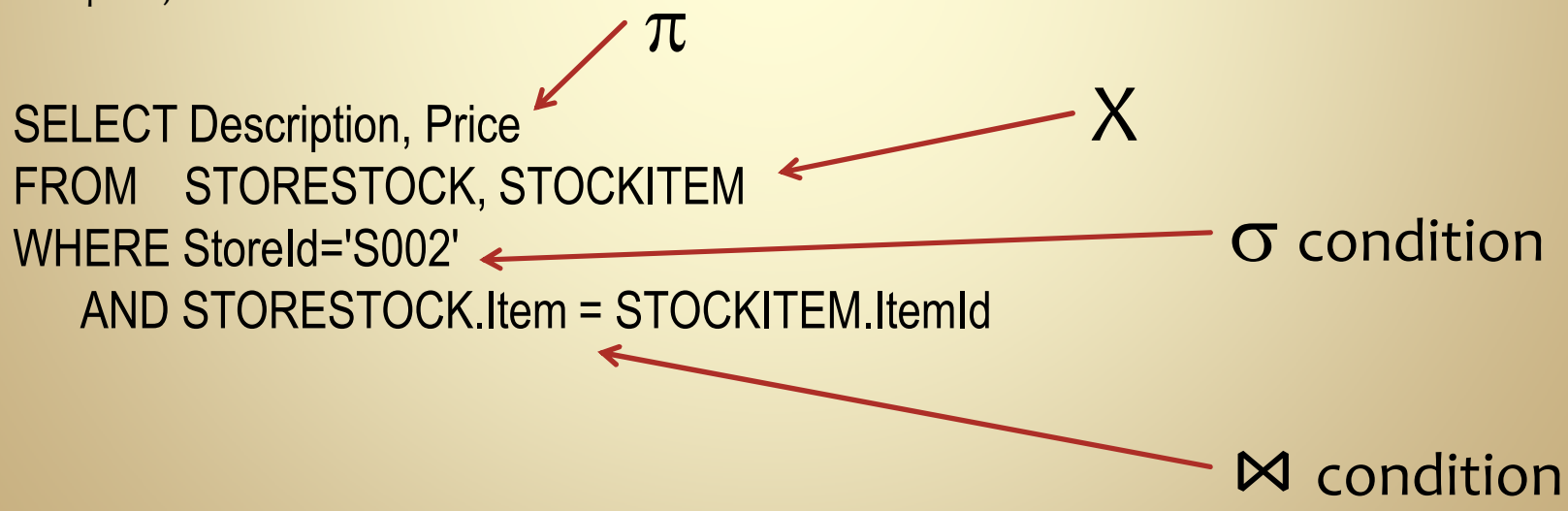
Declarative Programming

Queries are **declarative**, not **procedural**.

These queries specify the information we want, in terms of mathematical operations … but not how to compute it.

Does not say "compute a cross-product, then select …", rather give me something "equivalent to cross-product, select …"

$$\pi_{\text{<Description, Price>}}(\sigma_{\text{StoreId = "S002"}} (\text{STORESTOCK} \bowtie_{\text{<Item = ItemId>}} \text{STOCKITEM}))$$




```
SELECT Description, Price
FROM   STORESTOCK, STOCKITEM
WHERE StoreId='S002'
    AND STORESTOCK.Item = STOCKITEM.ItemId
```

Complete Set of Operations

- A *complete set* of operations
  is a subset of all operations that is sufficient to express all queries expressible by all operators
  - any operators outside a complete set can be expressed as combinations of operators in the complete set
- Complete Set: $\{ \sigma, \pi, \cup, -, \times \}$
- Operators outside a complete set are not *necessary*, but they are generally *convenient*
  - JOIN: $A \bowtie_C B = \sigma_C(A \times B)$
  - INTERSECTION: $A \cap B = (A \cup B) - ((A - B) \cup (B - A))$

Additional Operators

- There are some queries in commercial languages (i.e. SQL) that cannot be expressed in the basic relational algebra

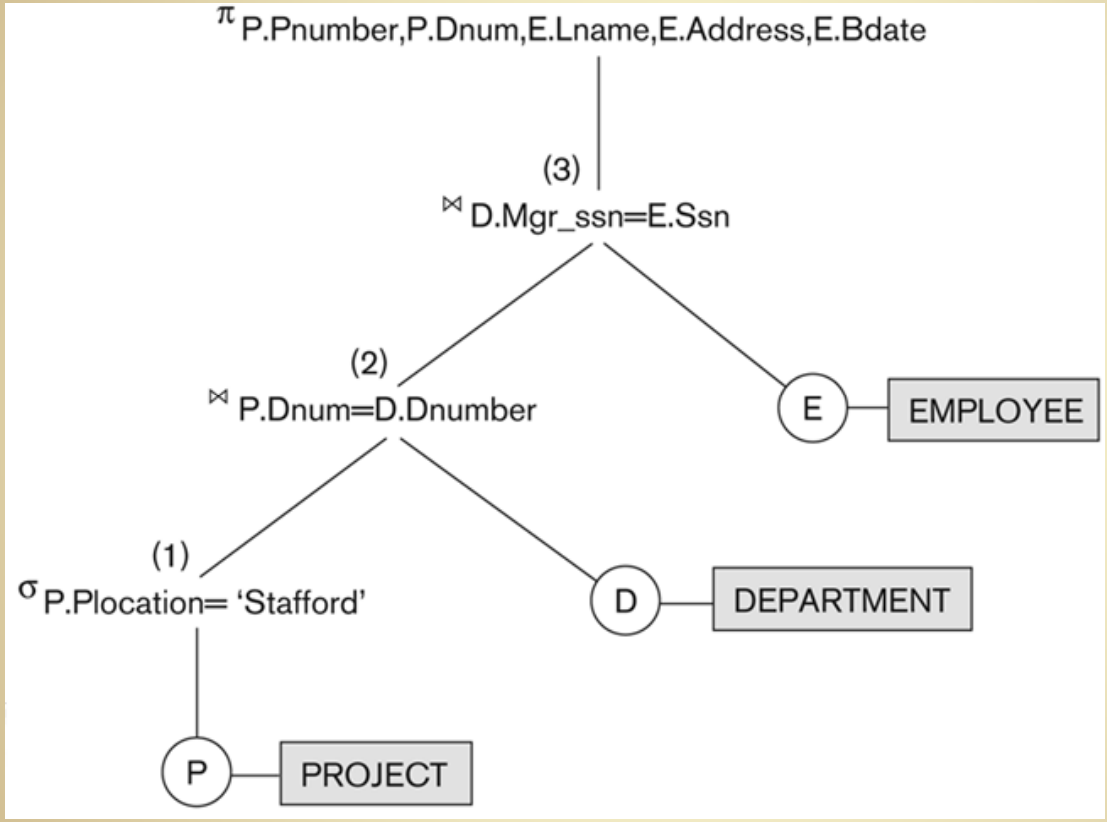
- A few additional operations handle most of these

  - **aggregation:** apply a function to a collection of values

  - **OUTER JOINS** and **OUTER UNIONS:** null values are used to keep data that would otherwise be discarded

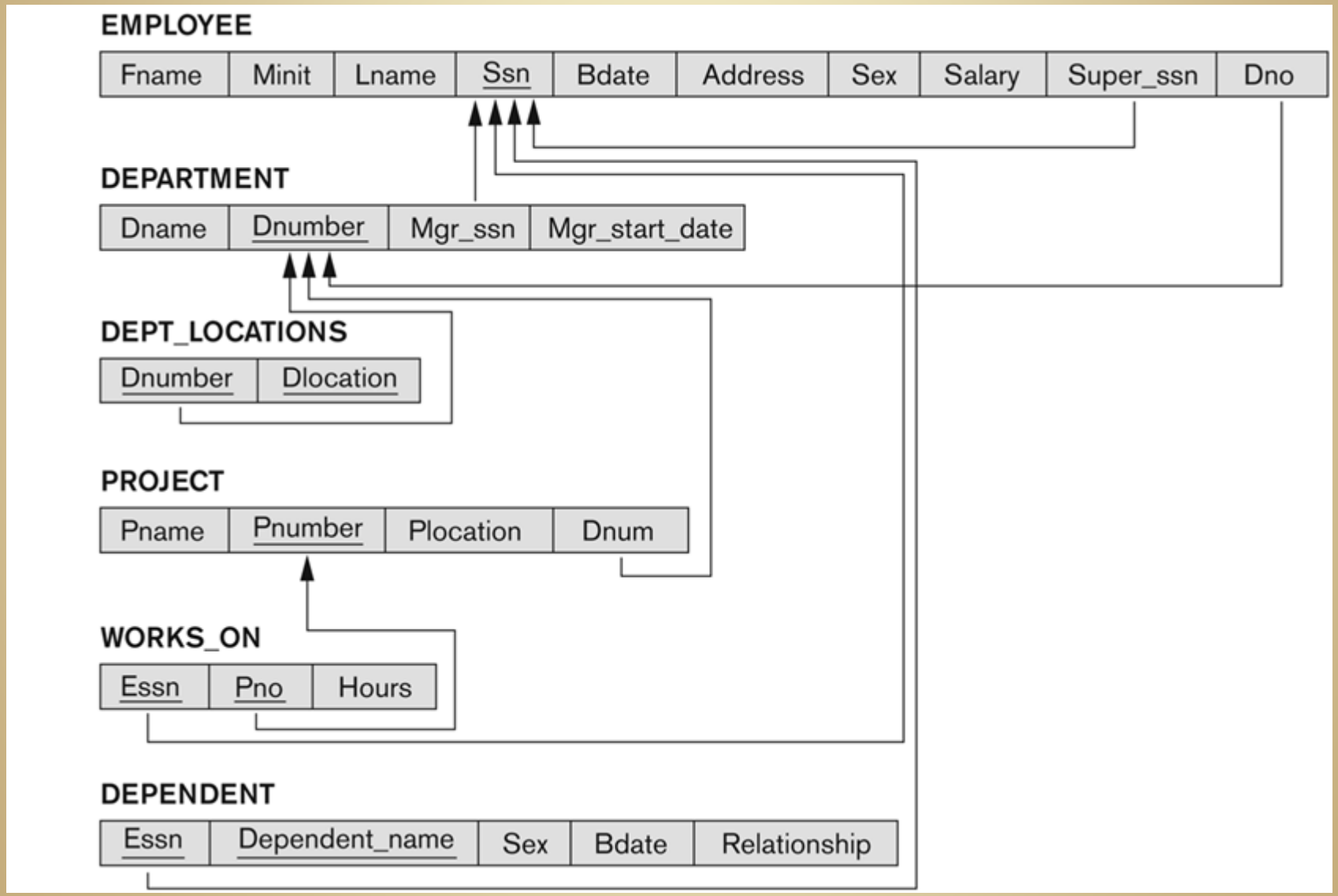
Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces all combinations of tuples from R_1 and R_2 that satisfy the join condition.	$R_1 \bowtie_{\text{<join condition>}} R_2$
EQUIJOIN	Produces all the combinations of tuples from R_1 and R_2 that satisfy a join condition with only equality comparisons.	$R_1 \bowtie_{\text{<join condition>}} R_2$, OR $R_1 \bowtie_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$
NATURAL JOIN	Same as EQUIJOIN except that the join attributes of R_2 are not included in the resulting relation; if the join attributes have the same names, they do not have to be specified at all.	$R_1 *_{\text{<join condition>}} R_2$, OR $R_1 *_{\text{(<join attributes 1>), (<join attributes 2>)}} R_2$ OR $R_1 * R_2$
UNION	Produces a relation that includes all the tuples in R_1 or R_2 or both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cup R_2$
INTERSECTION	Produces a relation that includes all the tuples in both R_1 and R_2; R_1 and R_2 must be union compatible.	$R_1 \cap R_2$
DIFFERENCE	Produces a relation that includes all the tuples in R_1 that are not in R_2; R_1 and R_2 must be union compatible.	$R_1 - R_2$
CARTESIAN PRODUCT	Produces a relation that has the attributes of R_1 and R_2 and includes as tuples all possible combinations of tuples from R_1 and R_2.	$R_1 \times R_2$
DIVISION	Produces a relation $R(X)$ that includes all tuples $t[X]$ in $R_1(Z)$ that appear in R_1 in combination with every tuple from $R_2(Y)$, where $Z = X \cup Y$.	$R_1(Z) \div R_2(Y)$

Query Trees



Query trees are representations of queries that can be manipulated by **query optimizers**, according to mathematical properties of the operators. (Chapter 15)

EXERCISE 2: Schema



EXERCISE 2: Queries

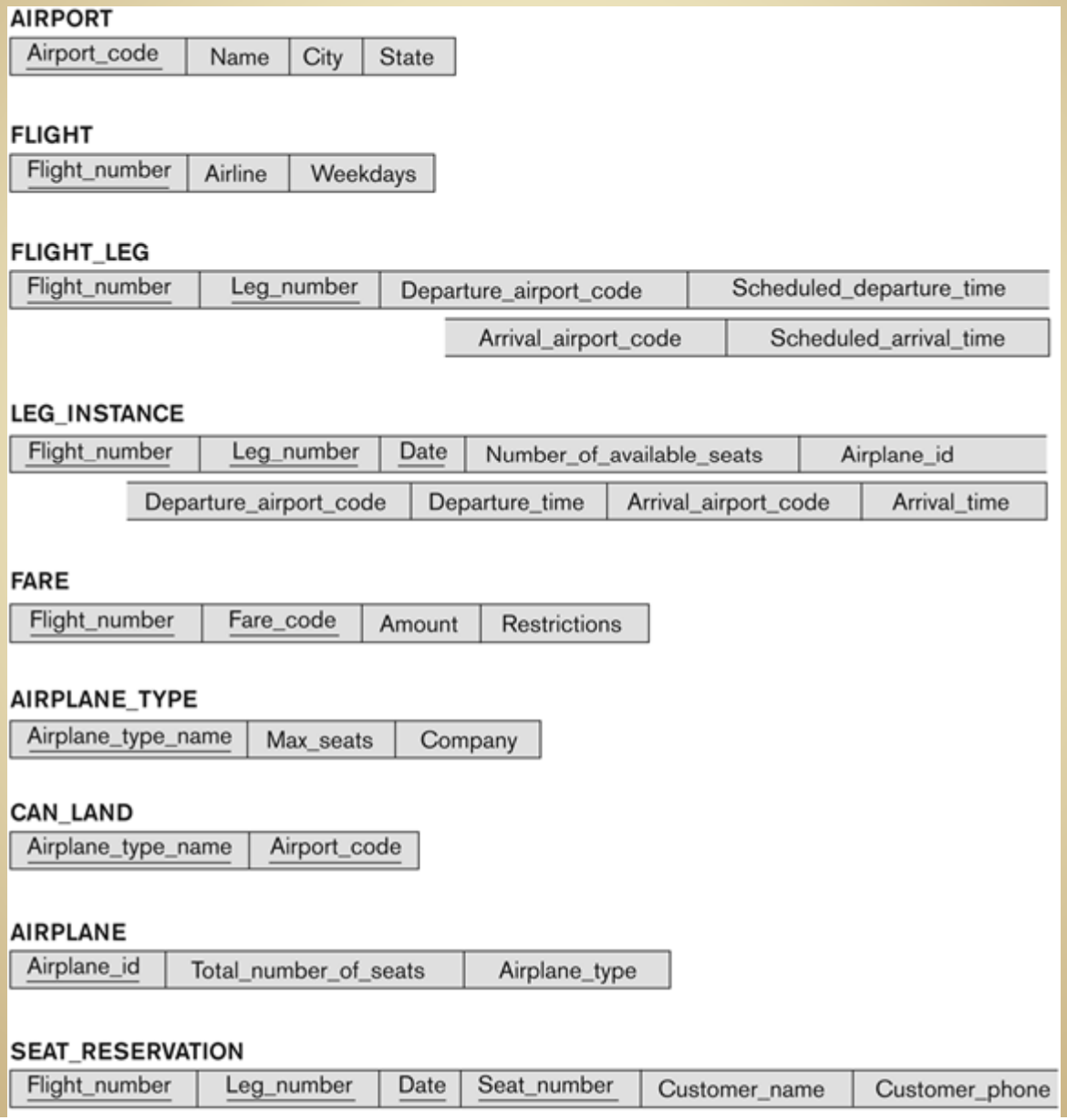
1. First and last names
   of all department managers.

2. Salaries of all employees
   who have worked on the Reorganization project.

3. SSN of all employees who have worked on a project
   that is controlled by a department different
   than the department that they are assigned to.

4. Last name of all employees who are not married.

Exercise 3: Schema



EXERCISE 3: Queries

1. List all airplane types that can land at any airport in San Francisco.
2. List the ids and number of seats for all airplanes that can land at any airport in Chicago.
3. List the name and phone number of all customers with a seat reserved on a flight that leaves Chicago O'Hara airport (ORD) on October 31, 2010.
4. List all airlines that have seats available for flights leaving Los Angeles (LAX) on September 25, 2010.
5. List all airlines that operate at San Jose International Airport (SJC).

RENAME

- *Rename* the attributes of a relation or change the relation name
- The general RENAME operation ρ:
  - $\rho_{S(B1, B2, \ldots, Bn)}(R)$ changes both:
    - the relation name to S, *and*
    - the column (attribute) names to B1, B1, …..Bn
  - $\rho_S(R)$ changes:
    - the *relation name* only to S
  - $\rho_{(B1, B2, \ldots, Bn)}(R)$ changes:
    - the *column (attribute) names* only to B1, B1, …..Bn

RENAME (shorthand)

- *shorthand* for renaming attributes :

$R1 \leftarrow \pi_{FNAME, LNAME, SALARY} (DEP5\_EMPS)$

R1 has same attribute names as DEP5_EMPS

$R2 \leftarrow \rho_{(F,L,S)} (\pi_{FNAME, LNAME, SALARY} (DEP5\_EMPS))$

R2 has attributes renamed to F, L and S

$R3 (F,L,S) \leftarrow \pi_{FNAME, LNAME, SALARY} (DEP5\_EMPS)$

R3 also has attributes renamed to F, L and S

Aggregate Functions

- Aggregate functions apply mathematical operations to a collection of values.
- Examples:
  - compute average salary for all employees
  - compute maximum salary for all managers
  - count number of airports
- Common aggregate functions:
  - SUM
  - AVERAGE
  - MAX
  - MIN
  - COUNT

Aggregate Function

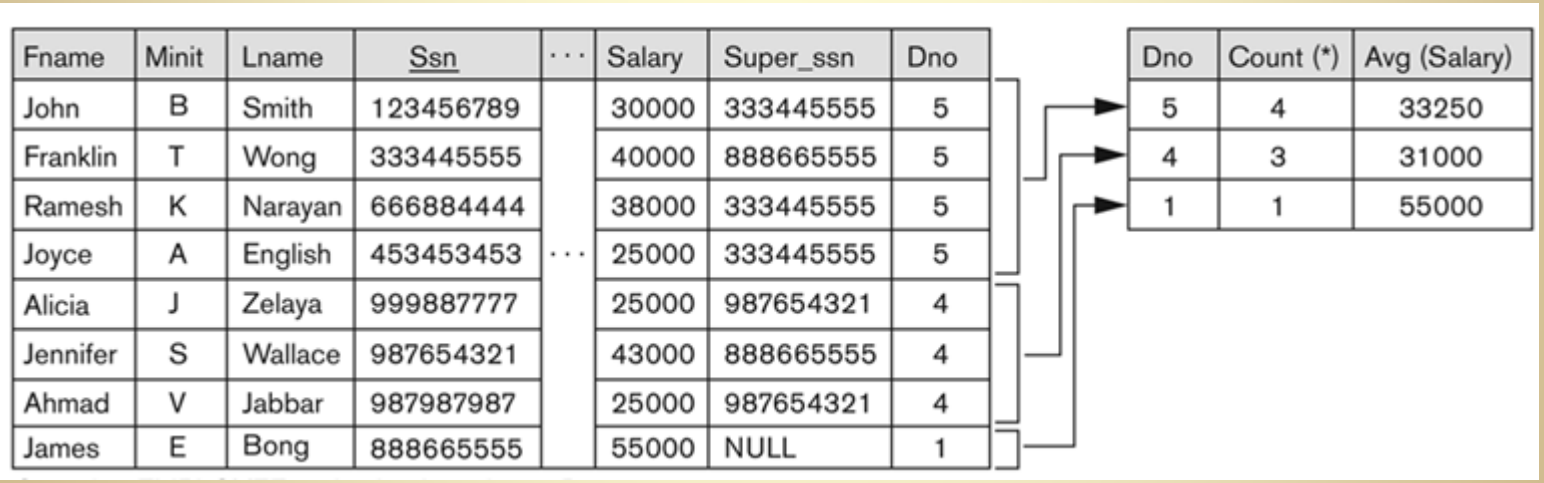
- Symbol for aggregate functions is $\mathcal{F}$
  - $\mathcal{F}_{\text{MAX Salary}}$ (EMPLOYEE)
    retrieves the maximum salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{MIN Salary}}$ (EMPLOYEE)
    retrieves the minimum Salary value from the EMPLOYEE relation
  - $\mathcal{F}_{\text{SUM Salary}}$ (EMPLOYEE)
    retrieves the sum of the Salary from the EMPLOYEE relation
  - $\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
    computes the number of employees and their average salary

Grouping with Aggregation

- Grouping can be combined with Aggregate Functions
- Example:
  - For each department, retrieve the DNO, COUNT SSN, and AVERAGE SALARY
  - $_{\text{DNO}}\mathcal{F}_{\text{COUNT SSN, AVERAGE Salary}}$ (EMPLOYEE)
- Group employees by DNO,
  then compute aggregate functions on each group

Grouping with Aggregation

DNO $\mathcal{F}$ COUNT SSN, AVERAGE Salary (EMPLOYEE)



Fname	Minit	Lname	Ssn	...	Salary	Super_ssn	Dno
John	B	Smith	123456789		30000	333445555	5
Franklin	T	Wong	333445555		40000	888665555	5
Ramesh	K	Narayan	666884444		38000	333445555	5
Joyce	A	English	453453453	...	25000	333445555	5
Alicia	J	Zelaya	999887777		25000	987654321	4
Jennifer	S	Wallace	987654321		43000	888665555	4
Ahmad	V	Jabbar	987987987		25000	987654321	4
James	E	Bong	888665555		55000	NULL	1

Dno	Count (*)	Avg (Salary)
5	4	33250
4	3	31000
1	1	55000

JOIN VARIANTS

- Equijoin
- Natural join
- Outer join
- Outer union

THETA JOIN

- Theta Join is the general case
  $$\text{STORESTOCK} \bowtie_{\langle \text{Item} = \text{ItemId} \rangle} \text{STOCKITEM}$$
  - The join condition is called *theta*
- *Join condition* can be any Boolean expression on the attributes of R and S
  - R.Ai<S.Bj AND (R.Ak=S.Bl OR R.Ap<S.Bq)
- Most join conditions involve one or more equality conditions, connected with AND
  - R.Ai=S.Bj AND R.Ak=S.Bl AND R.Ap=S.Bq

EQUIJOIN

- The most common use of join involves join conditions with *equality comparisons* only

- When the only comparison operator used is =, we call it an EQUIJOIN

  - In the result of an EQUIJOIN we always have one or more pairs of attributes (whose names need not be identical) that have identical values in every tuple.

NATURAL JOIN

- NATURAL JOIN removes the superfluous attributes in an EQUIJOIN
  - The standard definition of natural join requires that the two join attributes, or each pair of corresponding join attributes, *have the same name* in both relations
  - If this is not the case, a renaming operation is applied first.

- Operator for NATURAL JOIN is *

  R * S joins R and S with equality tests
  on all common attributes,
  then removes duplicate attributes

NATURAL JOIN

- $Q \leftarrow R(A,B,C,D) * S(C,D,E)$
  - The implicit join condition includes *each pair* of attributes with the same name, connected by AND
    - $R \bowtie_{R.C = S.C \text{ AND } R.D = S.D} S$
  - Result keeps only one attribute of each such pair:
    - Q(A,B,C,D,E)

OUTER JOIN

- THETA JOIN, NATURAL JOIN and EQUIJOIN, eliminate tuples without a *matching* (related) tuple
  - Tuples with null in the join attributes are also eliminated
  - This amounts to loss of information.

- OUTER JOINs keep the unmatched tuples and match them with NULLs
  - LEFT OUTER JOIN (=⋈): keep all tuples from left relation
  - RIGHT OUTER JOIN (⋈=): keep all tuples from right relation
  - FULL OUTER JOIN (=⋈=): keep all tuples from both relations

OUTER JOIN EXAMPLE

S

A	B
1	4
2	5
3	6

T

C	D
5	8
5	9
7	10

R1 = S $=\bowtie_{B=C}$ T (left join)

A	B	C	D
1	4	NULL	NULL
2	5	5	8
2	5	5	9
3	6	NULL	NULL

R2 = S $\bowtie=_{B=C}$ T (right join)

A	B	C	D
2	5	5	8
2	5	5	9
NULL	NULL	7	10

R3 = S $=\bowtie=_{B=C}$ T (full join)

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

notice that R3 = R1 U R2

OUTER UNION

- OUTER UNION computes
  the union of tuples from two relations
  when the relations are *not type compatible*

  - Tuples are included for all tuples from either relation that do not match a tuple in the other relation
  - Tuples are also included for pairs of tuples from each attribute that do match on common attributes
  - Remaining attribute values from both relations are kept
  - Missing values are replaced with NULL

OUTER UNION

- Example: OUTER UNION of
  STUDENT(Name, SSN, Department, Advisor) and
  INSTRUCTOR(Name, SSN, Department, Rank)

  - Tuples from the two relations are matched based on having the same combination of values of the shared attributes:
    Name, SSN, Department

  - The result relation will have the following attributes:

    STUDENT_OR_INSTRUCTOR (Name, SSN, Department, Advisor, Rank)

  - If a student is also an instructor, both Advisor and Rank will have a value; otherwise, one of these two attributes will be null

Recursive Closure

- Example:
  - Find all employees who work under (directly or indirectly) James Borg
- This operation is applied to a **recursive relationship.**
  - simplified schema:
    EMPLOYEE(name, ID, superID)
  - superId is the ID of an employees supervisor (may be NULL)

Recursive Closure

- Example:
  - Find all employees who work under Miro

EMPLOYEE

Name	ID	superID
Ang	1	5
Wassim	2	3
Iria	3	NULL
Jung	4	6
Miro	5	3
Sally	6	5

RECURSIVE CLOSURE

- Solution: Retrieve employees at each level and build solution with union

  $T1 = \pi_{ID}(\sigma_{Name=\text{“Miro”}} EMPLOYEE)$
  Repeat until T1 does not change:
  $T2 = \pi_{ID}(EMPLOYEE \bowtie_{EMPLOYEE.superID=T1.ID} T1)$
  $T1 = T1 \cup T2$

- cannot, in general, specify recursive closure without iteration (loops)
  - The SQL3 standard includes syntax for recursive closure.

DIVISION

- Example:
  - Retrieve names of employees who work on ***all*** projects that John Smith works on.

- $R = S \div T$

  $x \in R$, iff for every $y \in T$, $\langle x,y \rangle \in S$

- The tuples in R come from the tuples in S that match every tuple in T

DIVISION Example

R

A	B
a1	b1
a2	b1
a3	b1
a4	b1
a1	b2
a3	b2
a2	b3
a3	b3
a4	b3
a1	b4
a2	b4
a3	b4

S

A
a1
a2
a3

T

B
b1
b4

$T = R \div S$

T has all attributes of R that are not in S.

Find tuples in R that match every tuple in S.
Project the attributes that are not in S.

DIVISION Example

SSN_PNOS

Essn	Pno
123456789	1
123456789	2
666884444	3
453453453	1
453453453	2
333445555	2
333445555	3
333445555	10
333445555	20
999887777	30
999887777	10
987987987	10
987987987	30
987654321	30
987654321	20
888665555	20

SMITH_PNOS

Pno
1
2

SSNS

Ssn
123456789
453453453

SSNS = SSN_PNOS ÷ SMITH_PNOS

Retrieve names of employees who work on ***all*** projects that John Smith works on.

DIVISION

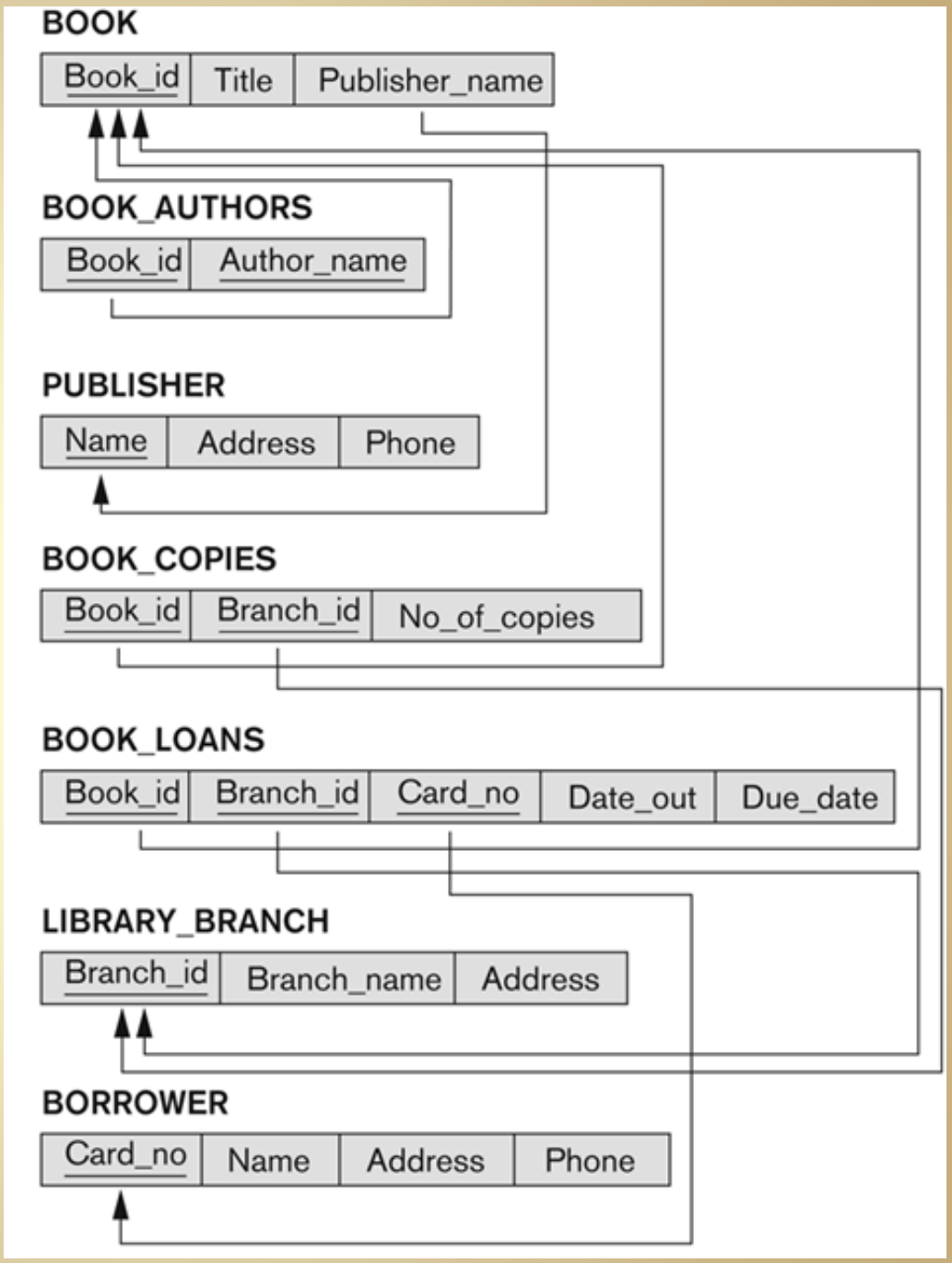
- R = S ÷ T can be computed as

$R1 = \pi_Y(S)$

$R2 = \pi_Y((T \times R1) - S)$

$R = R1 - R2$

EXERCISE 4: Schema



EXERCISE 4: Queries

1. Count the number of overdue books.

2. How many books by author Harry Crews are in the database?

3. Determine the number of library cards assigned to each borrower phone number.

4. Find names of all borrowers who do not have any book loans.

5. Do any library branches have every book?