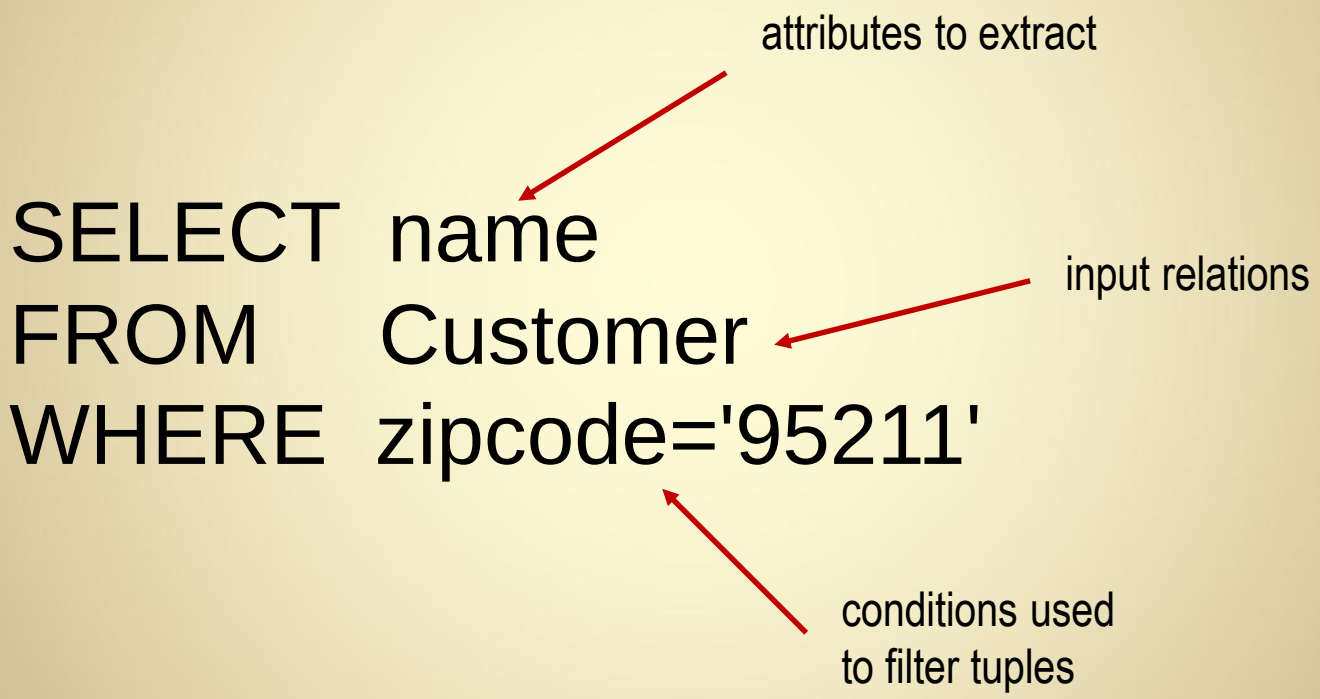
Advanced Database Management Systems

Lecture 6 – Chapter 6
The Relational Algebra

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 > }

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

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 > }

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

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

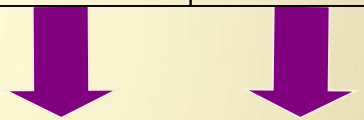
$R2 = \sigma_{StoreId = "S002"} (R1)$

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.

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

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

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

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

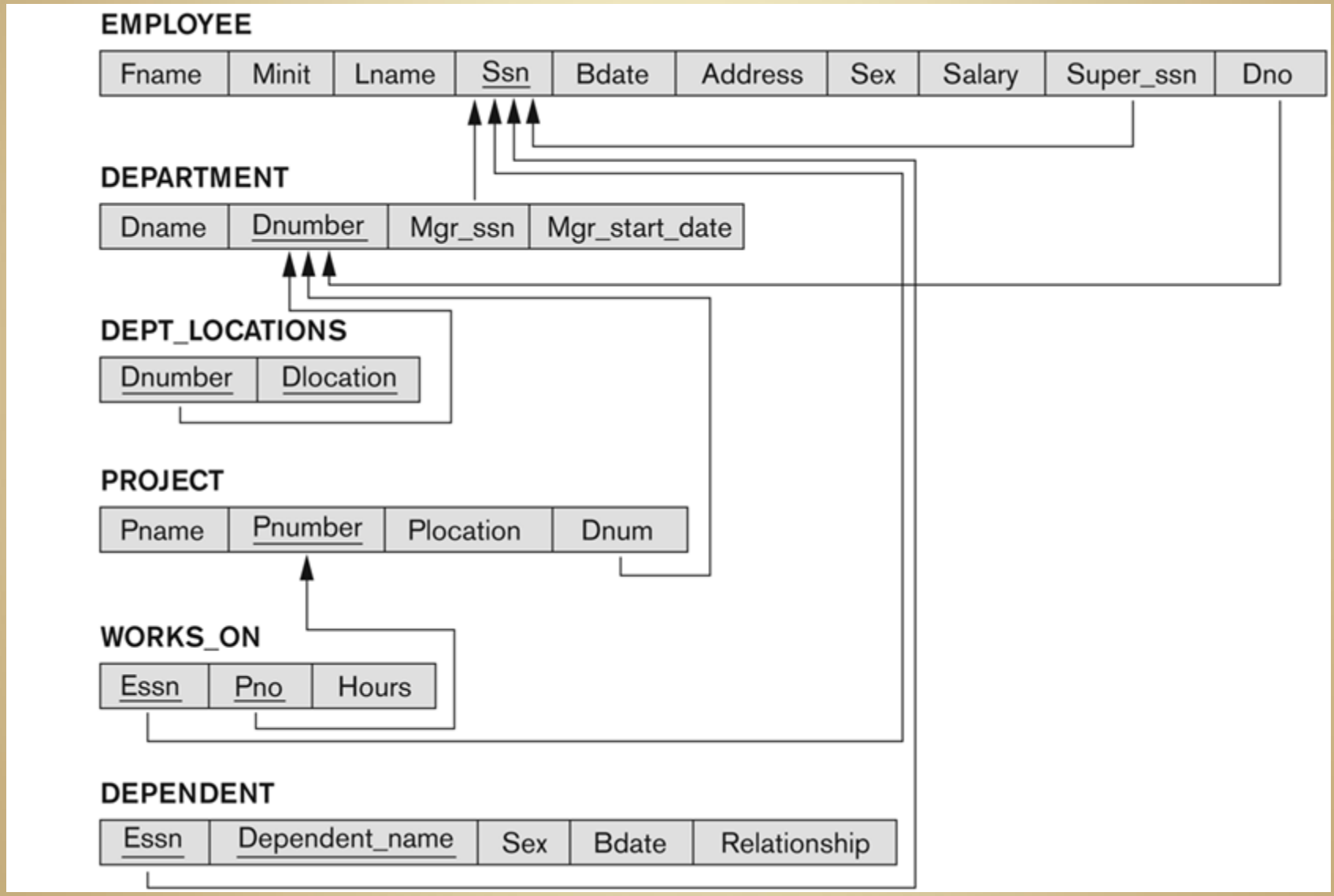
SQL Queries

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.

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

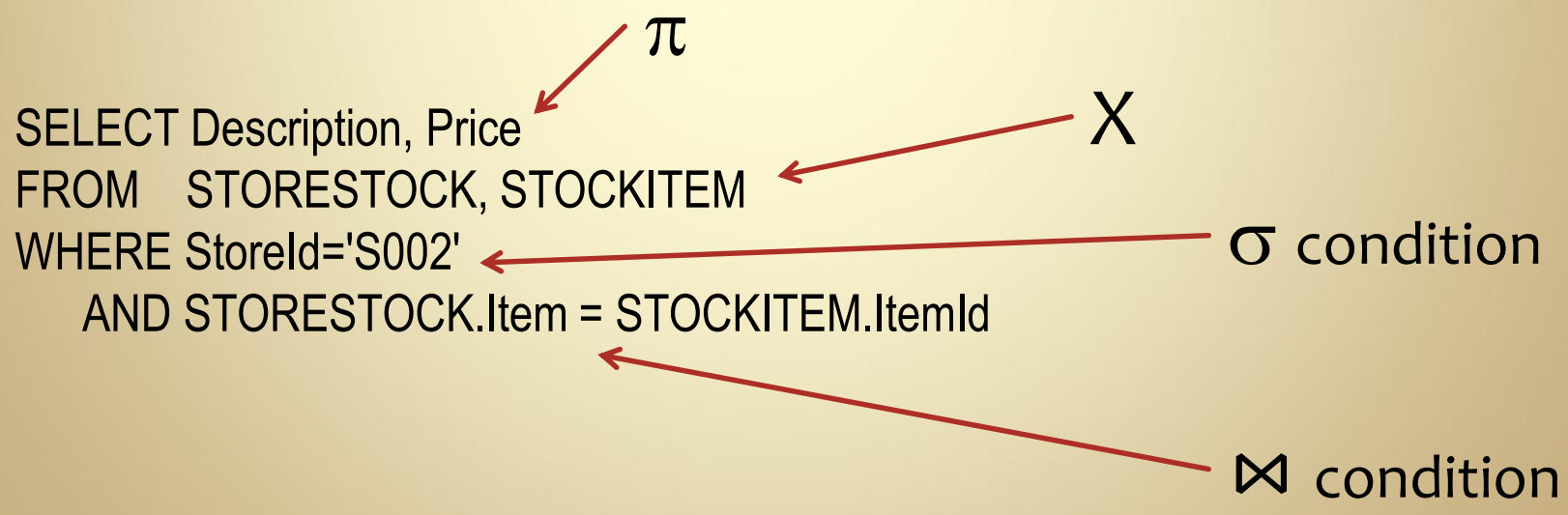
$\sigma_{\text{ItemId=Item}}$ (STOCKITEM × STORESTOCK)

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

SQL Queries

Relational algebra queries are easily translated to SQL queries

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



Declarative Programming

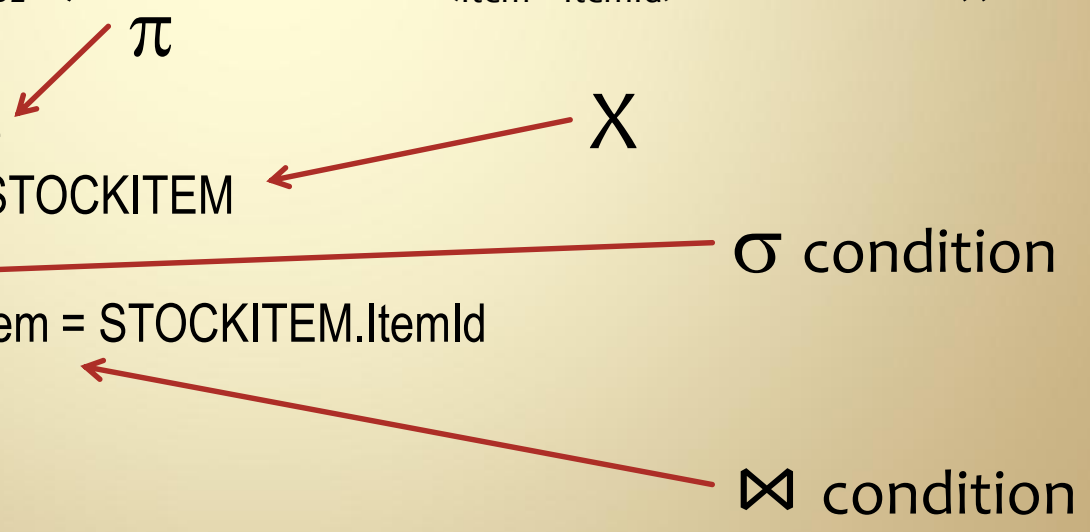
Queries are **declarative**, not **procedural**.

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

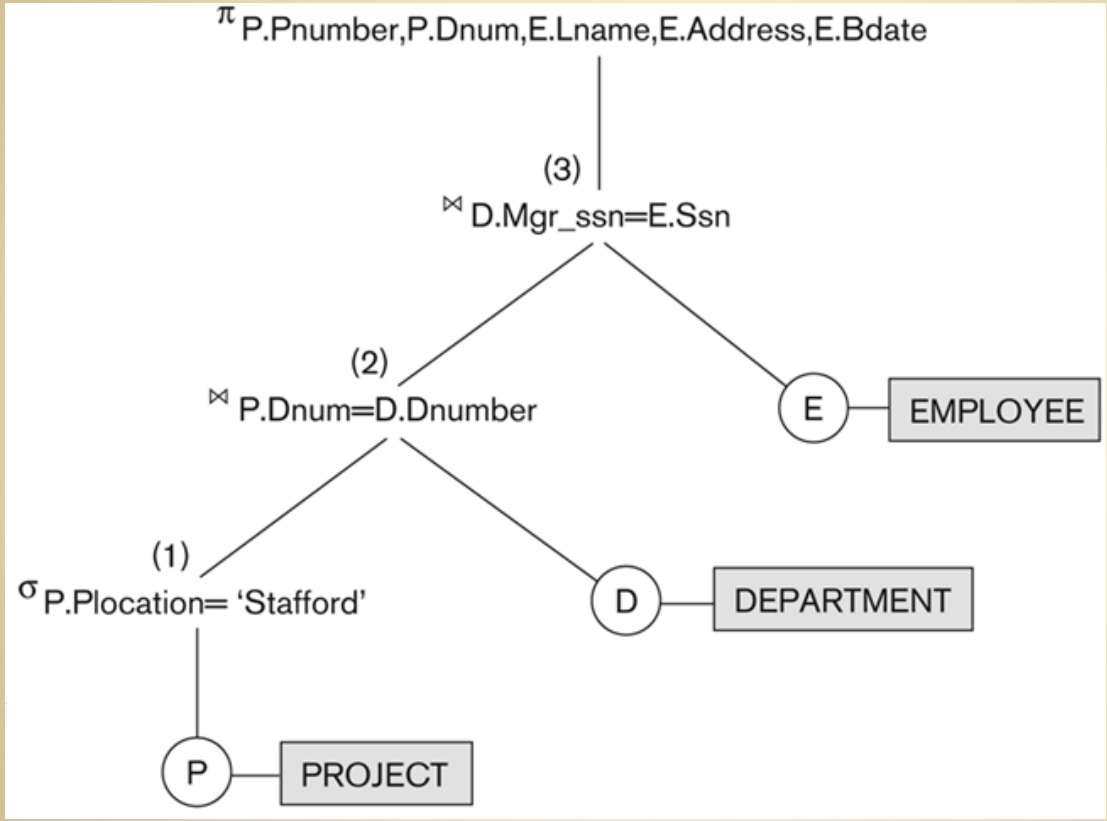
- A few additional operations handle most of these

  - **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

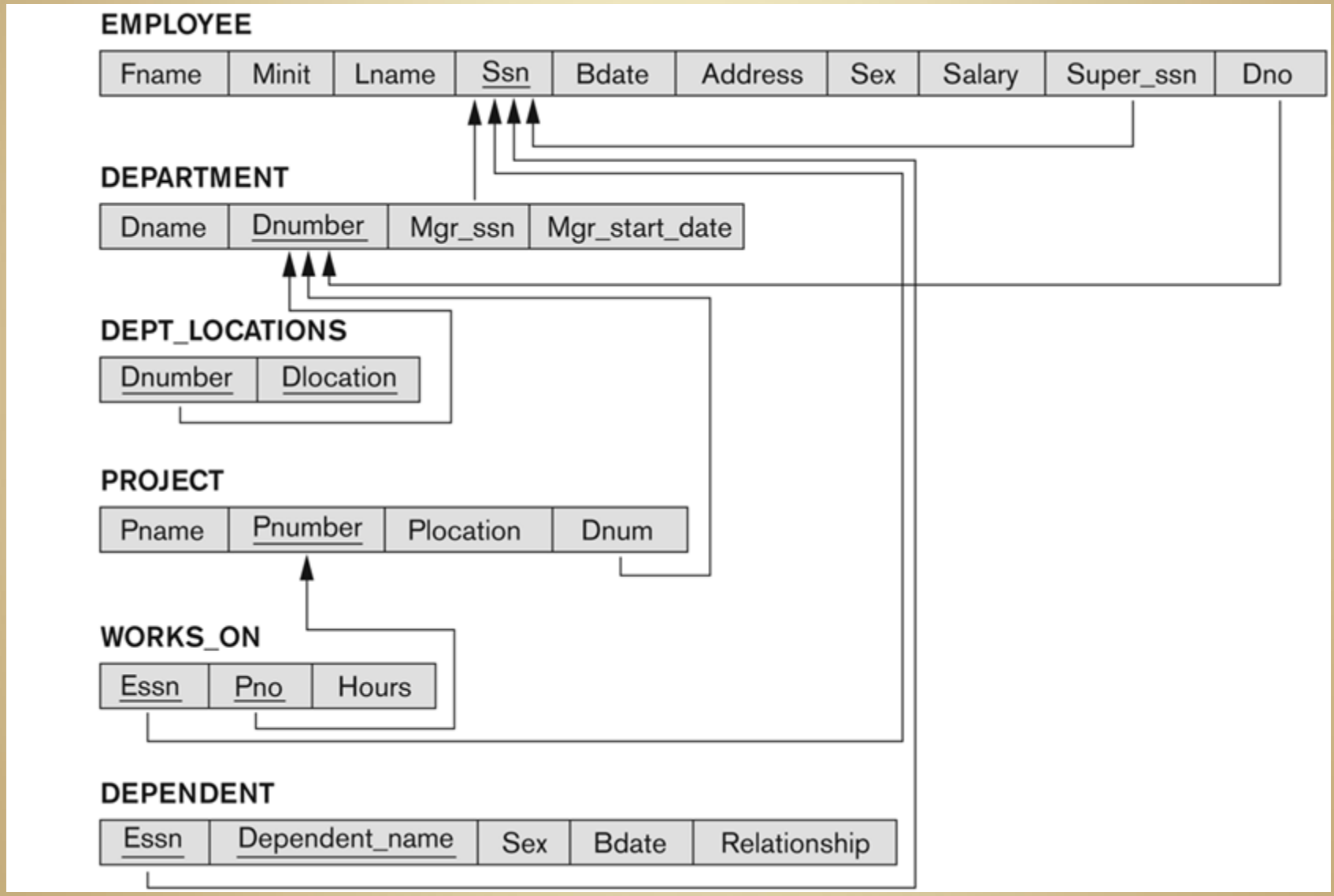
Operation	Purpose	Notation
SELECT	Selects all tuples that satisfy the selection condition from a relation R.	$\sigma_{\text{<selection condition>}}(R)$
PROJECT	Produces a new relation with only some of the attributes of R, and removes duplicate tuples.	$\pi_{\text{<attribute list>}}(R)$
THETA JOIN	Produces 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

AIRPORT

Airport_code	Name	City	State

FLIGHT

Flight_number	Airline	Weekdays

FLIGHT_LEG

Flight_number	Leg_number	Departure_airport_code	Scheduled_departure_time
		Arrival_airport_code	Scheduled_arrival_time

LEG_INSTANCE

Flight_number	Leg_number	Date	Number_of_available_seats	Airplane_id
		Departure_airport_code	Departure_time	Arrival_airport_code	Arrival_time

FARE

Flight_number	Fare_code	Amount	Restrictions

AIRPLANE_TYPE

Airplane_type_name	Max_seats	Company

CAN_LAND

Airplane_type_name	Airport_code

AIRPLANE

Airplane_id	Total_number_of_seats	Airplane_type

SEAT_RESERVATION

Flight_number	Leg_number	Date	Seat_number	Customer_name	Customer_phone

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, 2008.
4. List all airlines that have seats available for flights leaving Los Angeles (LAX) on September 25, 2008.
5. List all airlines that operate at San Jose International Airport (SJC).