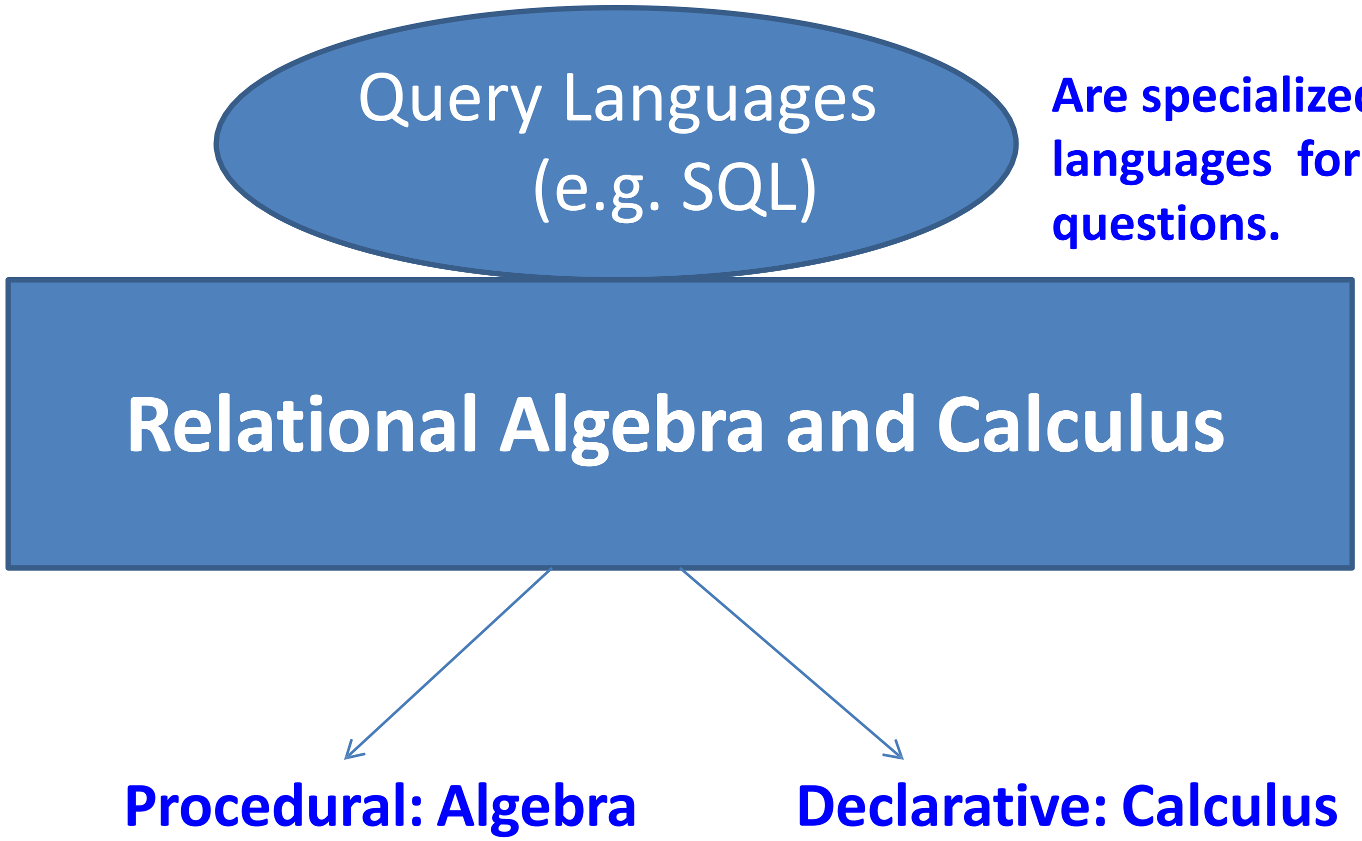
Unit 2

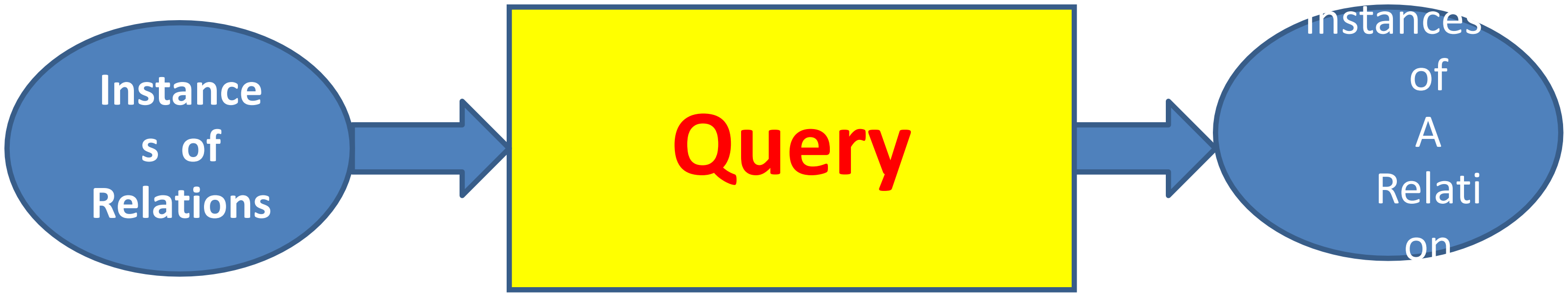
DBMS BTCS 501-18

Relational query languages
Relational algebra

Relational Algebra and Calculus



Are specialized languages for asking questions.


Instances of Relations
Query
Instances of A Relation

Relational Algebra

- Queries are composed using a collection of operators.
- Every operator:
  - Accepts one or two relation instances
  - Returns a relation instance.
- Compose relational algebra expression
- Each query describes a step-by-step procedure for computing the desired answer.

Relational Algebra

- **Five basic operators**
  - **Selection**
  - **Projection**
  - **Union**
  - **Cross-product**
  - **Difference**

Selection

$$\sigma_{Selection\_Criteria}(Input)$$

A relation instance

Manipulates data in a single relation

The selection operator specifies the tuples to retain through selection criteria.

A boolean combination (i.e. using V and Λ) of terms

Attribute op constant or attribute1 op attribute2

$<$, $<=$, $=$, $\neq$, $>=$, or $>$

Selection

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

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

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

Projection

$\pi_{fields}(Input)$

Allows us to extract columns from a relation

E

<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

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

age
35.0
55.5

<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

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

sname	rating
yuppy	9
rusty	10

Set Operations

- Takes as input two relation instances
- Four standard operations
  - Union
  - Intersection
  - Set-difference
  - Cross-product
- Union, intersection, and difference require the two input set to be union compatible
  - They have the same number of fields
  - corresponding fields, taken in order from left to right, have the same domains

Set Operation: Union

- R U S returns relation instance containing all tuples that occur in either relation instance R or S, or both.
- R and S must be union compatible.
- Schema of the result is defined to be that of R.

Set Operation: Union

S1

<u>sid</u>	sname	rating	age
22	dustin	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

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 U S2

Set Operation: Intersection

- R ∩ S: returns a relation instance containing all tuples that occur in both R and S.
- R and S must be union compatible.
- Schema of the result is that of R.

Set Operation: Intersection

S1

<u>sid</u>	sname	rating	age
22	dustin	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

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

S1 ∩ S2

Set Operation: Set-Difference

- R – S: returns a relation instance containing all tuples that occur in R but not in S.
- R and S must be union-compatible.
- Scheme of the result is the schema of R.

Set Operation: Set-Difference

S1

<u>sid</u>	sname	rating	age
22	dustin	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

S1 – S2

sid	sname	rating	age
22	dustin	7	45.0

Set Operation: Cross-Product

- R x S: Returns a relation instance whose scheme contains:
  - All the fields of R (in the same order as they appear in R)
  - All the fields os S (in the same order as they appear in S)
- The result contains one tuple <r,s> for each pair with r ∈ R and s ∈ S
- Basically, it is the Cartesian product.
- Fields of the same name are unnamed.

Set Operation: 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>	<u>bid</u>	<u>day</u>
22	101	10/10/96
58	103	11/12/96

S1 x 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

Renaming

- Name conflict can arise in some situations
- It is convenient to be able to give names to the fields of a relation instance defined by a relational algebra expression.

$$\rho_{S(A1,A2,..)}(R) \text{ or } \rho_{S}(R) \text{ or } \rho_{(A1,A2,..)}(R)$$

- Returns an instance of a new relation S with A1,A2.. Renamed attributes

Renaming

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

(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

sid1	sname	rating	age	sid2	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

Question: Can you define R ∩ S using other operators?

Other Operators?

- We can define any operation using the operators that we have seen.
- Some other operations appear very frequently.
- So they deserve to have their own operators.
  - Join
  - Division

Join

- Can be defined as cross-product followed by selection and projection.
- We have several variants of join.
  - Condition joins
  - Equijoin
  - Natural join

Condition Join /ThetA join

$$R \bowtie_c S = \sigma_c (R \times S)$$

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

(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

(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

Equijoin

$$\boxed{R \bowtie_{c} S}$$

- Condition consists only of equalities connected by ∧
- Redundancy in retaining both attributes in result
- So, an additional projection is applied to remove the second attribute.

Equijoin

Example:

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

(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

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

Natural Join

$S1 \ast R1$

- It is an equijoin in which equalities are specified on all fields having the same name in R and S
- We can then omit the join condition.
- Result is guaranteed not to have two fields with the same name.
- If no fields in common, then natural join is simply cross product.

Division

- Suppose A has two groups of fields <x,y>
- y fields are same fields in terms of domain as B
- A/B = <x> such as for every y value in a tuple of B there is <x,y> in A.

Division

- p2
- *B1*

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

pno

- *B2*

pno
p2
p4

- *B3*

pno
p1
p2
p4

sno
s1
s2
s3
s4

A/B
1

sno
s1
s4

A/B
2

sno
s1

A/B
3

Question: Can we define A/B using the other basic operators?

Disqualified x values:

$$\pi_X((\pi_X(A) \times B) - A)$$

A/B:

$$\pi_X(A) -$$

all disqualified tuples

Examples

Sailors

sid	sname	ratin	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Q1. Find the names of sailors who have reserved boat 103

Solution 1: $\pi_{sname}((\sigma_{bid=103}\text{Reserves}) \bowtie Sailors)$

Solution 2: $\pi_{sname}(\sigma_{bid=103}(\text{Reserves} \bowtie Sailors))$

Examples

Sailors

sid	sname	ratin	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

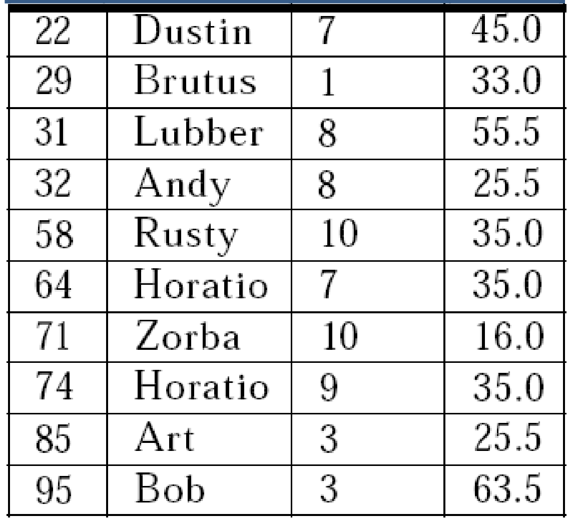
Q2: Find the names of sailors who have reserved a red boat.

Sol1: $\pi_{sname}((\sigma_{color='red'}\, Boats) \bowtie Reserves \bowtie Sailors)$

Sol2: $\pi_{sname}(\pi_{sid}((\pi_{bid}\, \sigma_{color='red'}\, Boats) \bowtie Res) \bowtie Sailors)$

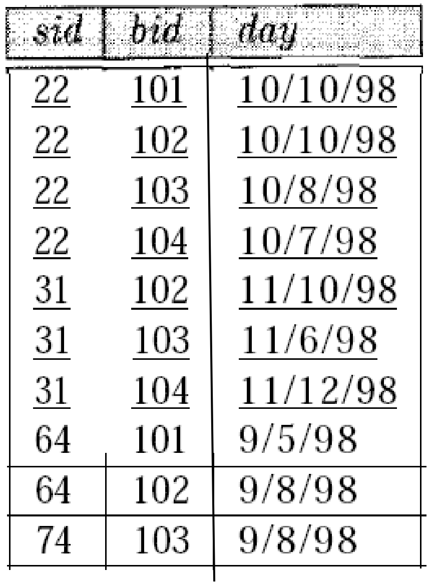
Examples

Sailors



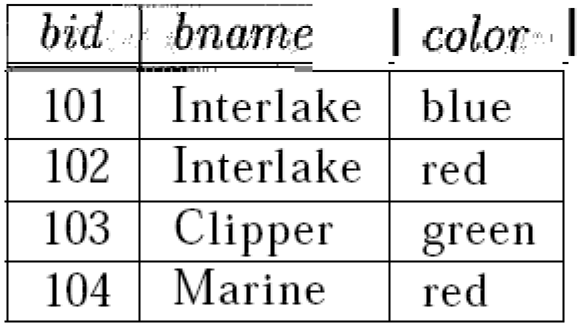
sid	sname	ratin	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves



sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats



bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Q3: Find the colors of boats reserved by Lubber.

$$\pi_{color}((\sigma_{sname='Lubber'} Sailors) \bowtie Reserves \bowtie Boats)$$

Examples

Sailors

sid	sname	ratin	age
22	Dustin	7	45.0
29	Brutus	1	33.0
31	Lubber	8	55.5
32	Andy	8	25.5
58	Rusty	10	35.0
64	Horatio	7	35.0
71	Zorba	10	16.0
74	Horatio	9	35.0
85	Art	3	25.5
95	Bob	3	63.5

Reserves

sid	*bid*	*day*
22	101	10/10/98
22	102	10/10/98
22	103	10/8/98
22	104	10/7/98
31	102	11/10/98
31	103	11/6/98
31	104	11/12/98
64	101	9/5/98
64	102	9/8/98
74	103	9/8/98

Boats

bid	*bname*	*color*
101	Interlake	blue
102	Interlake	red
103	Clipper	green
104	Marine	red

Q5. Find the names of sailors who reserved a red or a green boat.

$$\rho\ (Tempboats, (\sigma_{color='red' \vee color='green'}\ Boats))$$

$$\pi_{sname}(Tempboats \bowtie Reserves \bowtie Sailors)$$

Relational Calculus

- An alternative to relational algebra.
- Declarative
  - describe the set of answers
  - without being explicit about how they should be computed
- One variant is called: tuple relational calculus (TRC).
- Another variant: domain relational calculus (DRC)
- Calculus has variables, constants, comparison ops, logical connectives and quantifiers.

Tuple Relational Calculus

- A TRC query has the form {T | p(T)}
  - T is a tuple variable
  - p(T) is a formula that describes T
- Result: set of all tuples t to which p(T) evaluates to true when T = t
- Example: $\{S \mid S \in Sailors \wedge S.rating > 7\}$

Tuple Relational Calculus

Q: Find the names and ages of sailors with a rating above 7

$$\{P \mid \exists S \in Sailors(S.rating > 7 \land Pname = S.sname \land Page = S.age)\}$$

Q: Find the sailor name, boat id, and reservation date for each reservation.

$$\{P \mid \exists R \in Reserves\ \ \exists S \in Sailors$$
$$(R.sid = 8.sid \land P.bid = R.bid \land P.day = R.day \land P.sname = S.sname)\}$$

Domain Relational Calculus

- **Query** has the form: $\{\langle x1, x2, \ldots, xn \rangle \mid p(\langle x1, x2, \ldots, xn \rangle)\}$
- *Answer* includes all tuples $\langle x1, x2, \ldots, xn \rangle$ that make the *formula* $p(\langle x1, x2, \ldots, xn \rangle)$ be *true*.

Example: Find all sailors with a rating above 7

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

Giving each attribute a variable name

Ensures that I, N, T, and A are restricted to be fields of the same tuple

Algebra Vs Calculus

- Every query that can be expressed in relational algebra can also be expressed in relational calculus.
- The other way around is a bit tricky. Think, for example, about:

$\{S \mid \neg(S \in Sailors)\}$

Conclusions

- Relational algebra and calculus are the foundation of query languages like SQL.
- Queries are expressed by languages like SQL, and the DBMS translates the query into relational algebra.
  - DBMS tries to look for the cheapest relational expression.
- Section 4.2.6 is very useful, pay close attention to it.
- For the calculus part, we will use slides only.

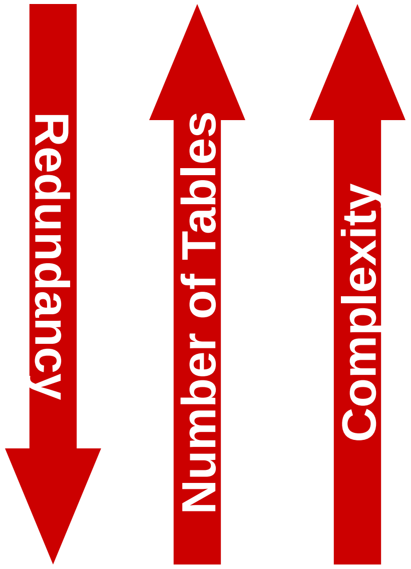
Normal Forms

Definition

- This is the process which allows you to winnow out redundant data within your database.
- This involves restructuring the tables to successively meeting higher forms of Normalization.
- A properly normalized database should have the following characteristics
  - Scalar values in each fields
  - Absence of redundancy.
  - Minimal use of null values.
  - Minimal loss of information.

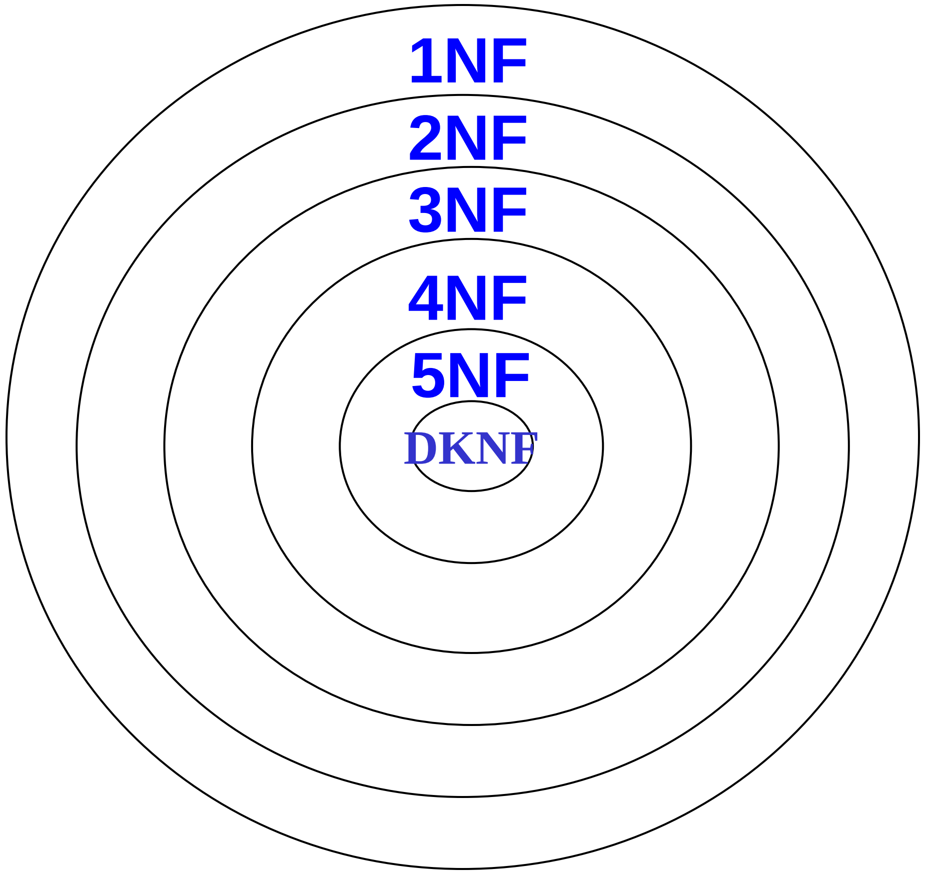
Levels of Normalization

- Levels of normalization based on the amount of redundancy in the database.
- Various levels of normalization are:
  - First Normal Form (1NF)
  - Second Normal Form (2NF)
  - Third Normal Form (3NF)
  - Boyce-Codd Normal Form (BCNF)
  - Fourth Normal Form (4NF)
  - Fifth Normal Form (5NF)
  - Domain Key Normal Form (DKNF)



Most databases should be 3NF or BCNF in order to avoid the database anomalies.

Levels of Normalization



Each higher level is a subset of the lower level

First Normal Form (1NF)

A table is considered to be in 1NF if all the fields contain only scalar values (as opposed to list of values).

Example (Not 1NF)

ISBN	Title	AuName	AuPhone	PubName	PubPhone	Price
0-321-32132-1	Balloon	Sleepy, Snoopy, Grumpy	321-321-1111, 232-234-1234, 665-235-6532	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Jones, Smith	123-333-3333, 654-223-3455	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Joyce	666-666-6666	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Roman	444-444-4444	Big House	123-456-7890	$25.00

Author and AuPhone columns are not scalar

1NF - Decomposition

1. Place all items that appear in the repeating group in a new table
2. Designate a primary key for each new table produced.
3. Duplicate in the new table the primary key of the table from which the repeating group was extracted or vice versa.

Example (1NF)

ISBN	Title	PubName	PubPhone	Price
0-321-32132-1	Balloon	Small House	714-000-0000	$34.00
0-55-123456-9	Main Street	Small House	714-000-0000	$22.95
0-123-45678-0	Ulysses	Alpha Press	999-999-9999	$34.00
1-22-233700-0	Visual Basic	Big House	123-456-7890	$25.00

ISBN	AuName	AuPhone
0-321-32132-1	Sleepy	321-321-1111
0-321-32132-1	Snoopy	232-234-1234
0-321-32132-1	Grumpy	665-235-6532
0-55-123456-9	Jones	123-333-3333
0-55-123456-9	Smith	654-223-3455
0-123-45678-0	Joyce	666-666-6666
1-22-233700-0	Roman	444-444-4444

Functional Dependencies

1. If one set of attributes in a table determines another set of attributes in the table, then the second set of attributes is said to be functionally dependent on the first set of attributes.

Example 1

ISBN	Title	Price
0-321-32132-1	Balloon	$34.00
0-55-123456-9	Main Street	$22.95
0-123-45678-0	Ulysses	$34.00
1-22-233700-0	Visual Basic	$25.00

Table Scheme: {ISBN, Title, Price}

Functional Dependencies: {ISBN} → {Title}

{ISBN} → {Price}

Functional Dependencies

Example 2

PubID	PubName	PubPhone
1	Big House	999-999-9999
2	Small House	123-456-7890
3	Alpha Press	111-111-1111

Table Scheme: {PubID, PubName, PubPhone}

Functional Dependencies: {PubId} → {PubPhone}

{PubId} → {PubName}

{PubName, PubPhone} → {PubID}

Example 3

AuID	AuName	AuPhone
1	Sleepy	321-321-1111
2	Snoopy	232-234-1234
3	Grumpy	665-235-6532
4	Jones	123-333-3333
5	Smith	654-223-3455
6	Joyce	666-666-6666
7	Roman	444-444-4444

Table Scheme: {AuID, AuName, AuPhone}

Functional Dependencies: {AuId} → {AuPhone}

{AuId} → {AuName}

{AuName, AuPhone} → {AuID}

FD – Example

Database to track reviews of papers submitted to an academic conference. Prospective authors submit papers for review and possible acceptance in the published conference proceedings. Details of the entities

- Author information includes a unique author number, a name, a mailing address, and a unique (optional) email address.
- Paper information includes the primary author, the paper number, the title, the abstract, and review status (pending, accepted,rejected)
- Reviewer information includes the reviewer number, the name, the mailing address, and a unique (optional) email address
- A completed review includes the reviewer number, the date, the paper number, comments to the authors, comments to the program chairperson, and ratings (overall, originality, correctness, style, clarity)

FD – Example

Functional Dependencies

- AuthNo → AuthName, AuthEmail, AuthAddress
- AuthEmail → AuthNo
- PaperNo → Primary-AuthNo, Title, Abstract, Status
- RevNo → RevName, RevEmail, RevAddress
- RevEmail → RevNo
- RevNo, PaperNo → AuthComm, Prog-Comm, Date, Rating1, Rating2, Rating3, Rating4, Rating5

Second Normal Form (2NF)

For a table to be in 2NF, there are two requirements

- The database is in first normal form
- All **nonkey** attributes in the table must be functionally dependent on the entire primary key

Note: *Remember that we are dealing with non-key attributes*

Example 1 (Not 2NF)

Scheme → {Title, PubId, AuId, Price, AuAddress}

1. **Key → {Title, PubId, AuId}**
2. **{Title, PubId, AuID} → {Price}**
3. **{AuID} → {AuAddress}**
4. **AuAddress does not belong to a key**
5. **AuAddress functionally depends on AuId which is a subset of a key**

Second Normal Form (2NF)

Example 2 (Not 2NF)

Scheme → {City, Street, HouseNumber, HouseColor, CityPopulation}

1. **key → {City, Street, HouseNumber}**
2. **{City, Street, HouseNumber} → {HouseColor}**
3. **{City} → {CityPopulation}**
4. **CityPopulation does not belong to any key.**
5. **CityPopulation is functionally dependent on the City which is a proper subset of the key**

Example 3 (Not 2NF)

Scheme → {studio, movie, budget, studio_city}

1. **Key → {studio, movie}**
2. **{studio, movie} → {budget}**
3. **{studio} → {studio_city}**
4. **studio_city is not a part of a key**
5. **studio_city functionally depends on studio which is a proper subset of the key**

2NF - Decomposition

1. If a data item is fully functionally dependent on only a part of the primary key, move that data item and that part of the primary key to a new table.
2. If other data items are functionally dependent on the same part of the key, place them in the new table also
3. Make the partial primary key copied from the original table the primary key for the new table. Place all items that appear in the repeating group in a new table

Example 1 (Convert to 2NF)

Old Scheme → {Title, PubId, AuId, Price, AuAddress}

New Scheme → {Title, PubId, AuId, Price}

New Scheme → {AuId, AuAddress}

2NF - Decomposition

Example 2 (Convert to 2NF)

Old Scheme → {<u>Studio</u>, <u>Movie</u>, Budget, StudioCity}

New Scheme → {<u>Movie</u>, <u>Studio</u>, Budget}

New Scheme → {<u>Studio</u>, City}

Example 3 (Convert to 2NF)

Old Scheme → {<u>City</u>, <u>Street</u>, <u>HouseNumber</u>, HouseColor, CityPopulation}

New Scheme → {<u>City</u>, <u>Street</u>, <u>HouseNumber</u>, HouseColor}

New Scheme → {<u>City</u>, CityPopulation}

Third Normal Form (3NF)

This form dictates that all **non-key** attributes of a table must be functionally dependent on a candidate key i.e. there can be no interdependencies among non-key attributes.

For a table to be in 3NF, there are two requirements

- The table should be second normal form
- No attribute is transitively dependent on the primary key

Example (Not in 3NF)

Scheme → {Title, PubID, PageCount, Price }

1. **Key → {Title, PubId}**
2. **{Title, PubId} → {PageCount}**
3. **{PageCount} → {Price}**
4. **Both Price and PageCount depend on a key hence 2NF**
5. **Transitively {Title, PubID} → {Price} hence not in 3NF**

Third Normal Form (3NF)

Example 2 (Not in 3NF)

Scheme → {<u>Studio</u>, StudioCity, CityTemp}

1. **Primary Key → {Studio}**
2. **{Studio} → {StudioCity}**
3. **{StudioCity} → {CityTemp}**
4. **{Studio} → {CityTemp}**
5. **Both StudioCity and CityTemp depend on the entire key hence 2NF**
6. **CityTemp transitively depends on Studio hence violates 3NF**

BuildingID	**Contractor**	**Fee**
100	Randolph	1200
150	Ingersoll	1100
200	Randolph	1200
250	Pitkin	1100
300	Randolph	1200

Example 3 (Not in 3NF)

Scheme → {BuildingID, Contractor, Fee}

1. **Primary Key → {BuildingID}**
2. **{BuildingID} → {Contractor}**
3. **{Contractor} → {Fee}**
4. **{BuildingID} → {Fee}**
5. **Fee transitively depends on the BuildingID**
6. **Both Contractor and Fee depend on the entire key hence 2NF**

3NF - Decomposition

1. Move all items involved in transitive dependencies to a new entity.
2. Identify a primary key for the new entity.
3. Place the primary key for the new entity as a foreign key on the original entity.

Example 1 (Convert to 3NF)

Old Scheme → {Title, PubID, PageCount, Price }

New Scheme → {PubID, PageCount, Price}

New Scheme → {Title, PubID, PageCount}

3NF - Decomposition

Example 2 (Convert to 3NF)

Old Scheme → {<u>Studio</u>, StudioCity, CityTemp}

New Scheme → {<u>Studio</u>, StudioCity}

New Scheme → {<u>StudioCity</u>, CityTemp}

Example 3 (Convert to 3NF)

Old Scheme → {BuildingID, Contractor, Fee}

New Scheme → {BuildingID, Contractor}

New Scheme → {Contractor, Fee}

BuildingID	**Contractor**
100	Randolph
150	Ingersoll
200	Randolph
250	Pitkin
300	Randolph

Contractor	**Fee**
Randolph	1200
Ingersoll	1100
Pitkin	1100

Boyce-Codd Normal Form (BCNF)

- BCNF does not allow dependencies between attributes that belong to candidate keys.
- BCNF is a refinement of the third normal form in which it drops the restriction of a non-key attribute from the 3rd normal form.
- Third normal form and BCNF are not same if the following conditions are true:
  - The table has two or more candidate keys
  - At least two of the candidate keys are composed of more than one attribute
  - The keys are not disjoint i.e. The composite candidate keys share some attributes

Example 1 - Address (Not in BCNF)

Scheme → {City, Street, ZipCode }

1. **Key1 → {City, Street }**
2. **Key2 → {ZipCode, Street}**
3. **No non-key attribute hence 3NF**
4. **{City, Street} → {ZipCode}**
5. **{ZipCode} → {City}**
6. **Dependency between attributes belonging to a key**

Boyce Codd Normal Form (BCNF)

Example 2 - Movie (Not in BCNF)

Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

1. Key1 → {MovieTitle, PersonName}
2. Key2 → {MovieID, PersonName}
3. Both role and payment functionally depend on both candidate keys thus 3NF
4. {MovieID} → {MovieTitle}
5. Dependency between MovieID & MovieTitle Violates BCNF

Example 3 - Consulting (Not in BCNF)

Scheme → {Client, Problem, Consultant}

1. Key1 → {Client, Problem}
2. Key2 → {Client, Consultant}
3. No non-key attribute hence 3NF
4. {Client, Problem} → {Consultant}
5. {Client, Consultant} → {Problem}
6. Dependency between attributess belonging to keys violates BCNF

BCNF - Decomposition

1. Place the two candidate primary keys in separate entities
2. Place each of the remaining data items in one of the resulting entities according to its dependency on the primary key.

Example 1 (Convert to BCNF)

Old Scheme → {City, Street, ZipCode }

New Scheme1 → {ZipCode, Street}

New Scheme2 → {City, Street}

- **Loss of relation {ZipCode} → {City}**

Alternate New Scheme1 → {ZipCode, Street }

Alternate New Scheme2 → {ZipCode, City}

Decomposition – Loss of Information

1. If decomposition does not cause any loss of information it is called a **lossless** decomposition.
2. If a decomposition does not cause any dependencies to be lost it is called a **dependency-preserving** decomposition.
3. Any table scheme can be decomposed in a lossless way into a collection of smaller schemas that are in BCNF form. However the dependency preservation is not guaranteed.
4. Any table can be decomposed in a lossless way into 3rd normal form that also preserves the dependencies.
   - 3NF may be better than BCNF in some cases

Use your own judgment when decomposing schemas

BCNF - Decomposition

Example 2 (Convert to BCNF)

Old Scheme → {MovieTitle, MovieID, PersonName, Role, Payment }

New Scheme → {MovieID, PersonName, Role, Payment}

New Scheme → {MovieTitle, PersonName}

- Loss of relation {MovieID} → {MovieTitle}

New Scheme → {MovieID, PersonName, Role, Payment}

New Scheme → {MovieID, MovieTitle}

- We got the {MovieID} → {MovieTitle} relationship back

Example 3 (Convert to BCNF)

Old Scheme → {Client, Problem, Consultant}

New Scheme → {Client, Consultant}

New Scheme → {Client, Problem}

Fourth Normal Form (4NF)

- **Fourth normal form eliminates independent many-to-one relationships between columns.**
- **To be in Fourth Normal Form,**
  - **a relation must first be in Boyce-Codd Normal Form.**
  - **a given relation may not contain more than one multi-valued attribute.**

Example (Not in 4NF)

Scheme → {MovieName, ScreeningCity, Genre)

Primary Key: {MovieName, ScreeningCity, Genre)

1. **All columns are a part of the only candidate key, hence BCNF**
2. **Many Movies can have the same Genre**
3. **Many Cities can have the same movie**
4. **Violates 4NF**

Movie	**ScreeningCity**	**Genre**
Hard Code	Los Angles	Comedy
Hard Code	New York	Comedy
Bill Durham	Santa Cruz	Drama
Bill Durham	Durham	Drama
The Code Warrier	New York	Horror

Fourth Normal Form (4NF)

Example 2 (Not in 4NF)

Scheme → {Manager, Child, Employee}

1. Primary Key → {Manager, Child, Employee}
2. Each manager can have more than one child
3. Each manager can supervise more than one employee
4. 4NF Violated

Manager	Child	Employee
Jim	Beth	Alice
Mary	Bob	Jane
Mary	NULL	Adam

Example 3 (Not in 4NF)

Scheme → {Employee, Skill, ForeignLanguage}

1. Primary Key → {Employee, Skill, Language }
2. Each employee can speak multiple languages
3. Each employee can have multiple skills
4. Thus violates 4NF

Employee	Skill	Language
1234	Cooking	French
1234	Cooking	German
1453	Carpentry	Spanish
1453	Cooking	Spanish
2345	Cooking	Spanish

4NF - Decomposition

1. Move the two multi-valued relations to separate tables
2. Identify a primary key for each of the new entity.

Example 1 (Convert to 3NF)

Old Scheme → {MovieName, ScreeningCity, Genre}

New Scheme → {MovieName, ScreeningCity}

New Scheme → {MovieName, Genre}

Movie	**Genre**
Hard Code	Comedy
Bill Durham	Drama
The Code Warrier	Horror

Movie	**ScreeningCity**
Hard Code	Los Angles
Hard Code	New York
Bill Durham	Santa Cruz
Bill Durham	Durham
The Code Warrier	New York

4NF - Decomposition

Example 2 (Convert to 4NF)

Old Scheme → {Manager, Child, Employee}

New Scheme → {<u>Manager, Child</u>}

New Scheme → {<u>Manager, Employee</u>}

Manager	**Child**
Jim	Beth
Mary	Bob

Manager	**Employee**
Jim	Alice
Mary	Jane
Mary	Adam

Example 3 (Convert to 4NF)

Old Scheme → {Employee, Skill, ForeignLanguage}

New Scheme → {Employee, Skill}

New Scheme → {Employee, ForeignLanguage}

Employee	**Skill**
1234	Cooking
1453	Carpentry
1453	Cooking
2345	Cooking

Employee	**Language**
1234	French
1234	German
1453	Spanish
2345	Spanish

Fifth Normal Form (5NF)

- Fifth normal form is satisfied when all tables are broken into as many tables as possible in order to avoid redundancy. Once it is in fifth normal form it cannot be broken into smaller relations without changing the facts or the meaning.

Domain Key Normal Form (DKNF)

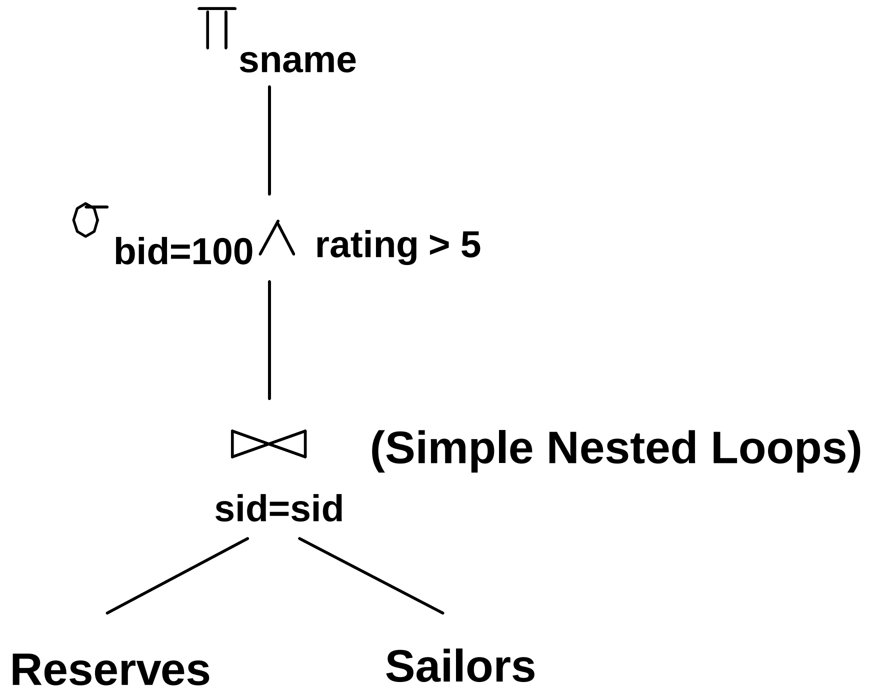
- The relation is in DKNF when there can be no insertion or deletion anomalies in the database.

Query Optimization

Goal:

Declarative SQL query $\longrightarrow$ *Imperative query execution plan:*

```
SELECT  S.sname
FROM  Reserves R, Sailors S
WHERE  R.sid=S.sid AND
   R.bid=100 AND S.rating>5
```



π_{sname}
$\sigma_{\text{bid=100} \wedge \text{rating > 5}}$
$\bowtie_{\text{sid=sid}}$
(Simple Nested Loops)
Reserves
Sailors

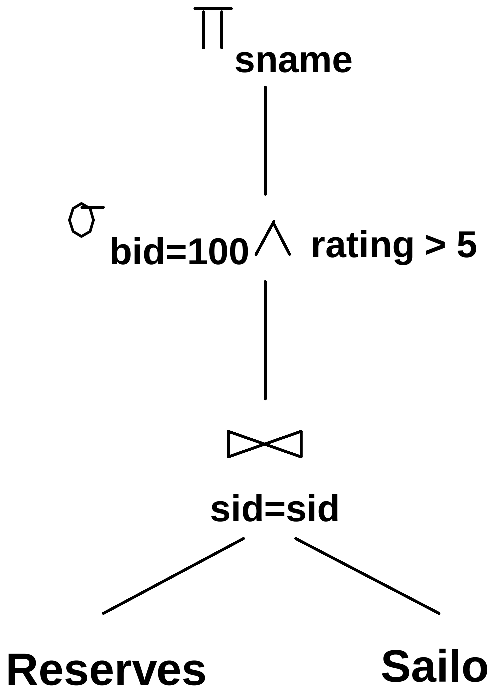
Plan: ***Tree of R.A. ops, with choice of alg for e***

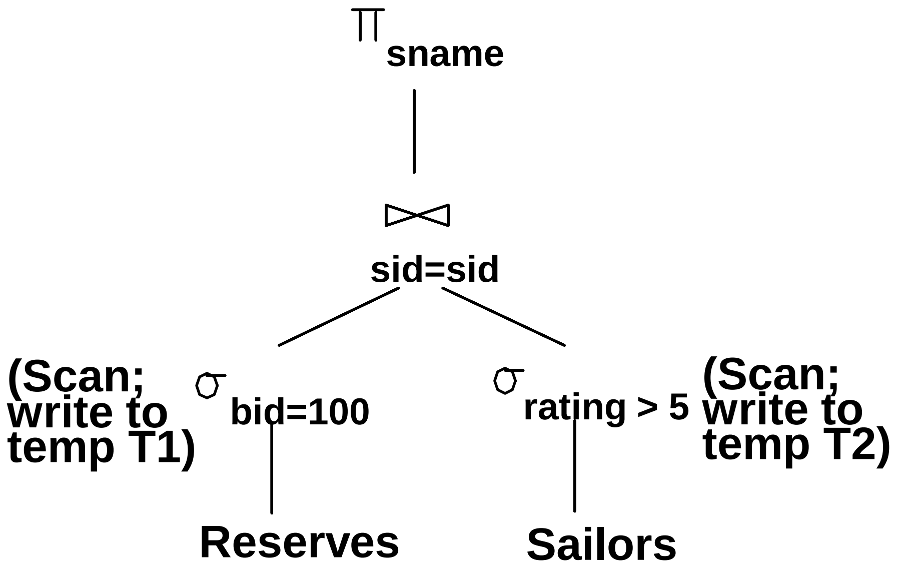
Ideally: Want to find best plan. Practically: Avoid worst plans!

Query Optimization Issues

- Query rewriting:
  - transformations from one SQL query to another one using semantic properties.
- Selecting query execution plan:
  - done on single query blocks (I.e., S-P-J blocks)
  - main step: join enumeration
- Cost estimation:
  - to compare between plans we need to estimate their cost using statistics on the database.

Query Rewriting: Predicate Pushdown





The earlier we process selections, less tuples we need to ma
higher up in the tree (but may cause us to loose an important
of the tuples.

Query Rewrites: Predicate Pushdown (more complicated)

```
Select   bid, Max(age)
From     Reserves R, Sailors S
Where    R.sid=S.sid
GroupBy  bid
Having Max(age) > 40
```

```
Select   bid, Max(age)
From     Reserves R, Sailors S
Where    R.sid=S.sid  and
         S.age > 40
GroupBy  bid
Having Max(age) > 40
```

- Advantage: the size of the join will be smaller.
- Requires transformation rules specific to the grouping/aggr operators.

- Won't work if we replace Max by Min.

Query Rewrite: Predicate Movearound

Sailing wizz dates: when did the youngest of each sailor level

```
Select   sid, date
From     V1, V2
Where    V1.rating = V2.rating  and
         V1.age = V2.age
```

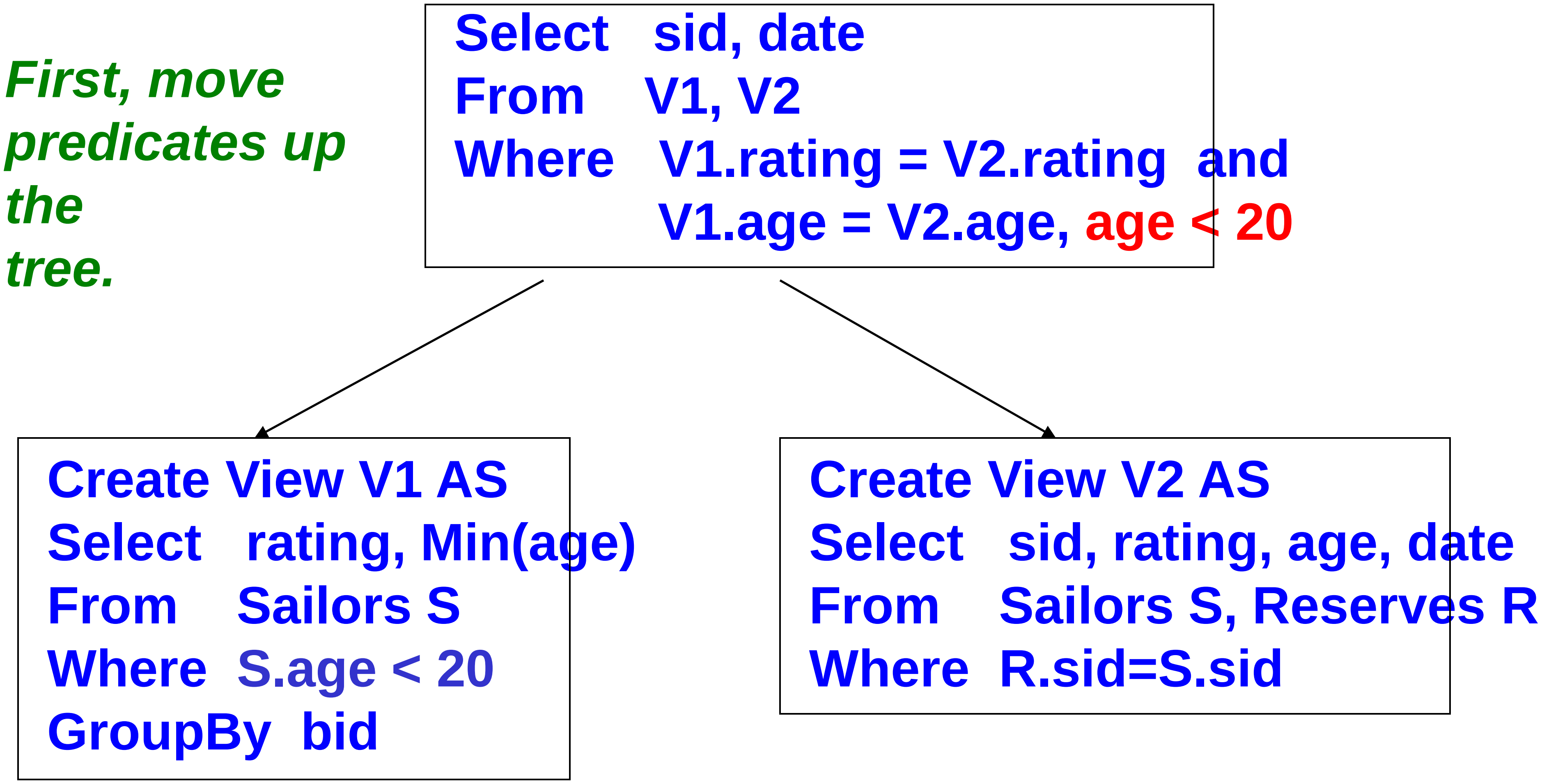
```
Create View V1 AS
Select   rating, Min(age)
From     Sailors S
Where    S.age < 20
GroupBy  bid
```

```
Create View V2 AS
Select   sid, rating, age, date
From     Sailors S, Reserves R
Where    R.sid=S.sid
```

Query Rewrite: Predicate Movearound

Sailing wizz dates: when did the youngest of each sailor level rent boats?

First, move predicates up the tree.



```
Select   sid, date
From     V1, V2
Where    V1.rating = V2.rating  and
         V1.age = V2.age, age < 20
```

```
Create View V1 AS
Select   rating, Min(age)
From     Sailors S
Where    S.age < 20
GroupBy  bid
```

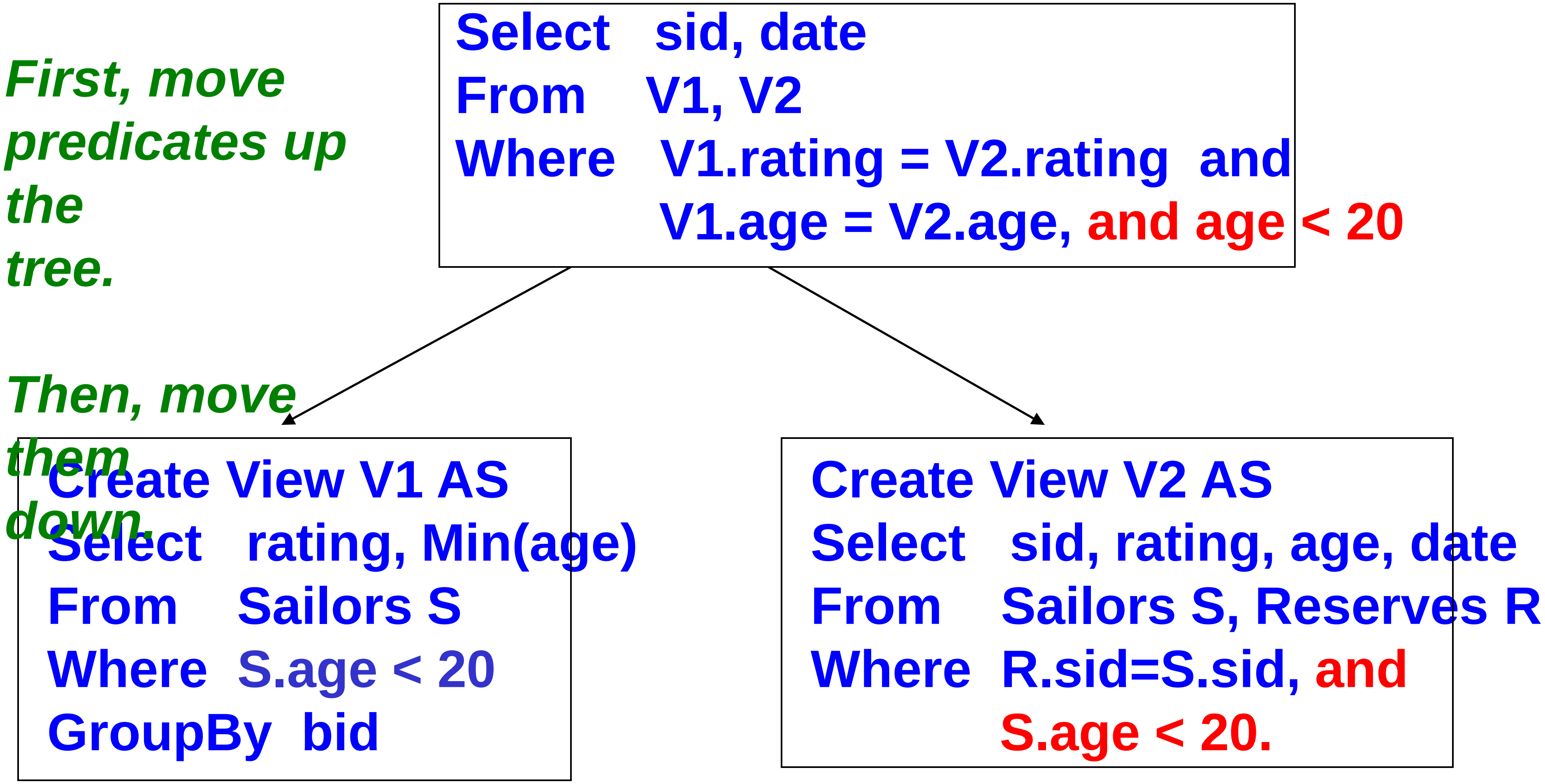
```
Create View V2 AS
Select   sid, rating, age, date
From     Sailors S, Reserves R
Where    R.sid=S.sid
```

Query Rewrite: Predicate Movearound

Sailing wizz dates: when did the youngest of each sailor level rent boats?

First, move predicates up the tree.

Then, move them down.



```
Select   sid, date
From     V1, V2
Where    V1.rating = V2.rating  and
         V1.age = V2.age, and age < 20
```

```
Create View V1 AS
Select   rating, Min(age)
From     Sailors S
Where    S.age < 20
GroupBy  bid
```

```
Create View V2 AS
Select   sid, rating, age, date
From     Sailors S, Reserves R
Where    R.sid=S.sid, and
         S.age < 20.
```

Query Rewrite Summary

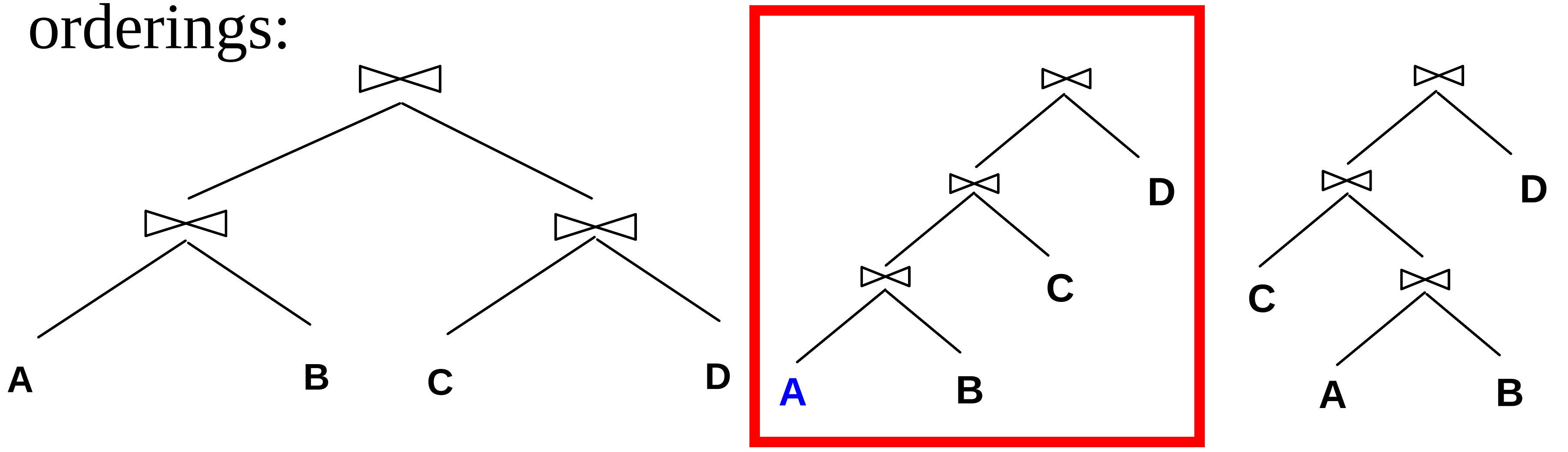
- The optimizer can use any *semantically correct* rule to transform one query to another.
- Rules try to:
  - move constraints between blocks (because each will be optimized separately)
  - Unnest blocks
- Especially important in decision support applications where queries are very complex.

Enumeration of Alternative Plans

- Task: create a query execution plan for a single Select-project-join block (well, and aggregates).
- Main principle: some sort of search through the set of plans.
  - Assume some cost estimation model; more later.
- Single-relation block case (only select, project, aggregation):
  - Each available access path is considered, and the one with the least estimated cost is chosen.
  - The different operations are essentially carried out together (e.g., if an index is used for a selection, projection is done for each retrieved tuple, and the resulting tuples are *pipelined* into the aggregate computation).

Queries Over Multiple Relations

- In principle, we need to consider all possible join orderings:



- As the number of joins increases, the number of alternative plans grows rapidly; *we need to restrict the search space.*
- System-R: consider *only left-deep join trees*.
  - Left-deep trees allow us to generate all *fully pipelined* plans:Intermediate results not written to temporary files.
    - Not all left-deep trees are fully pipelined (e.g., SM join).

Enumeration of Left-Deep Plans

- Enumerated using N passes (if N relations joined):
  - Pass 1: Find best 1-relation plan for each relation.
  - Pass 2: Find best way to join result of each 1-relation plan (as outer) to another relation. *(All 2-relation plans.)*
  - Pass N: Find best way to join result of a (N-1)-relation plan (as outer) to the N'th relation. *(All N-relation plans.)*
- For each subset of relations, retain only:
  - Cheapest plan overall, plus
  - Cheapest plan for each *interesting order* of the tuples.

Enumeration of Plans (Contd.)

- ORDER BY, GROUP BY, aggregates etc. handled as a final step, using either an `interestingly ordered' plan or an additional sorting operator.
- An N-1 way plan is not combined with an additional relation unless there is a join condition between them, unless all predicates in WHERE have been used up.
  - i.e., avoid Cartesian products if possible.
- In spite of pruning plan space, this approach is still exponential in the # of tables.
- If we want to consider all (bushy) trees, we need only a slight modification to the algorithm.

Sailors:

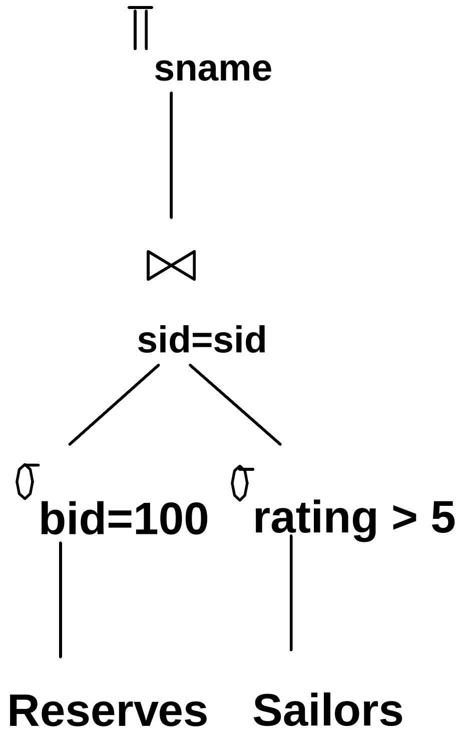
B+ tree on *rating*

Hash on *sid*

Reserves:

B+ tree on *bid*

Example



- Pass 1:
  - *Sailors*: B+ tree matches *rating>5*, and is probably cheapest. However, if this selection is expected to retrieve a lot of tuples, and index is unclustered, file scan may be cheaper.
    - Still, B+ tree plan kept (tuples are in *rating* order).
  - *Reserves*: B+ tree on *bid* matches *bid=100*; cheapest.
- Pass 2: We consider each plan retained from Pass 1 as the outer, and consider how to join it with the (only) other relation.
  - e.g., *Reserves as outer*: Hash index can be used to get Sailors tuples that satisfy *sid* = outer tuple's *sid* value.

Nested Queries

- Nested block is optimized independently, with the outer tuple considered as providing a selection condition.
- Outer block is optimized with the cost of `calling' nested block computation taken into account.
- Implicit ordering of these blocks means that some good strategies are not considered. *The non-nested version of the query is typically optimized better.*

```
SELECT  S.sname
FROM  Sailors S
WHERE EXISTS
   (SELECT  *
    FROM  Reserves R
    WHERE
R.bid=103
    AND  R.sid=S.sid)
```

Nested block to optimize:

```
SELECT  *
FROM  Reserves R
WHERE  R.bid=103
   AND  S.sid= outer
value
```

Equivalent non-nested query:

```
SELECT  S.sname
FROM Sailors S, Reserves R
WHERE  S.sid=R.sid
  AND R.bid=103
```

Cost Estimation

- For each plan considered, must estimate cost:
  - Must estimate *cost* of each operation in plan tree.
    - Depends on input cardinalities.
  - Must estimate *size of result* for each operation in tree!
    - Use information about the input relations.
    - For selections and joins, assume independence of predicates.
- We'll discuss the System R cost estimation approach.
  - Very inexact, but works ok in practice.
  - More sophisticated techniques known now.

Statistics and Catalogs

- Need information about the relations and indexes involved. ***Catalogs*** typically contain at least:
  - # tuples (NTuples) and # pages (NPages) for each relation.
  - # distinct key values (NKeys) and NPages for each index.
  - Index height, low/high key values (Low/High) for each tree index.
- Catalogs updated periodically.
  - Updating whenever data changes is too expensive; lots of approximation anyway, so slight inconsistency ok.
- More detailed information (e.g., histograms of the values in some field) are sometimes stored.

Size Estimation and Reduction Factors

```
SELECT  attribute list
FROM  relation list
WHERE  term1 AND ... AND termk
```

- Consider a query block:
- Maximum # tuples in result is the product of the cardinalities of relations in the FROM clause.
- *Reduction factor (RF)* associated with each *term* reflects the impact of the *term* in reducing result size. *Result cardinality* = Max # tuples * product of all RF's.
  - Implicit assumption that *terms* are independent!
  - Term *col=value* has RF *1/NKeys(I)*, given index I on *col*
  - Term *col1=col2* has RF *1/MAX(NKeys(I1), NKeys(I2))*
  - Term *col>value* has RF *(High(I)-value)/(High(I)-Low(I))*